Unidirectional Links in Wireless Sensor Networks

Problems and Solutions for MAC- and Routing Layer

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Diplom Informatiker

Reinhardt Karnapke

geboren am 08.03.1978 in Berlin

Gutachter: Prof. Dr.-Ing. Jörg Nolte

Gutachter: Prof. Dr.-Ing. Rolf Kraemer

Gutachter: Prof. Dr.-Ing. Jochen H. Schiller

Tag der mündlichen Prüfung: 07.12.2012

Abstract

Experiments with wireless sensor networks have shown that unidirectional communication links are quite common. What is even more, they have also shown that the range of a unidirectional link can exceed that of a bidirectional one by far. Still, most of todays routing protocols do not use them, they only eliminate their implications. Those protocols that do use unidirectional links introduce a lot of protocol overhead [45].

One possible conclusion that is often drawn from this fact is that it does not pay to use unidirectional links in a routing protocol. An alternative one is that the overhead produced by the protocols needs to be reduced. This thesis follows the second line of reasoning, and introduces, describes and evaluates five new routing protocols for wireless sensor networks with unidirectional links.

Wireless Sensor Networks, Unidirectional Links, Routing Protocols

Zusammenfassung

Experimente mit drahtlosen Sensornetzen haben gezeigt, dass unidirektionale Verbindungen häufig auftreten und oft eine grössere Distanz überbrücken als bidirektionale. Trotzdem werden sie in den meisten Routingprotokollen nicht genutzt. Diese Protokolle beschränken sich darauf, negative Auswirkungen der unidirektionalen Verbindungen zu eliminieren. In Protokollen, die unidirektionale Verbindungen verwenden, entsteht hierdurch meist ein hoher, zusätzlicher Aufwand [45].

Viele Protokollentwickler schliessen daraus, dass es sich nicht lohnt, unidirektionale Verbindungen zu nutzen. Eine alternative Schlussfolgerung wäre es zu sagen, dass der Aufwand, der durch die Verwendung der unidirektionalen Verbindungen entsteht, reduziert werden muss. In dieser Doktorarbeit wurde der zweite Weg gewählt: Es werden fünf neue Routingprotokolle für drahtlose Sensornetze vorgestellt und evaluiert, welche unidirektionale Verbindungen nutzen.

Drahtlose Sensornetze, Unidirektionale Verbindungen, Routing Protokolle

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> "*Unidirectional links commonly occur in wireless ad hoc networks because of the differences in node transceiver capabilities or perceived interference levels. Unidirectional links can presumably benefit routing by providing improved network connectivity and shorter paths. But prior work indicates that routing over unidirectional links usually causes high overheads.*"[45]

1.1 Motivation

Most of the wireless data communication today uses one hop only, i.e. direct communication between devices. Common examples include Bluetooth keyboards that transmit directly to a computer or laptops that communicate with a base station using WLAN. Another example can be found in factory automation. When moving parts have to be monitored, connecting them with cables is often impossible: The movement would shear off the cables. An example for such moving machinery can be found in [34], which describes a sensor network used for vibration analysis in a semiconductor plant. Also, the use of wireless communication can reduce installation costs. When an existing factory should be augmented with additional monitoring equipment, it might be necessary to pass fire doors or other safety installments. Keeping these operational would require opening a wall to install the cables and sealing it again afterwards, which is very costly. For all these reasons, more and more wireless communication devices are installed every day.

In ever expanding networks, the communication patterns also change. When it is no longer possible for each device to communicate directly, multihop communication is necessary, where devices that are close to the destination forward the messages from those that are too far away to communicate directly.

One of the fields in which multihop communication is very important are wireless sensor networks. These networks consist of lots of cheap sensor nodes which are equipped with a number of sensors (e.g. temperature, light) which are important to the application, a small CPU, a few kB of RAM, a small flash memory and a radio transceiver. Often, these nodes are powered by a battery pack, but other types of power supply also exist. As wireless sensor networks should contain lots of nodes, the price for individual nodes must be kept low, resulting in the need for cheap components. Cheap radio modules which are not calibrated often have very different radio characteristics, which, in combination with environmental influences, lead to often changing links between nodes (communication neighborhood). Nodes that can communicate now may not be able to do so in a minute, but again in five minutes from now. Also, differences in the used transceivers, in node placement and battery status often lead to unidirectional links. A unidirectional link between two nodes A and B exists if node A can communicate with node B, but not vice versa.

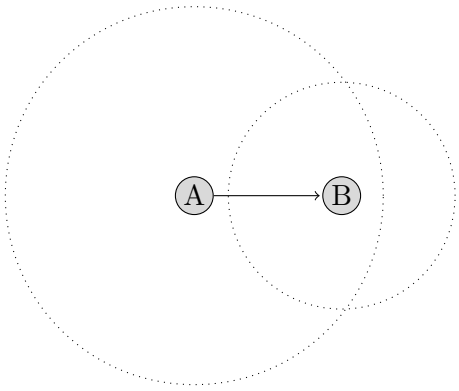


Figure 1.1: An Example of a Unidirectional Link

Figure 1.1 shows an example of such a unidirectional link from node A to node B. Node A might be placed on higher ground or might be using a fresh set of batteries. Whatever the reason, its transmission range is greater than that of node B, resulting in said unidirectional link. More about the frequent occurrence of unidirectional links, which has been confirmed in many real world experiments and described in literature, is presented in chapter 2.

While the route from source to destination is known implicitly in most cable networks, the ever changing links result in the need for adaptive protocols in wireless networks. But most of the protocols presented in literature assume bidirectional links, i.e. when node A can receive messages from node B, node B can also receive messages from node

A. Often, these protocols include some means to detect unidirectional links, in order to remove their implications. A prominent example is blacklisting of nodes, which is used in many routing protocols, e.g. AODV [57] (see section 3.2.2). For more information about the way MAC and routing protocols react to the presence of unidirectional links see chapter 3.

Making unidirectional links usable for communication protocols introduces overhead, as the upstream node of a unidirectional link (node A in figure 1.1) needs to be informed of the existence of that link somehow. Marina and Das have evaluated some routing protocols that make usage of unidirectional links [45]. Their results show that the gain in connectivity is small, and the overhead is too big when per-hop acknowledgments are used. Therefore they argue that the costs are not worth the gain and routing protocols should only utilize bidirectional links.

However, literature suggests that unidirectional links are common, and the connectivity evaluations made for this thesis (section 5.3) show that they occur much more often than bidirectional links. In some cases, using only bidirectional links leads to network separation which can be avoided when unidirectional ones are included.

Therefore, this thesis follows a different line of reasoning from the one presented in [45]: If the costs of per-hop acknowledgments are too high, they should be omitted. Also, all of the evaluated protocols made unidirectional links usable explicitly, by informing the upstream node of their existence. If the methods used for this are too expensive, they must be replaced by ones that are less expensive, or the unidirectional links must be used only implicitly. In this context, using unidirectional links implicitly means that the upstream nodes are not informed of the link, but the messages forwarded along these links are processed and maybe forwarded nonetheless.

Following this reasoning, five new protocols have been designed in this thesis, which are intended for use in networks with mostly unstable links, many of which are supposed to be unidirectional. The source routing protocol Buckshot Routing (section 4.1), its distance vector based equivalent BuckshotDV (section 4.2) and the overhearing supported enhancement OSBRDV (section 4.3) focus on using unidirectional links implicitly, without informing upstream nodes of their existence. The other two protocols, Unidirectional Link Triangle Routing (ULTR, section 4.4) and Unidirectional Link Counter (ULC, section 4.5) were designed for sensor networks in which each node knows its two hop neighborhood anyway. This can be the case either because the application uses a neighborhood discovery protocol, or because a TDMA MAC is used. In both cases, the gathered information can be supplied to the routing protocol without introducing additional costs.

1.2 Structure of this thesis

In this work new routing protocols for wireless networks with unidirectional links are presented. Even though these protocols focus on using unidirectional links which are often experienced in wireless sensor networks (WSN), they can be just as easily transferred to mobile ad-hoc networks (MANETs) where these links also occur. The reason for the focus on unidirectional links is explained in chapter 2. The chapter describes experiments that have shown the frequent occurrence of unidirectional links before explaining their impact on the medium access control (MAC) - and routing layers. Even though this work is focused on the development of new routing protocols, it is nonetheless essential to think about the implications for the MAC protocols first. Once those are clear, the impact on the routing layer and some cross-layer issues are discussed.

Chapter 3 shows related work, divided into two sections. Section 3.1 discusses exemplarily chosen MAC protocols that are able to use unidirectional links (section 3.1), because a routing protocol that uses unidirectional links needs a MAC protocol that can use them, too. Otherwise, the routing protocol might choose a node as next hop that is not known by the MAC protocol, causing the MAC protocol to discard the message. The second, much larger section of this chapter is dedicated to the description of existing routing protocols, both from MANETs and WSN (section 3.2). Each protocol is classified and its ability to work in the presence of unidirectional links is described. If it can use them, the costs for the usage are also determined.

The five newly developed protocols Buckshot Routing, BuckshotDV, OSBRDV, ULTR and ULC are described in chapter 4. Also, the basic advantages and disadvantages of each of them are described in theory there.

The evaluation can be found in chapter 5. The first section shows the simulation model (section 5.1), before the real world experiment environment is described in section 5.2 and connectivity measurements are presented in section 5.3. The sections 5.4 to 5.6 describe the results of different simulations and real world experiments according to their application scenario before a closer look is taken at the interaction between routing protocols and duplicate suppression in section 5.7. The chapter is concluded with a summary of the obtained results and a guide for choosing the right protocol under given network conditions (section 5.8).

A list of certain features that could enhance the presented protocols in the future can be found in chapter 6, while concluding remarks are given in chapter 7.

Chapter 2

Unidirectional Links

Different classifications of link quality are used in literature. The most commonly used classification divides links into bidirectional links, asymmetric links and unidirectional links. A bidirectional link is always defined as a link between two nodes which can be used to transmit a message from either of those two nodes to the other one. The terms asymmetric link and unidirectional link are not always defined as clearly, and sometimes used synonymously. Common definitions for asymmetric links focus on a variation of either RSSI values (Received Signal Strength Indication) or packet loss (delivery ratio). When the delivery ratio is used, unidirectional links can be seen as a subclass of asymmetric links where the delivery ratio in one direction is 0. But this definition requires the transmission of multiple messages in order to evaluate the delivery ratio.

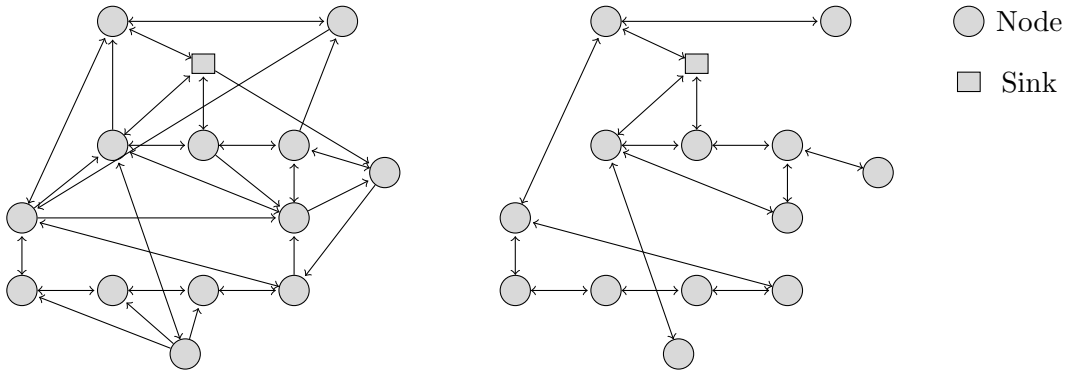
For this thesis an unidirectional link is defined as follows: A link from node A to node B is unidirectional, if node B can receive messages from A, but not vise versa. Even though this definition is quite straightforward, it is only a theoretical one. In practice, it is fairly hard to establish such criteria. It is not possible to monitor the status of all links globally. Even the status of a single link can only be measured at a certain time. Moreover, only one direction of the link can be measured because the transceivers used on typical sensor nodes can only transmit or receive and use one channel at a given time. Worse still, links change over time. A link that seems to be bidirectional at one moment can become unidirectional or even vanish completely at the next moment.

2.1 Unidirectional Links in Real-World Experiments

This section summarizes some of the experiments with wireless sensor networks which have shown that unidirectional links do not only exist, but are in fact fairly common.

2.1.1 The Heathland Experiment: Results And Experiences

Turau et al. describe an experiment they conducted in the Lüneburger Heide in Germany [71]. The original goal was to evaluate a routing protocol, which is not characterized further in the paper. Rather, the observations they made concerning the properties of the wireless medium are described, focusing on the frequency of changes and the poor stability of links. This experiment was conducted using up to 24 Scatterweb ESB [67] sensor nodes, which were affixed to trees, poles etc., and left alone for two weeks after program start. One of the purposes of the network was the documentation of the logical topology (radio neighborhood of nodes), which was evaluated by building a new routing tree every hour, e.g. for use in a sense-and-send application. The neighborhood was evaluated using the Wireless Neighborhood Exploration protocol (WNX) [71], which can detect unidirectional and bidirectional links. All unidirectional links were discarded and only the bidirectional ones were used to build the routing tree.



(a) All Links (b) Only Bidirectional Links

Figure 2.1: A Communication Graph (a)with and (b)without Unidirectional Links (taken from [71] presentation: [72])

Figure 2.1(a) shows one complete communication graph obtained by WNX, while figure 2.1(b) shows the same graph without unidirectional links, where a lot of redundant paths have been lost by the elimination. In fact, one quarter of the nodes are only connected to the rest of the network by a single link when unidirectional links are

removed. If this single link breaks, the nodes become separated, even though there are still routes to and from them. Thus, the removal of unidirectional links increases the probability of network separation. As described above, WNX was used to evaluate links between individual nodes. These links were divided into bidirectional and unidirectional ones, and all unidirectional discarded. The sink then started building a depth first spanning tree, which was chosen intentionally in contrast to the breadth first trees used normally. The authors state that, to evaluate their routing protocol, they needed to have a certain number of hops, which a breadth first search would not have delivered.

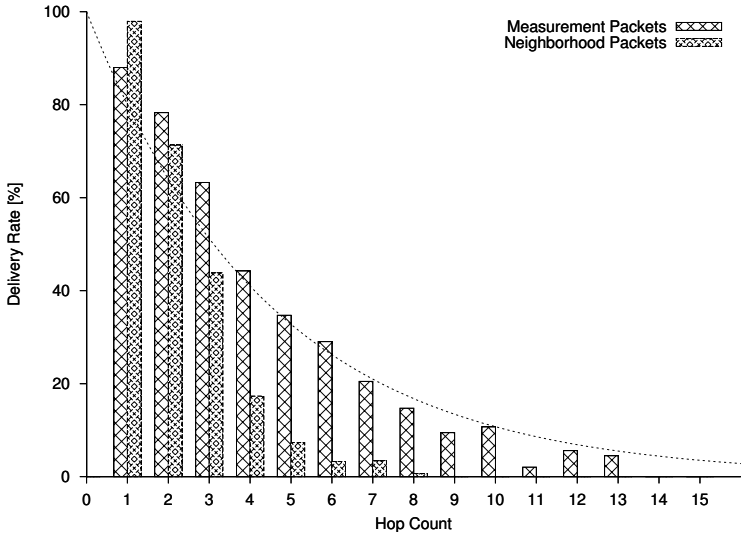


Figure 2.2: Packet Delivery versus Hop Count (taken from [71])

Figure 2.2 shows the delivery ratio observed for measurement packets (small) and neighborhood information packets (large), depending on the distance (hop count). It can be seen that the delivery ratio seems to shrink logarithmically with the distance: The function $f = 100 * 0.8^d$ (plotted curve) closely approximates the values obtained for the measurement packets.

As expected from theory, the experiments confirm that shortest path routing is indeed a good choice for lossy networks such as wireless sensor networks, because the overall loss probability is smaller for shorter paths. Therefore, the protocols developed in this thesis (presented in section 4) will also try to use the shortest path whenever possible.

2.1.2 On exploiting asymmetric wireless links via one-way estimation

Sang et al. propagate a similar opinion [65]. They evaluate the three kinds of links (asymmetric, unidirectional, bidirectional) using protocols like ETX (Expected Transmission Count) [12]. These protocols search for reliable connections, but most focus on bidirectional links only. This leads to the fact that a link with a reliability of 50% in both directions is preferred to one with 100% from node A to node B and 0% from B to A. If data needs to be transmitted only from A to B without need for acknowledgment, this choice is obviously wrong. To prevent this wrong choice, the authors of [65] propose a protocol called ETF (Expected Number of Transmissions over Forward Links), which is able to use unidirectional links. They also show that the reach of reliable unidirectional links is greater than that of reliable bidirectional links.

In experiments with XSM motes [65] the nodes were placed in a seven by seven square, with a distance of about one meter between nodes. In four sets of experiments at different times of day each node sent 100 messages at three different power levels. Then the packet reception rate was recorded. It is defined for a node A as the number of packets A received from a node B divided by the number of messages sent (100). Following this, the packet reception rates of nodes A and B are compared. If the difference is less than 10%, the link is considered bidirectional. If it is more than 90% the link is considered unidirectional. All other links are called asymmetric. The XSM nodes offer 9 different transmission power levels, of which three were evaluated: the lowest, the highest and the third in between. Table 2.1 shows the results of the experiments.

Table 2.1: Link Quality versus Transmission Power (taken from [65])

	bidirectional	asymmetric	unidirectional	number of links
power level 1	50%	43%	7%	500
power level 3	65%	22%	13%	1038
power level 9	88%	6%	6%	1135

The results show that even when using the maximum transmission strength, 12% of the links would have been discarded by ETX and similar link quality evaluation protocols that focus only on bidirectional links. As the lifetime is one of the major optimization goals in a sensor network and receiving/transmitting consumes a lot of energy, it is rather uncommon to have all nodes constantly transmit using the highest transmission strength. In fact, current research projects like [47] try to minimize power

consumption by adjusting the transmission strength depending on the required reach and reliability.

The observations of [65] are summarized by the authors in three points:

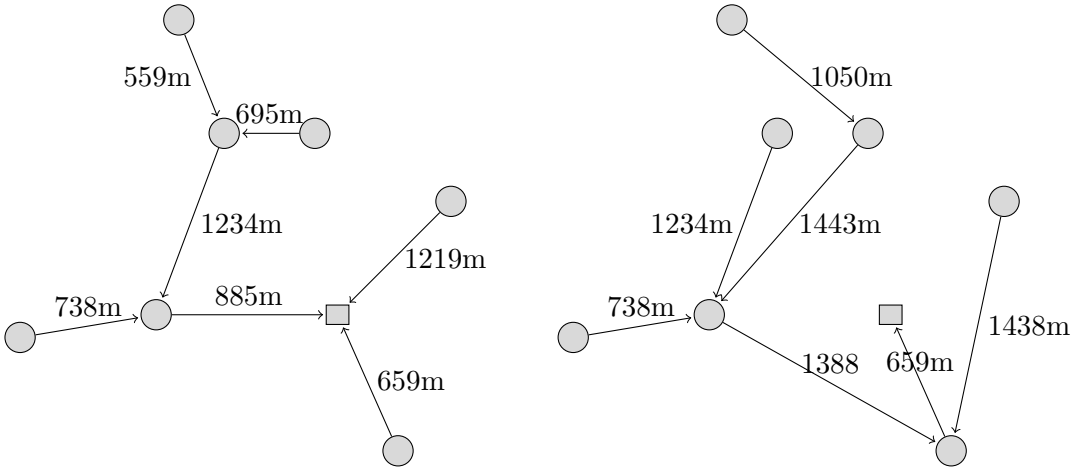
1. Wireless links are often asymmetric, especially if transmission power is low.
2. Dense networks produce more asymmetric links then sparse ones.
3. Symmetric links only bridge short distances, while asymmetric and especially unidirectional ones have a much longer reach.

A conclusion drawn from these three facts is that the usage of unidirectional links in a routing protocol can increase the efficiency of a routing protocol considering energy and/or latency.

2.1.3 Design and Deployment of a Remote Robust Sensor Network: Experiences from an Outdoor Water Quality Monitoring Network

A sensor network which monitors water pumps within wells is described in [13]. The sensors were used to monitor the water level, the amount of water taken and the saltiness of the water in a number of wells which were widely distributed. The necessity for this sensor network arose because the pumps were close to shore and a rise in saltiness was endangering the quality of the water. The average distance between wells was 850 meters and the transmission range was about 1500 meters. Communication was realized using 802.11 WLAN hardware both for the nodes as well as for the gateway. For data transmission between nodes Surge_Reliable [81] was used, which makes routing decisions based on the link quality between nodes.

During the experiments it could be seen, that the (logical) topology of the network changed dynamically, even though all nodes were stationary. The authors claim that these changes were probably due to antenna size and changes in temperature and air moisture. In this context it is important to remember that the distance of nodes was far below the range of the transmitters (about 50%). While about 70% of the routing trees observed followed the theory (figure 2.3(a)), there were a lot of strange ones. In one case the average distance between connected nodes even rose to 1135 meters, as nodes that should have been able to communicate directly with the gateway were connected to nodes on the far side instead. In one of these routing trees (figure 2.3(b)), a single node had to take care of all communication with the gateway, even nodes that were on the other side were using it as next hop. One possible reason for this strange behavior



(a) A Normal Connectivity Graph

(b) A Degenerated Connectivity Graph

Figure 2.3: Two Communication Graphs (taken from [13])

is that Surge_Reliable chooses the nodes with the best link quality, but only considers bidirectional links. If unidirectional links would have been used, the results could have been quite different.

2.1.4 VigilNet: An integrated sensor network system for energy-efficient surveillance

VigilNet, a military sensor network for terrain surveillance, is described in [21]. This project aims at the detection of moving vehicles using magnetic sensors attached to Mica2 sensor nodes. The transport of messages from the nodes to the sink was realized using a diffusion based algorithm, similar to Directed Diffusion [27], which produced a routing tree with its root at the sink. To eliminate unidirectional links, a protocol called Link Symmetry Detection was developed. Each node periodically transmitted the list of its neighbors. A node that received such a neighbor list checked the list to determine if it was mentioned. If it was not, the link was an incoming unidirectional one. When building the routing tree after deployment, the transmission power of all nodes was halved. Now all nodes determined their parent node from the neighbor lists received with this half strength. At the end of this setup phase, all nodes switched to full transmission power. The intention behind this scheme was to ensure that the connection to the father node would not break. During the experiments, the authors noted that asymmetric links were far more common than expected. They put this fact down to

differences in hardware, as the transceivers were not calibrated before the experiment. Another interesting effect seen in these experiments is that only about 2/3 of all nodes were able to communicate directly with the sink, because only bidirectional links were used. If the usage of unidirectional links was enabled, a lot of multi hop communication might have been saved.

2.1.5 Taming the underlying challenges of reliable multihop routing in sensor networks

The main focus of [81] is link quality estimation. The authors measured link quality for a sensor network deployment consisting of 50 Mica Motes from Berkeley.

Figure 2.4 shows the results they obtained. All nodes within a distance of about 10 feet (about 3 meters) or less from the sender received more than 90% of the transmitted packets. The region within 10 feet of the sending node is therefore called the *effective region*. It is followed by the *transitional region*. Nodes in this region cannot be uniformly characterized as some of them have a high reception rate while others received no packets at all. In the *transitional region*, asymmetric links are common. The last region is the *clear region* and contains only nodes that did not receive any transmissions.

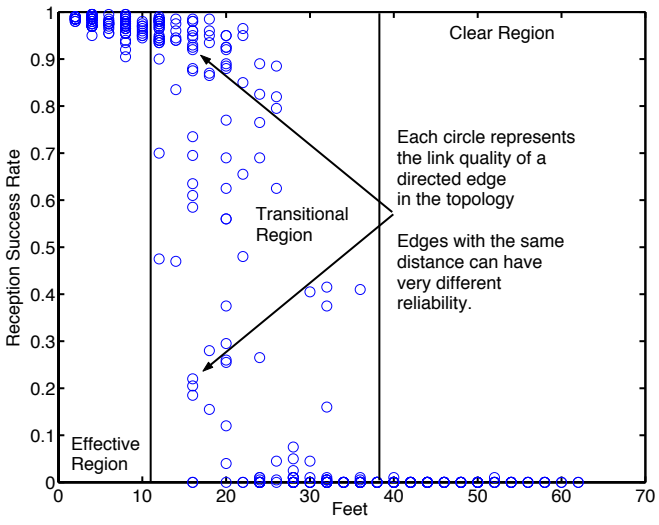


Figure 2.4: Effective, Transitional and Clear Region (taken from [81])

The authors argue that nodes in the *effective region* should be preferred when neighbor table decisions are made. A neighbor table decision must be made when a message is received from a node that is not in the table and the table is full. Then, the choice is

to remove one of the entries to make room for the new one, or to discard the new one silently.

When a new entry is made, the question remains which other entry should be removed. This is a special case of cache management problems. The authors decided to use the *frequency* algorithm. Each neighbor entry holds a frequency value which is incremented every time a message is received from the corresponding node. When a message from a node not in the table is received, the table is checked for entries with frequency value 0. If one is found, it is removed and the new neighbor entered. Otherwise, no new entry is made, the information about the possible neighbor is discarded and all entries in the neighbor table are reduced by one.

Link estimation is realized passively by snooping. Each node overhears all transmissions from its neighbors and notes the sequence numbers contained therein. If it overhears the same sequence number twice or more, it knows that retransmissions are underway, meaning the link is lossy. To enable the correct working of this snooping mechanism, the authors assume a minimum transmission rate for each node. Still, snooping is only possible for bidirectional links and only for direct neighbors.

One of the major problems of this approach is information asymmetry. Each node has estimations for in-bound link quality, but the routing decisions must be made based on out-bound links. Therefore, link quality estimates are shared with the neighbors on a periodic basis.

For this thesis, the estimated link quality does not play a major role. On the contrary, the protocols presented herein are meant to abstract from such problems in order to keep them simple yet effective. Another example for this is that the authors of [81] argue that minimum hop count routing often leads to choosing nodes from the *transitional region* as next hop, which in turn leads to more frequent route breaks. To take care of this fact, one of the protocols which were developed in this thesis (Buckshot Routing, see section 4.1) uses a kind of multi-path routing, which renders it immune to single broken links..

2.1.6 Understanding packet delivery performance in dense wireless sensor networks

Zhao and Govindan measured the properties of wireless sensor networks on the physical and medium access control layers [85]. These measurements were conducted using up to 60 Mica motes, which were placed in three different environments: An office building, a parking lot and a habitat. The experiments for the physical layer were realized with a single sender and multiple receiver nodes, and have shown the existence of a *grey area*

in reception which can consist of up to one third of the network. In this *grey area*, the reception quality of nodes varies a lot, both spatial as well as in time. This observation is similar to the *transitional region* described in [81](see section 2.1.5). Another result described by the authors is that in the parking lot and indoor environments nearly 10% of links are asymmetric. Please note that what the authors call asymmetric links is otherwise referred to as unidirectional links in this thesis: "*Asymmetry occurs when a node can transmit to another node but not vice versa*" [85]

The authors suggest that neighbors should be selected based on the measured packet delivery performance, when routing decisions are made. This poses two problems:

- The measurement can only be made by the receiver and must be communicated to the sender, and
- this measurement induces a lot of overhead, as a significant number of messages has to be transmitted in order to get a reasonably good value for packet delivery performance.

The MAC layer evaluation used a simple CSMA/CA protocol, which is the default implementation for TinyOS. It was augmented with a retransmission scheme, to make use of the link-layer acknowledgments that were being transmitted anyway. An interesting observation the authors made is that between 50% and 80% of the communication energy is used for reliability arrangements: retransmissions, forward error correction, encoding and similar.

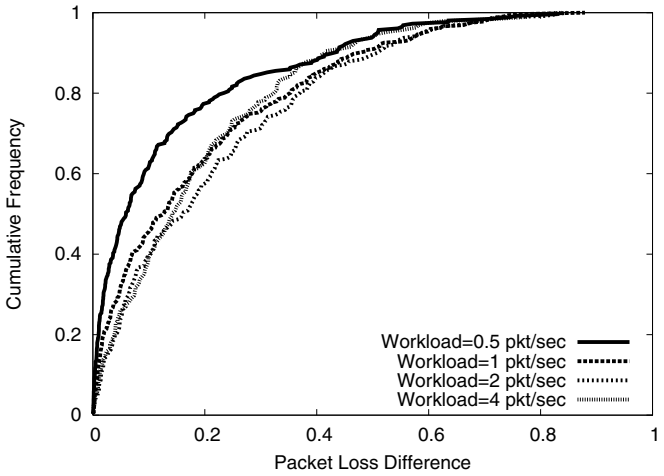


Figure 2.5: Packet Loss Difference for Pairs of Nodes (taken from [85])

Figure 2.5 shows the finding of the authors that is most important for this thesis. They have defined the packet loss difference for two nodes as the difference between the packet delivery efficiency of both nodes. The figure shows that asymmetric (unidirectional) links are quite common: More than 10% of the surveyed links have a difference of more than 50%.

The final claim the authors make is about asymmetric (i.e. unidirectional) links: *"The fraction of asymmetric links is high enough that topology control mechanisms should, we argue, carefully target such links"*. All protocols presented in this thesis (section 4) are based on the existence of unidirectional links, and aim at using them in order to increase routing performance.

2.1.7 Lessons Learned from Implementing Ad-hoc Multicast Routing in Sensor Networks

The implications of implementing a MANET protocol for wireless sensor networks are described in [62, 63]. The authors state that most routing protocols that have been developed for wireless sensor networks are based on a tree topology, due to the assumption that data has to be transported to a single sink. They argue that a number of applications for sensor networks exist that do not follow this model and require multicast routing. Tracking of firefighters, mobile nodes and disaster recovery scenarios are mentioned as examples. The authors also state that many multicast routing protocols for MANETS have been simulated under unrealistic conditions.

The multicast routing protocol chosen for implementation is Adaptive Demand-Driven Multicast Routing (ADMR). It uses forwarding trees to transmit messages from multiple sources to multiple sinks. One of the first drawbacks of this protocol as described in [62] is that it uses inverse routes for so-called *Receiver-Join* messages, which are transmitted by sinks that want to be part of a multicast group. Another problem the authors identified is the usage of a minimum hop count weight function. They argue that this weight function leads to a high loss rate due to the usage of potentially unstable links. Instead, a weight function based on link stability should be used.

To validate this claim, the authors evaluated link quality of pairs of nodes (forward link and backward link). Each node transmitted a certain number of packets, and all nodes recorded how many packets they received. The link delivery ratio is calculated by dividing the number of messages received from a node by the number of messages transmitted by this node. Figure 2.6 shows the obtained results, sorted by the link delivery ratio of the forward link. It can be seen in the figure that a lot of discrepancies exist.

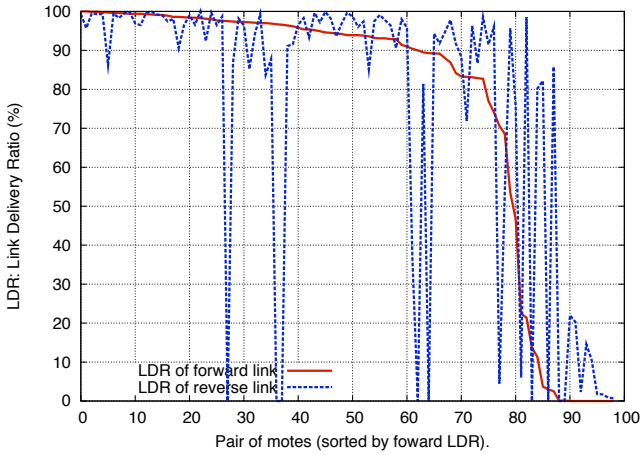


Figure 2.6: Differences in Link Delivery Ratio (taken from [62])

To cope with such discrepancies, ETX [12] has been developed. The quality indicator presented by ETX takes forward and backward link quality into account. The authors of [62] propose a different approach. They show that the path delivery ratio would be the optimal weight function for routing protocols, but it is hard to obtain and requires additional communication. Also, the link quality indicator that is already available in the radio module (chipcon CC2420 [23]) on the used motes delivers values that are fairly close to the measured link quality.

Another problem identified by the authors is that routing protocols for MANETS are often designed with large storage space in mind, and protocol data tables (e.g. routing tables) may not fit completely into the RAM of a sensor node. Replacement strategies are needed and evaluated, but none of them leads to a desirable result. Instead, the authors claim that a way needs to be found to make the flash memory usable for such purposes.

While the observation that path reversals may lead to bad packet receptions rates is valid, one of the conclusions drawn from this fact, namely that lowest hop count should not be used, is not shared by the author of this thesis. The asymmetric and unidirectional links can be made usable, as long as their implications are already considered during protocol development. See chapter 4 for details.

2.1.8 Multichannel reliability assessment in real world WSNs

Ortiz and Culler studied the feasibility of using multiple channels in wireless sensor networks [52]. They evaluated link quality in three different testbeds: A machine room, a computer room and an office building, using up to 60 sensor nodes. During the experiments, each node transmitted 100 messages and each other node recorded the number of received messages, enabling easy calculation of the packet reception rate.

The authors found that asymmetric links were indeed common in their testbeds. Furthermore, they defined a link between two nodes to be unidirectional if the packet reception rate was above a threshold T in one direction, and less than T in the other. T was varied between 1% and 90%. In the machine room this lead to 32 - 36% of links being unidirectional, 18 - 34% in the computer room and 10 - 46% in the office building.

2.1.9 Murphy loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture

Langendoen et al. describe the deployment of a sensor network used to monitor temperature and humidity on potato plants [36]. These influence factors are monitored to prevent fungal infections of the plants. Normally, all plants are treated with anti-fungus chemicals, in the experiment only those which exhibited optimal conditions for fungal growth were to be treated in order to reduce pollution and costs.

The program on the sensor nodes consisted of medium access control (T-MAC), routing (MintRoute) and over the air reprogramming (Deluge), and of course the sampling application. The authors describe lots of problems they encountered during the deployment. For this work, however, only those induced by the used communication protocols are relevant.

All three protocols are working fine when used separately, but did not work in combination. T-MAC has been designed with a low network load in mind, and had massive problems when Deluge started to flood update messages. MintRoute produced long routes even for nodes that could theoretically communicate directly with the gateway. The authors blame this fact on the often occurring replacements in the routing table: They were using 109 nodes of which 70 were in direct neighborhood of the gateway. But the routing table of MintRoute only held 16 entries. The replacement strategy was least heard, meaning that those nodes from which nothing had been heard for the longest time were removed. The gateway never transmitted any messages of its own accord, and thus was removed from the routing tables very often.

T-MAC used its own neighbor list which featured 20 entries, but used a first-in first-out strategy which lead to different entries than the ones stored by MintRoute.

When MintRoute had chosen a neighbor as next hop, T-MAC often did not know that neighbor and discarded the packet. Together with some other problems which are not mentioned here (see [36] for details) these problems resulted in only 2% of the expected data reaching the gateway.

The cross-layer cooperation between MAC and routing layer, established for example by using a common neighbor table as discussed at various points throughout this thesis (for example in section 4.6) addresses the inter-protocol problems described above.

2.2 Impact on the MAC Layer

MAC protocols can be roughly divided into two categories: contention based protocols and dividing protocols.

Contention based protocols work on the basic principle that a node that wants to transmit a message needs to compete for the medium with its neighbors. The easiest way to do this is to listen to the medium and transmit if it is free. If it is occupied, listening is repeated after a certain or random amount of time.

Dividing protocols divide the access to the medium according to certain properties. These include frequency (FDMA), encoding (CDMA) and time (TDMA) [66, 70]. Each of those focuses on one property that has to be different in order to allow concurrent transmissions with the same value for the other properties. If two nodes transmit on a different frequency, they can use the same encoding and transmit at the same time (FDMA). With different encoding, these two nodes can send on the same frequency at the same time (CDMA). If they transmit at different points in time, they can use the same encoding and frequency (TDMA). There is a fourth dimension to this, which is often ignored: SDMA. It focuses on the spatial differences. However, moving a node in order to enable it to transmit is not a commonly used ability because of the overhead. It is normally used in all other protocols implicitly, allowing nodes that are at least two hops distant to transmit in the same slot, use the same frequency or encoding. Of course, it is also possible to combine the different approaches. For example, TDMA and FDMA are combined in GSM [19, 66, 70].

FDMA protocols require complex and often expensive hardware. Even if each node is assigned its own transmission frequency, the receiving nodes would need to be able to listen on multiple frequencies or know beforehand, when to switch to which frequency. Therefore, FDMA is not often used in wireless sensor networks. The usage of CDMA induces a lot of computation overhead and different codes are needed for all nodes within two hop communication range. The ever changing nature of this two hop neighborhood and the battery powered nature therefore prohibit the usage of CDMA in most sensor networks. Therefore, only contention based and TDMA MAC protocols and the impact of unidirectional links on them are considered in this thesis, but the results can easily be transferred to any of the other protocols by replacing time with e.g. frequency.

2.2.1 Contention Based Protocols

A typical problem that has to be solved in wireless networks is the hidden station problem. In the hidden station problem, node A wants to transmit data to a node B. Before starting the transmission, it listens to the medium, to determine whether the channel is occupied (CSMA). If it is not, node A begins its transmission. The problem is that collisions occur at the receiver, not at the sender. A third node C might well be in transmission range of B, but not of A. Therefore, when listening to the medium, it would assume that the channel is free and begin its transmission, even though A is transmitting. Consequently, the messages from A and C collide at node B.

To solve this problem, virtual channel sensing has been invented. After the real channel sensing, when node A assumes that the medium is free, it sends a Request To Send (RTS) message which contains the ID of the intended recipient and the length of the proposed data transmission i.e. the amount of time in which the medium will be occupied. When node B receives the RTS message and the medium is free, it transmits a Clear to Send (CTS) message, which once again contains the length of the transmission. This way, all nodes that could disrupt the transmission from A to B have received either the RTS or the CTS and know that the medium is occupied, even though it might seem to be free.

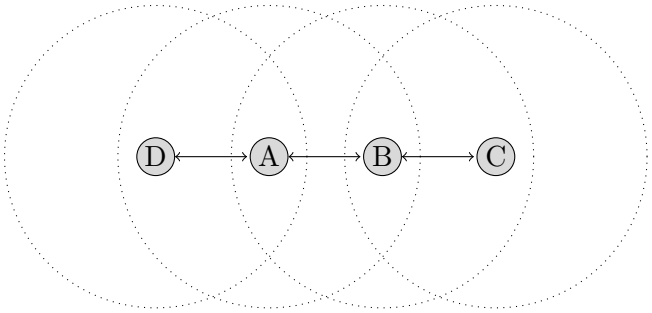


Figure 2.7: Virtual Channel Sensing

Figure 2.7 shows an example. In step 1 node A transmits the RTS. Nodes B and D receive it and know about A's wish to transmit data to B. In step 2 node B transmits the CTS, informing C about the impending communication, while at the same time informing A that it might start transmitting. In step 3 the data message is transferred from A to B without collision, as all nodes know that they have to wait until the end of that transmission.

For the given scenario virtual channel sensing does indeed solve the hidden station problem. However, one of the basic assumptions of this protocol is that all links are bidirectional. In networks with unidirectional links a number of problems arise.

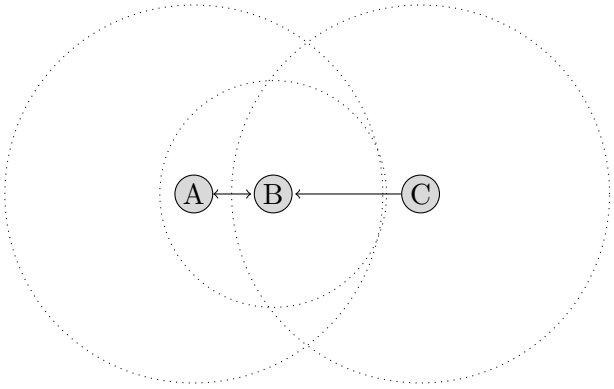


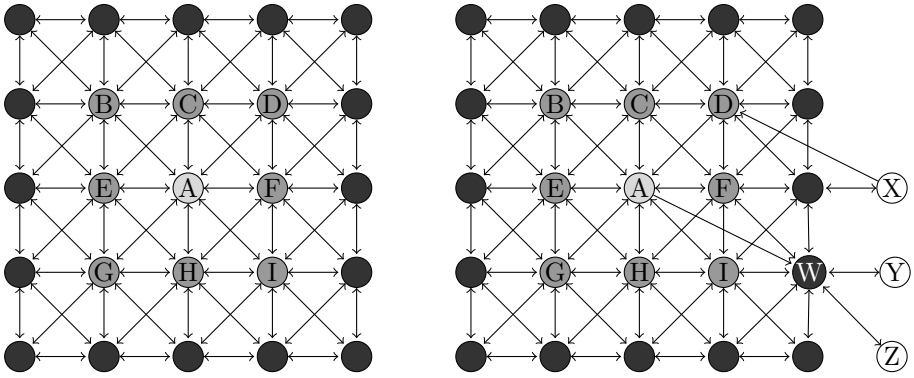
Figure 2.8: Virtual Channel Sensing with Unidirectional Links

In the example seen in figure 2.8 the communication range of nodes A and C is much larger than the one of node B. There are multiple possible reasons for this, e.g. stronger batteries or a higher vantage point. Whatever the reason, there is an unidirectional link from C to B. When node B transmits its CTS message in response to the RTS from A, C does not receive it. A on the other hand does, and assumes that the medium has been reserved for its data packet and starts transmitting. If C transmits to any of its neighbors now, it destroys the message from A at B. Therefore, the hidden station problem cannot be solved by a traditional RTS-CTS mechanism when unidirectional links occur. There are some approaches that tackle this problem, e.g. by forwarding the CTS message over multiple hops as BW_RES [59] (section 3.1.3) does. Different approaches are discussed in section 3.1.

2.2.2 TDMA Protocols

TDMA MAC protocols divide time into slots and frames, where only one node may transmit[1] in a given slot. This slot re-occurs every frame, and must be unique within the collision domain. When all nodes have been assigned their correct slots, no collisions will occur. Also, the hidden station problem does not exist, because the slots are defined just in such a way, that no nodes within two hops of each other may transmit at the same time.

If a node A has been assigned a slot `x`, none of its neighbors may have the same slot. Additionally, as node A will transmit to one of its neighbors, no other node that may transmit to any of them may have the same slot either. All these nodes are members of A's collision domain.



(a) Bidirectional Links Only (b) Unidirectional Links Added

Figure 2.9: Two-Hop Neighborhood

In networks where all transmission ranges are equal and only bidirectional links exist this collision domain can be easily identified. It consists of all nodes in the two-hop-neighborhood of the sender, meaning all nodes that can be reached by flooding a message with a time to live of 2. In Figure 2.9(a) the collision domain of the light grey node A consists of 24 nodes; its 8 direct neighbors, nodes B - I (dark grey) and their 16 neighbors (black). As long as the neighborhood remains stable, no collisions can occur. If the neighborhood can change, adaptive protocols are needed. In this case contention based protocols are often preferred, because they do not introduce additional overhead, but there are also a number of adaptive TDMA protocols.

[1]There are also receiver based TDMA protocols where the slots are not assigned to a sender but rather to a receiver. The problems described here remain the same, though.

As shown at the beginning of this chapter, stable bidirectional links are not the normal case, at least not in wireless sensor networks. Instead, they are an uncommon feature. Even under good conditions, unidirectional links and frequent link changes dominate the appearance of the network. When unidirectional links are taken into account, the collision domain for TDMA protocols has to be redefined.

Figure 2.9(b) shows the same network again, this time including two unidirectional links (X→D and A→W). It is no longer the case that all nodes belonging to node A's collision domain can simply be reached by flooding a message over two hops. All neighbors of A are in the collision domain, where all neighbors now means all nodes which can be reached by A in one hop. Node A could disturb their communication with another node that uses the same slot, regardless whether they can transmit to A or not. In the example this means that the node W now also is counted a neighbor of A.

The second part of the collision domain are all nodes that can transmit to one of A's neighbors or to A directly. Note that here the unidirectional links play a prominent role. Node D has a bidirectional link to A and thus is one of its direct neighbors. D also has an incoming unidirectional link from node X, which is not reachable from node A with two hops. Still, it is in its collision domain, as messages from A and X would collide at D if both nodes were to choose the same slot.

The unidirectional link from A to W means that all nodes that can communicate with W, including Y and Z, are also in the collision domain of A. But A does not even know it has an outgoing unidirectional link to W, whereas W can sometimes deduct this knowledge. This information asymmetry has to be solved by informing A of the communication opportunity to W if unidirectional links should be used. If only their implications should be removed, it could be sufficient for some protocols that node W informs its neighbors that the slot used by A is already taken.

Figure 2.10 shows another example, the same one that was used for the contention based protocols. During network initialization, all nodes broadcast a hello message to explore their neighborhood. Node A receives messages from node B. Node B receives messages from A and C. C does not receive any messages. Now all nodes know which other nodes they can listen to, but none knows which ones they can communicate with. To solve this problem, status messages are used which include neighborhood information. Node A broadcasts that it can hear node B. B receives this message and knows that the link to A is bidirectional. Node B broadcasts that it can hear A and C. Node A receives this message and knows that the link to B is bidirectional. Node C broadcasts an empty list and does not receive anything. As node B receives the empty list from node C, it knows that the link from C is unidirectional incoming.

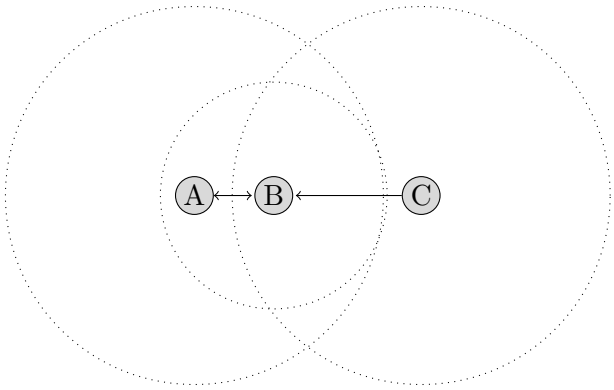


Figure 2.10: Collision Domain

When slots are assigned, nodes A and B know that they may not choose the same slot, and B knows that A and C may not choose the same one, but B has no way of informing C about that fact.

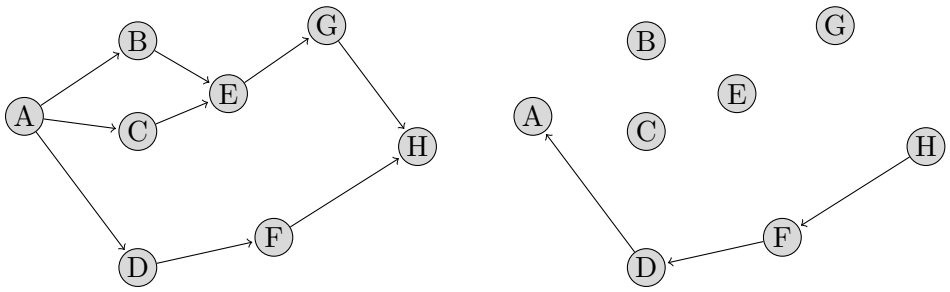
There are two ways of solving this problem, both involving node B. In a larger network, where B and C have a common neighbor or where there is indeed any route from B to C at all, B might inform C about the collision domain by sending a message over multiple hops. Please note that this is already becoming a routing problem, because the message has to be routed over multiple hops. Another way would be for B to listen to the transmissions from C to find out which slot it has chosen, and inform A that this slot is already taken. Even though this might sound easy, it is far from that. Node A must remain silent long enough for B to determine the slot chosen by C, or B must detect a collision in the slot. With only 3 nodes participating it might be easy to identify the reason for the collision, but in a real network with more nodes, outside influence and changing radio ranges it is far more complicated.

A similar approach is used in MLMAC-UL (section 3.1.9): Node B detects a collision in a slot (say slot 3). In its own slot, it transmits a status message containing the number of the slot (3) in which it detected a collision. All nodes that receive this message (node A in the example) and have chosen that slot release it and search for a new slot.

2.3 Impact on the Routing Layer

Most existing routing protocols are built for bidirectional links. A common way to detect a route from one node to another is to flood a message into the network. This is sometimes called *Route Request* Message (RREQ) in protocols like Dynamic Source Routing (DSR) [28] or Ad-Hoc On Demand Distance Vector Routing (AODV) [57]. In other protocols like Directed Diffusion[27] the flooded messages are called *Interests*. The basic mechanism is the same, though. Once the destination is reached, a message is sent back along the reversed path. In agent based protocols like Rumor Routing [7] the network is not flooded. Instead, an agent is sent which travels through the net using a random walk pattern. Still, the assumption that all links are bidirectional is the same. When an agent which was sent because of a certain event reaches a node, this node remembers the event and the next hop of the route leading to this event, which is the reversion of the route the agent has traveled.

Figure 2.11(a) shows the propagation of a RREQ message as used in AODV. The source, node A, wants to transmit to node H, the destination. As it does not know a path to node H, it floods the network with a RREQ message which reaches the destination through multiple paths. Once a RREQ message reaches the destination, the node addressed in the message sends a route reply message (RREP) along the reversed route. In the example the message from node F arrives first, and is answered with a RREP (figure 2.11(b)). All other RREQ messages that arrive later are ignored.



(a) Route Request Propagation (b) Route Reply Propagation

Figure 2.11: Route Discovery in AODV

In networks with stable bidirectional links these protocols provide good results. But when only one unidirectional link exists within the path the fastest RREQ takes, the RREP will be lost. Even worse, due to the fact that only the first RREQ is answered, all further RREQ messages that arrive at the destination are ignored, any bidirectional

paths that may exist are not found. When the source does not receive a RREP message after a certain time, it will restart the route discovery with the same result, only increasing network traffic but never finding a path.

This problem is exacerbated by the characteristics of unidirectional links. Most routing protocols assume that the fastest way is the optimal one, and therefore ignore all following messages. As observed above, the range of unidirectional links is often far greater than that of bidirectional ones, leading to fewer hops needed to reach the destination. Thus, the messages forwarded over unidirectional links will likely arrive earlier than those using the bidirectional ones.

A lot of routing protocols cope with this problem by eliminating the implications of unidirectional links. This elimination can be done e.g. by blacklisting as in AODV, or by requesting explicit acknowledgments, which is possible in DSR. There are also protocols which enable the usage of unidirectional links. Some do that by finding one way from source to destination and another one from destination to source as in one version of DSR, others by providing an abstraction between MAC and routing like, for example, the sub routing layer [61]. This abstraction can use multiple hops as return path from an unidirectional link, and presents the routing protocol with a network consisting only of bidirectional links. However, all these protocols introduce a significant communication overhead.

2.4 Cross-Layer Issues

The usage of unidirectional links in routing offers chances for cross-layer optimization. The information, whether a link between two nodes is uni- or bidirectional, cannot only be useful for the routing layer, but also for other protocols like medium access control, retransmission and transport. Link-layer protocols like Medium Access Control suffer heavily from unidirectional links. Most protocols assume that communication is symmetric i.e. that when node X hears node Y node Y can hear node X too. This is reflected in the protocols by the usage of flow control which is based on RTS/CTS signals or in the assumption that the medium is free when a node does not hear anything (CSMA). In the case of link a in figure 2.12 both approaches would fail because node Y cannot hear node X. Other protocols which use a timed schedule for sending (TDMA) suffer from asymmetric links, too. A local TDMA slot needs to be defined in order to avoid collisions. A node that is downstream of an unidirectional link may reserve an unnecessary slot for the upstream node while the upstream node does not realize that it can disturb the communication of the downstream node.

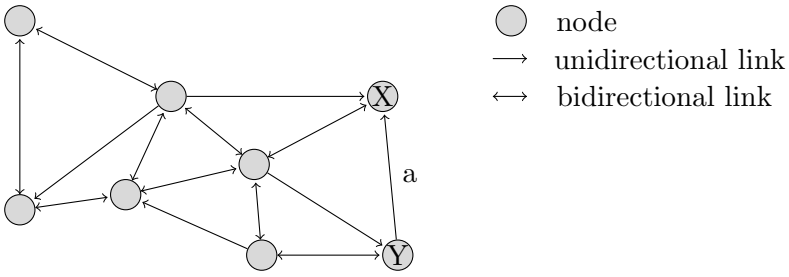


Figure 2.12: An example Network with Unidirectional and Bidirectional Links

The information, that the link between nodes X and Y is unidirectional should be made available to the MAC layer, when it is gathered by the routing protocol. On the other hand, if the MAC layer discovers a link breakage, it should inform the routing protocol that this link is no longer available. As it is not foreseeable which other layers may be interested in this kind of information, a cross-layer data structure should be used, which can be accessed by anyone interested.

In some protocols for wireless sensor networks, cross-layer issues are solved by combining the two layers of routing and medium access control. An example combination of tree routing and MAC layer is D-MAC [39, 40], which schedules the sleep cycles of nodes according to their height in the routing tree, thereby minimizing the end-to-end delay. While such approaches offer some performance improvements, they force the application designer to use the specified MAC-layer, when a certain routing protocol is used.

In this thesis a different approach is used, where cross-layer optimizations are still possible, but routing- and MAC-layer are only loosely coupled through the usage of a cross-layer data structure, which can be either used or ignored, depending on the MAC-layer implementation desired.

Chapter 3

MAC and Routing Protocols for Wireless Networks

Even though the declared goal of this thesis is the design of new routing protocols, it is still useful to take a look at a few medium access control protocols, albeit not in such depths as will be done for the routing protocols. The reason for this lies in the nature of unidirectional links. As described in chapter 2, their properties have a deep impact on multiple layers. Also, a routing protocol can be only as good as the underlying MAC protocol. If the MAC is not able to work with unidirectional links, e.g. because it needs direct link layer acknowledgments (i.e. RTS-CTS mechanisms), there can be no improvement whatever routing protocol is used. Another reason for the inclusion of MAC protocols in this thesis is the fact that in some protocols for wireless sensor networks the MAC and routing layers are combined into one.

Even though there are many more protocols in existence, only the most commonly used ones and those that use unidirectional links are discussed here. This chapter starts with a discussion of selected MAC protocols (section 3.1), before state of the art routing protocols are discussed in depth (section 3.2). The chapter ends with a list of own publications that were used in this chapter (section 3.3).

3.1 Medium Access Control

Using unidirectional links on the routing layer requires a MAC layer that does not suppress them. Therefore, a few selected MAC protocols are presented in this section, even though the focus of this thesis lies on routing protocols. Please note that only a few chosen protocols are described and the presentation is by no means exhaustive. Table 3.1 shows the protocols surveyed in this thesis. Protocols are described by their name, the type, and their way of dealing with unidirectional links. `Type` can either be contention based (CSMA) or time multiplexed (TDMA) or, in one case, a combination of both. The field `Unidirectional Links` takes on three values: Unidirectional Links not considered in the protocol (ignored), considered but not used (implications removed) and made usable (used). This section also includes a protocol that is not mentioned in the table, because it is not exactly a MAC protocol. The link layer tunneling mechanism [14] is not concerned with medium access control, but is included here because it deals with unidirectional links on layer 2 (see section 3.1.11).

Table 3.1: MAC Protocols Surveyed in this Thesis

Name	Source	Type	Unidirectional Links	Section
MMP	[18]	CSMA	ignored	3.1.1
NMAC	[42, 44]	CSMA	used	3.1.2
BW_RES	[59]	CSMA	implications removed	3.1.3
ECTS-MAC	[42, 44]	CSMA	used	3.1.4
AMAC	[80]	CSMA	used	3.1.5
PANAMA	[5]	CDMA & TDMA	implications removed	3.1.6
LMAC	[73]	TDMA	ignored	3.1.7
AI-LMAC	[64]	TDMA	ignored	3.1.7
MLMAC	[41, 43]	TDMA	implications removed	3.1.8
MLMAC-UL	[42, 44]	TDMA	used	3.1.9
D-MAC	[39, 40]	TDMA	ignored	3.1.10

3.1.1 MMP

The Multicast MAC Protocol (MMP) [18] is an extension of the IEEE 802.11 MAC in DCF (Distributed Coordination Function) mode.

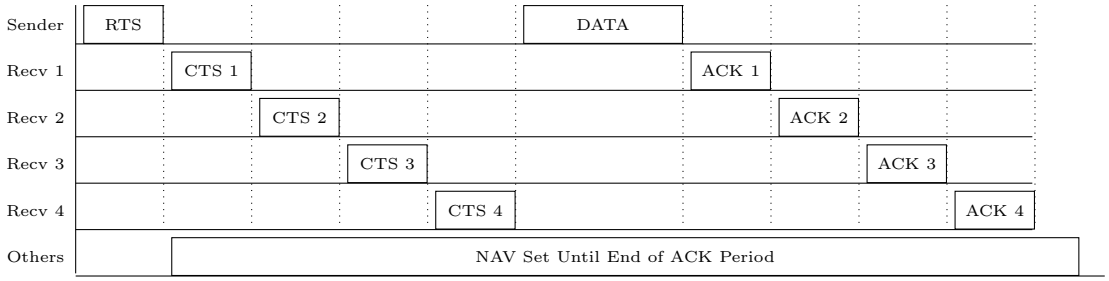


Figure 3.1: Message propagation in MMP (taken from [42])

The Request To Send (RTS) message of MMP contains the addresses of all nodes that should receive the multicast message. When a node receives this RTS, it waits a certain time, correlating to its position in the RTS, and sends a CTS. When the slots for all CTS messages have passed and the sender of the RTS has received at least one CTS, it starts the transmission of the data packet. After the transmissions, the acknowledgment messages are sent by all of the receivers in the same order as the CTS messages. Figure 3.1 shows an example for one sender and 4 receivers. MMP does not directly address the problem of unidirectional links. It is included here nevertheless, because it is the basis for NMAC (section 3.1.2), which can use unidirectional links.

3.1.2 NMAC

NMAC (Neighbor MAC) [42, 44] is a modification of the Multicast MAC protocol (see section 3.1.1). As its name suggests, MMP was designed for multicast, not for broadcast. In NMAC its behavior has been changed to enable broadcast transmissions, and to enable it to use unidirectional links. A neighborhood discovery protocol is used to detect any unidirectional links.

Because of the neighborhood discovery protocol, each node knows how many neighbors it has and addresses them all in the RTS packet. When a node receives an RTS message it waits for a time corresponding to its position in the RTS before transmitting a CTS. If it has received a certain percentage of the expected CTS messages, the sender of the RTS transmits the data package after the time for all CTS messages has passed.

Nodes that are connected by a unidirectional link can also be addressed in the RTS message and also transmit a CTS, but are not counted when the percentage of received messages is evaluated, as their CTS messages cannot be received by the sender of the RTS anyway.

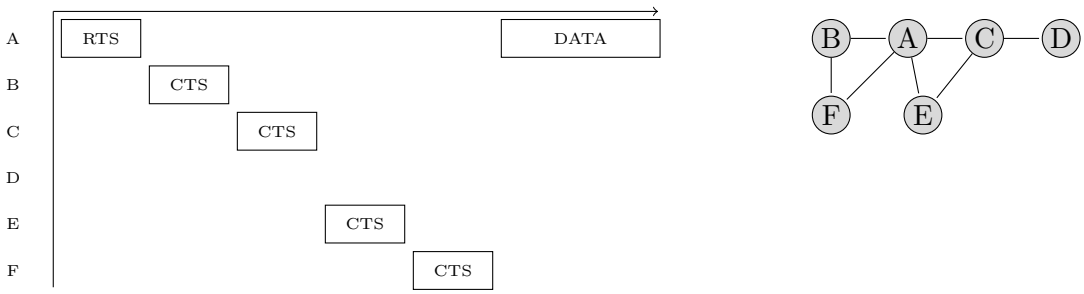


Figure 3.2: Message Propagation in NMAC (taken from [42])

The functionality of NMAC is depicted in figure 3.2. Node A wants to transmit a message, and has nodes B, C, E and F as neighbors. It includes their addresses in its RTS message and waits until all four have had the chance to transmit their CTS messages. Then it checks if the amount of CTS messages received is high enough, e.g. 75%, and transmits the data packet if it is.

3.1.3 BW_RES

Another extension to IEEE 802.11 is BW_RES [59]. It is based on the principle of forwarding CTS packets to all nodes that may disturb the planned communication. To determine how far a BW_RES message must be forwarded, the transmission strength of all nodes must be known. The lowest one equals one unit, the highest one N units. The authors show that a CTS message needs to be retransmitted 2N-1 times to ensure that it is heard at least N units distant. A node that receives a CTS message waits between 0 and 6 SIFS (Short Interframe Spacing) before transmitting the BW_RES packet to prevent collisions. While this approach ensures that data communication in the presence of unidirectional links is possible, it delays the transmission and increases the network load proportionally to the maximum difference in transmission strength of nodes.

3.1.4 ECTS-MAC

ECTS-MAC (Extended Clear To Send MAC) [42, 44] is a contention based protocol for sparse networks with rare communication. It is similar to BW_RES [59] (see Section 3.1.3), because it also tries to forward the CTS message to reduce the probability of collision. Unlike BW_RES, it does not calculate distances and power levels. Also, all Extended Clear To Send (ECTS) messages are sent at the same time, whereas all BW_RES messages are sent one after another. This leads to more collisions of ECTS messages, but saves a lot of time.

When a node receives a CTS message it forwards it with a certain probability. Experiments have shown that 50% is a suitable value for sparse networks. If the probability is less, the ECTS message is not received by enough neighbors. If it is higher, the ECTS packets collide more often. These collisions are also the reason why the ECTS-MAC should only be used in sparse networks. To a certain extend, this effect is alleviated by reducing the probability of sending, but this also leads to more nodes that do not receive any ECTS messages. ECTS-MAC uses a neighborhood discovery protocol to detect unidirectional links. This is necessary to enable transmitting via a unidirectional link, because acknowledgments need to be forwarded to the sender using a second node.

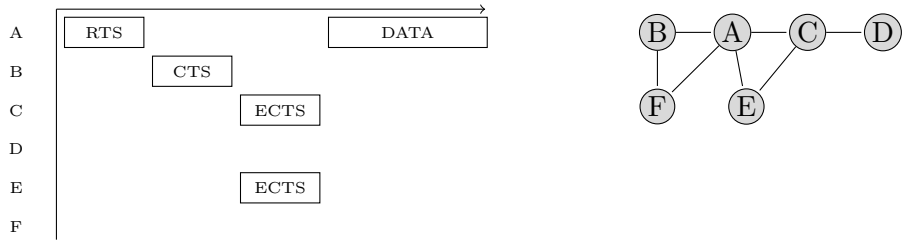


Figure 3.3: Propagation of ECTS Messages (taken from [42])

Figure 3.3 shows an example of a sparse network, where the probability that an ECTS message is generated upon reception of a CTS message is set to 50%. Node A wants to transmit to node B, which transmits a CTS message in response to the RTS from A. Nodes A, C, E and F receive the CTS message, and two of them generate an ECTS message. Both ECTS messages are sent simultaneously. After the time for just a single ECTS transmission has passed, node A can transmit its data packet.

3.1.5 AMAC

AMAC [80] is built on top of the Sub Routing Layer (SRL) project [61], which is used to detect unidirectional links. When SRL is used with a routing protocol, it provides the abstraction of a network with only bidirectional links. To do this, it must identify unidirectional links, and find a suitable reverse route leading through multiple nodes. SRL uses a reverse distributed Bellman-Ford algorithm to find reverse routes and also monitors the network for link changes. AMAC uses the information from SRL to make unidirectional links usable on the MAC layer. Four new types of messages are introduced to make communication over unidirectional links possible by forwarding protocol messages through neighboring nodes. The four types of messages in AMAC are: XRTS (Extended RTS), XCTS (Extended CTS), TCTS (Tunneled CTS) and TACK (Tunneled ACK). XRTS and XCTS are used to inform the nodes about the communication that cannot normally receive receive RTS and CTS, but which may still disturb the transmission, e.g. because of their long communication range. The TCTS is sent by the destination of an RTS message if it was received over a unidirectional link. In this case direct sending of a CTS is not possible, therefore the TCTS must be forwarded by a neighboring node that can communicate with both participants of the communication (tunneled). Once the communication is complete, the destination sends a TACK message which is again tunneled for the same reason.

3.1.6 PANAMA

PANAMA (Pair wise Link Activation and Node Activation Multiple Access) [5] consists of two different algorithms. PAMA-UN (Pair wise link Activation Multiple Access Unidirectional Networks) is intended for unicast communication, while NAMA-UN (Node Activation Multiple Access for Unidirectional Networks) supplies broadcast communication. PANAMA is based on CDMA (Code Division Multiple Access) and uses DSSS (Direct Sequence Spread Spectrum). DSSS is based on spread codes which spread the signal over a large spectrum, resulting in a resistance to narrow-band disturbances and a better signal to noise ratio. Also, time is divided into slots in PANAMA.

In each slot, nodes with orthogonal spread codes can transmit simultaneously. Codes are reassigned every slot, nodes compete for the codes by comparing their priority. The node with the highest priority has won the medium and all its neighbors configure their radio modules to use its spread code. The link characteristic (bidirectional or unidirectional) is a part of the bandwidth value which is featured in the computation of the priority. The main difference between NAMA-UN and PAMA-UN is the way priorities are computed. In NAMA-UN, the priority depends on the sending node, whereas in

PAMA-UN it is calculated using all incoming links of both nodes participating in the communication. The calculation of priorities is the most complex part of PANAMA. Each node needs to know the exact priorities of all its neighbors at any time. The priority is 0 if the bandwidth from the sender to the receiver is 0 (unidirectional link from this node to its neighbor). A node wins the contention if its priority is higher than that of all its neighbors and there is no upstream-only-neighbor (neighbor with a unidirectional link to this node) that uses the same spread code. The priority of all neighbors k in slot t is calculated as follows: $p_k^t = \sqrt[bw_k]{Rand(k+t)}$ where bw_k is the bandwidth of node k. *Rand* is a random function which delivers a number between 0 and 1. The value of p_k is set to 0 if bw_k equals 0.

In PAMA-UN the computation of the priority depends on all incoming links of both participating nodes x,y,: $p_{(x,y)}^t = \sqrt[bw_{(x,y)}]{Rand(x+y+t)}$. Both protocols, PAMA-UN and NAMA-UN depend on knowledge about the two-hop neighbors of a node. To determine this, a neighborhood protocol is used, which transmits updates about the neighborhood of a node regularly. Each node can compute its two-hop neighborhood by combining these messages from all its one-hop neighbors. The update messages can contain information about multiple links. This information contains the ID of the neighbors, the status of the link (bidirectional or unidirectional), the type of change (add or delete a link/neighbor) and the current bandwidth. Depending on whether mobility is used or not, the interval at which these messages are sent can be adjusted.

3.1.7 LMAC and AI-LMAC

LMAC [73] (lightweight medium access control) is based on a TDMA scheme. Time is divided into frames and slots. Each node reserves a slot during which it can send. This slot re-occurs every frame. Every slot is used to send a control message followed by data payload.

Table 3.2 shows the contents of an LMAC control message. Its total size amounts to 12 Bytes. It contains the identity of the sender and its slot number followed by the most important field `Occupied Slots`, which represents a Bit mask of Slots. An unused slot is represented by a 0 while a 1 represents an occupied one. Thus it is possible for every node to determine unoccupied slots by combining the control messages of its neighbors. This is done by performing a simple OR operation on the fields `Occupied Slots` of all received control messages. The distance to the Gateway is also transmitted, along with information of overheard collisions. Finally, the ID of the destination and the size of the data unit are given. The initialization of nodes is started by the gateway, which defines its own slot and is used for synchronization.

Table 3.2: The control message used in LMAC

Description	Size (bytes)
Identification	2
Current Slot Number	1
Occupied Slots	4
Distance to Gateway	1
Collision in Slot	1
Destination ID	2
Data Size (bytes)	1
Total	12

After one frame, all direct neighbors of the gateway know its slot and choose their own ones. This information is transmitted to their neighbors who synchronize on these messages. After each frame, a new set of nodes with a higher hop distance from the gateway is synchronized until every node knows its slot. These slots only need to be locally unique, as the nodes only compete with others up to two hops distant. To conserve node energy, a node's transceiver is turned off for the remainder of the current slot when it is not addressed in the control message. As slots are computed just once in LMAC, this protocol is not suitable for mobile sensor networks, where nodes can enter and leave other nodes' radio neighborhood at any time. Moreover, links need to be stable over time, which is not the normal case in wireless sensor networks, as the experiments presented in chapter 2 have shown.

AI-LMAC is introduced in [64]. It is an enhancement of LMAC which allows dynamic reallocation of slots, depending on the network load. The authors assume a routing tree which leads to a sink and optimize the slot usage along the branches of this tree. This is realized by the usage of so-called Data Distribution Tables, which are used to determine the network load which results after a query from the sink. With this information, slots can be reserved according to the presumed needs.

3.1.8 MLMAC

Another TDMA based protocol is MLMAC [41, 43], an extension of LMAC. The main difference between LMAC and MLMAC is their intended scenario. While LMAC assumes a static sensor network where communication takes place between nodes and the gateway, all nodes are assumed to be mobile and communicate among each other in

Table 3.3: The control message used in Mobile LMAC (MLMAC)

Description	Size (bytes)
Identification	1
Slot number and Status	1
Occupied Slots	1-2
Identity of the Synchronization	1
Age of Synchronization	1
Total	5-6

MLMAC. There may be one or many gateways or maybe there is none. MLMAC does not depend on a gateway to start the synchronization, instead, it is fully dynamic.

The choice of scenario has a number of consequences. First, there may not be a gateway to start the synchronization. Second, the chosen slots are not fixed in time. Due to mobility, it may become necessary for a node to choose a new slot when it enters a different radio neighborhood. Third, MLMAC differentiates between bidirectional and unidirectional links, whereas LMAC assumed that all links were bidirectional.

The node that wants to send a packet first starts the synchronization. This removes the necessity to use the field `Distance to Gateway` for synchronization. Even when it is not used for synchronization, the field `Distance to Gateway` could be used to support routing decisions in stationary sensor networks. In mobile sensor networks however, this distance could not be determined only once and stored for further use, as the mobility of nodes will lead to a change in topology after a certain time. This time depends on the range of the transceivers and on the speed of the nodes of course, but eventually the change in topology will take place. The field `Distance to Gateway` is removed from the header of MLMAC. The fields `Destination ID` and `Collision in Slot` are not used either, because of the hardware the authors used for their feasibility study. There was no way to shut down the transceivers and the radio module used a built in checksum to discard faulty packets. Thus, collisions could not be detected directly and this part of LMAC was not implemented. As this decision was based solely on the used ER400TRS radio modules [11], the authors state that it could be revoked when a different platform is used.

MLMAC's control message format is shown in table 3.3. This control message is quite different from the one used in LMAC. As the intended scenario contained a smaller sensor network, the field containing the identity of the sender was reduced to one Byte.

`Slot number and Status` contains five bits for the senders slot and three bits for its status. The field `Occupied Slots` is used exactly as in LMAC, only its size is reduced.

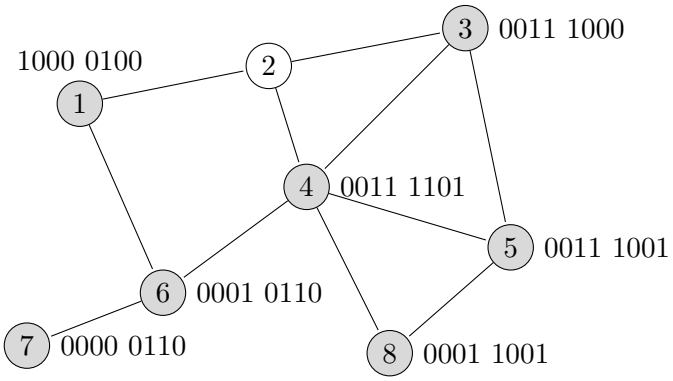


Figure 3.4: Occupied slots as seen by each node (taken from [41, 43])

The field `Occupied Slots` in the control message of a node contains the used slots of all its neighbors and itself. In the case of node 4 on figure 3.4 for example the slots 3, 4, 5, 6 and 8 would be marked as used, which results in a representation as 00111101. Note that in this example the third bit from the left represents the third slot. The figure shows how slots are chosen with a simple example containing only eight nodes. In this example you can see that node 2 is not synchronized yet. It receives the control messages from its neighbors and combines them. 10000100 (from node 1) | 00111000 (from node 3) | 00111101 (from node 4) = 10111101 (seen on node 2).

This means that node 2 can choose between slots 2 and 7. If it chooses slot 2, its control message would contain the Bit mask 11110000 in the field `Occupied Slots`, as node 2 receives messages from nodes 1,3, and 4 and adds its own choice. If it chooses slot number 7 the field `Occupied Slots` would contain the Bit mask 10110010. Note that this method solves the hidden station problem. The number of slots can be chosen between 3 and 16 in the prototype implementation, thus the size of `Occupied Slots` varies between 1 and 2 Byte. If more slots per frame are needed, the size of the field `Occupied Slots` grows. Thus far, the slots are chosen in the same way as in LMAC, except for the fact that the synchronization is started by a node rather than by the gateway.

The second difference is the fact that MLMAC stays adaptive even after slots are chosen. The last two fields of a control message are needed, because every node can start synchronization. Due to this fact, it is possible that two distant nodes start a synchronization separately, as both of them assume that they are the first to send. Their

neighbors would synchronize with them and increase the `Age of Synchronization` by one before retransmitting. In this case, two different synchronizations would be flooding the network and meet somewhere in between the two starter nodes. At this point, nodes would realize that some of their neighbors use a different synchronization by comparing the field `Identity of the Synchronization`. Now the field `Age of Synchronization` is compared. If the received value is equal to or higher than the local value stored in a node, this node becomes unsynchronized again and tries to find a new slot.

Due to the mobility of nodes, a node X may leave the radio range of node Y. Both nodes then realize that they do not receive any more control messages from each other and remove the other one from the neighbor list. When X moves into the radio range of another node Z which knows a different node W which uses the same slot as X, the control messages of X and W collide at Z. Therefore, Z does not receive any more control messages in that slot and marks it as unused. Nodes X and W receive the control message from Z and realize that there must have been a collision of control messages. After this, they give up their current slot and try to find a different one.

To determine whether a link is unidirectional or bidirectional, a neighbor list is used. In this list a counter is stored for every neighbor. When a node X receives a control message from node Y which does not contain the slot of node X, X increments the counter for Y in its neighbor list. If the received control message contains X's slot, the counter is decreased. The range of the counter is 0 - N where N can be configured freely. Then, a threshold can be set, from which on the link will be noted as (partially) unidirectional.

MLMAC's state machine is shown in figure 3.5. The rectangles represent states, the arrows transitions between them.

Initially, all nodes begin in the `Wait`-state. When they want to send a message without having received a control message yet, they change into the `Starter`-state (8). When only one node switches to starter, this is a stable state and the node remains there. If another node switched to the `Starter`-state earlier, this node gains knowledge of that fact after some time and switches to the `Sleep`-state (9) from which it will return into the `Wait`-state after a certain time (6).

If a node received a control message from another node in `Starter`- or `Ready`-state while in the `Wait`-state, it synchronizes its local time with that of the originator of the control message and switches into the `Unsync`-state (1). After waiting one frame to overhear all transmitted control messages and calculate used slots, it chooses its own slot and transitions into the `Sync`-state (2). When the node's newly chosen slot becomes

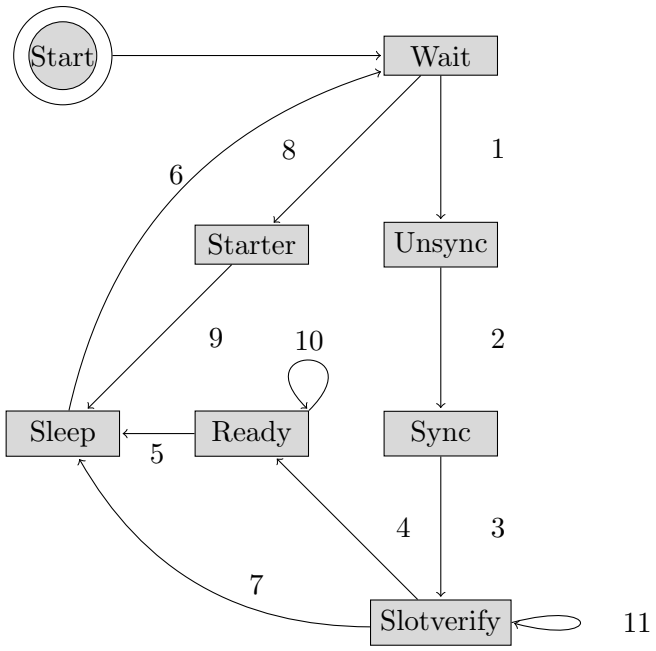


Figure 3.5: The finite state machine used in MLMAC (taken from [41, 43])

active for the next time, it starts to transmit its own control messages in every frame and changes to state `Slotverify` (3). This state is used to verify that no other node has chosen the same slot during the last frame. This would be indicated by a collision of control messages in this slot and lead to a change into the `Sleep`-state (7).

A node X that transmitted a control message can determine if a collision occurred by listening to its neighbors' control messages. If no collision occurred, the neighboring nodes have added X's slot to the field `Occupied Slots` in their control messages. Otherwise they did not. When X receives control messages containing its slot, it knows that no collision occurred because no other node has chosen the same slot. Therefore, it switches into the `Ready`-state (4).

Like the `Starter`-state, this is a stable state as long as no collision occurs. If a collision occurs, there must have been a mistake in the process of choosing slots, and the node returns into the `Sleep`-state (5) and finally into the `Wait`-state to start over again.

Note that MLMAC also distinguishes between collisions on unidirectional and bidirectional links. If a collision on a unidirectional link occurred on a node in `Slotverify`-state (11) or `Ready`-state (10), this node stays in the same state.

3.1.9 MLMAC-UL

MLMAC-UL [42, 44] is an enhancement of MLMAC (section 3.1.8) that focuses on unidirectional links. While MLMAC was able to function in their presence, it only distinguished between control messages received over bidirectional links and those received over unidirectional links to remove negative implications of incoming unidirectional links.

In this section only the changes in MLMAC are discussed, which have been made. Instead of ignoring collisions that occurred because of unidirectional links, MLMAC-UL uses a neighborhood discovery protocol to determine neighbors that can be used to inform the upstream node (originator) of a unidirectional link about the link and make it usable to forward messages.

The first addition is an independent neighborhood discovery protocol, which is similar to the ones used in AMAC and PANAMA (see sections 3.1.5 and 3.1.6). It transmits the neighborhood table of a node periodically infrequently. In the case of changes, only small update messages are sent. The periodic sending of tables is used to remove any errors resulting from loss of update packets.

Another change in MLMAC-UL is the fact that nodes can give up their slots. If a node has transmitted only status messages for a certain time (e.g., 6 frames) it will inform its neighbors that it is giving up the slot and that it may be used by another node. Moreover, a node may not only hold one slot in MLMAC-UL. Rather, each node can use as many slots as it needs by claiming any unused ones, when it has to transmit lots of data messages. Once the send queue is emptied, it can give up the additional slots one after the other. For this to be effective it is useful to define a larger frame size from the beginning, so that there are always enough free slots available (see figure 3.6). This ability to hold more slots was introduced to reduce the delay and make MLMAC a better competitor for contention based protocols.

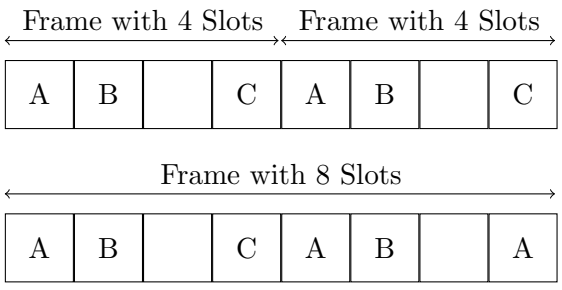


Figure 3.6: Using different Frame sizes (taken from [42, 44])

Each node maintains a list of all its neighbors. Three entries define this list: The link quality, the unidirectionality status and the compressed neighborhood information from that neighbor. The link quality can be good (more than 90% reception rate), medium (between 30% and 90% reception rate) or bad (less than 30% reception rate). The unidirectionality status can be either bidirectional, unidirectional-sender or unidirectional-receiver. The compressed neighborhood list is maintained by the neighborhood discovery protocol and used to identify the two-hop-neighborhood of the current node.

The state machine of MLMAC-UL can be seen in figure 3.7. The arrows in the figure represent the transitions between states and are described in the following.

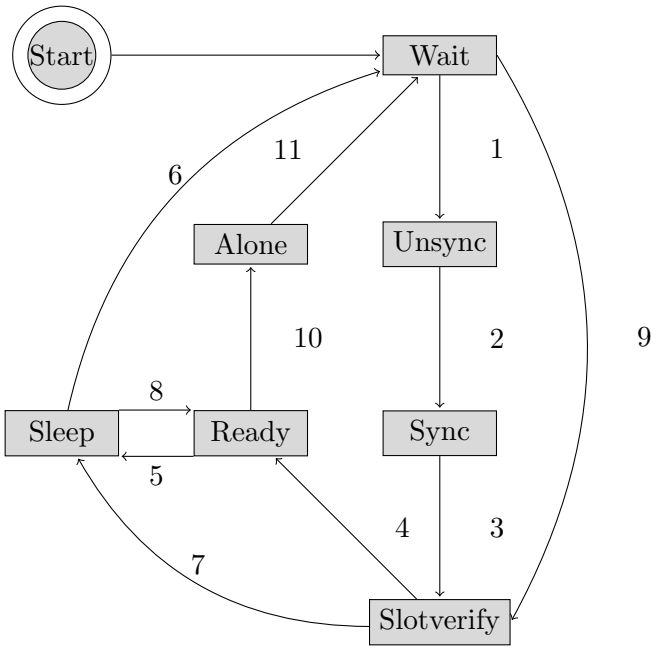


Figure 3.7: The State Machine of MLMAC-UL (taken from [42, 44])

1. When a node needs to acquire its first slot it switches into the state `Unsync`.
2. The node was in state `Unsync` for one frame. It chooses a slot and transitions into the `Sync`-state. If no slot was empty, the node stays in its current state for another frame.
3. When its chosen slot arrives, the node changes to state `Slotverify`.
4. The node sends in its slots. After one frame, it reaches the `Ready`- state.

5. If a negative acknowledgment for the last slot was received, the slot is deleted and the node changes to state `Sleep`.

6. The node returns to the `Wait`-state after a random amount of time.

7. Same as 5.

8. There is data to be transmitted and no neighboring node is transmitting. The node chooses a slot and an identification for the synchronization. After waiting for a random time it transmits the data and switches to `Ready`.

9. If this node did not communicate before or it had previously given up one slot, a new slot is acquired and the node changes into the state `Slotverify`.

10. No messages from neighbors were received for 5 frames even though this node is transmitting. This means that this node is either completely isolated, or has only unidirectional links to others, but no incoming link from any of them. This node switches to the `Alone`-state and does not try to transmit anymore, even when data is available.

11. A message from a neighbor was received, which means that this node is no longer alone, or a certain number of frames (e.g., 200) have passed. The node switches to `Wait` and starts again.

3.1.10 D-MAC

Even though the authors of D-MAC [39, 40] only claim to have developed a MAC-protocol, it is actually a combination of MAC - and routing. In D-MAC, the network is assumed to consist of one sink and multiple nodes, which are connected to the sink by a routing tree. For such a scenario, latencies due to sleep delays and contention near the sink are the main problems.

D-MAC tries to solve these problems by creating a TDMA schedule that enables nodes on a path from leaves of the tree to the sink to wake up one after another. Nodes are classified by their height in the tree (the distance from the sink measured in hops). Nodes that have the same height need to compete for the medium. The problem of contention is solved by introducing a `more data` flag. If a node has won the medium, it knows that one of its siblings has lost, and sets this flag in its message to the common parent. When the parent node receives this message, it knows that it has to stay awake longer, in order to receive the data from its other child.

But even nodes that do not have the same parent might have to compete for the medium. In this case the node that loses the competition can transmit a `More to Send` message in the appropriate period. It is a very small message containing only the address of the father node, informing it that there is still data to be received later.

Even though D-MAC is energy aware and reduces latencies, it cannot be used in the context of this thesis. Its assumptions that links are bidirectional and stable for a long time and the fact that it includes a tree routing mechanism disqualify it for the any-to-any routing protocols that use unidirectional links developed in this thesis.

3.1.11 A Link-Layer Tunneling Mechanism for Unidirectional Links

RFC 3077 [14] has been proposed by the unidirectional link routing group (UDLR) [51] at the Internet Engineering Task Force (IETF) [16]. Its main goal is to make unidirectional links usable in the internet. There, unidirectional links have a different nature then those dealt with in the rest of this thesis. As RFC 3077 deals with internet connections, the links are stable, and unidirectional links exist over a really long period. An example for unidirectional links as mentioned in RFC 3077 are satellite connections, where the satellites can transmit to a lot of receivers (”local” broadcast), but the receivers cannot transmit back to the satellite.

One assumption made by the authors of the RFC is that nodes can be divided into three categories: `Receivers`, `Send-only feed` and `Receive capable feed`. `Receivers` are on the lower end of a unidirectional link, i.e. have an incoming only link. `Send-only feeds`, e.g. satellites, have an outgoing unidirectional link. `Receive-capable feed` are routers that have ”send-and-receive connectivity to a unidirectional link” [14].

Another assumption made is that each router has more than one IP connection, allowing for the tunneling of messages.

The basic idea behind the tunneling mechanism is the forwarding of link layer messages of one interface using the routing layer of another interface on the same node when this link is unusable.

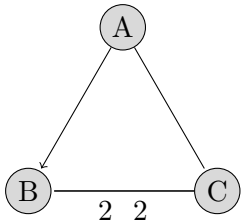


Figure 3.8: Three Nodes with a Unidirectional Link from Node A to Node B

Figure 3.8 shows an example consisting only of three nodes. Node A has an outgoing unidirectional link to node B (Interfaces A1 => B1) and both nodes are connected by bidirectional links to node C (Interfaces A2 <=> C1, B2 <=> C2). Node A can transmit to node B using interface A1 on the unidirectional link, but node B cannot answer. On node B, when a message addressed at interface A1 is passed from the routing layer to the link layer, the tunneling mechanism takes over. It encapsulates the message into an IP message for node C, interface C2, that is handed back to the routing layer and transmitted over interface B2. Upon reception of this message on interface C2, node C forwards it to node A, interface A2 using interface C1. When node A receives the message over interface A2, it is de-capsulated and handed to interface A1, which handles it as if it had been received directly from node B.

This approach offers the possibility of using any routing protocol over the tunneling mechanisms, and hides the existence of unidirectional links from them. It cannot, however, hide the longer delay, which can be a huge problem for timeouts used in the routing protocols. Also, as stated by the authors, this tunneling mechanism does not work "where a pair of nodes are connected by 2 unidirectional links in opposite direction" (using different interfaces). This refers to the fact that all links on the tunnel have to be bidirectional. If the link from node B to node C in the example above was unidirectional, the mechanism described in RFC 3077 would have failed, even though a detour existed.

Additional information about the tunneling mechanism, like e.g. the Dynamic Tunnel Configuration Protocol used there, are not relevant to this thesis and thus are not discussed here. The details can be found in RFC 3077 [14]. For this thesis it is only important to notice that one of the basic assumptions, the fact that all links are stable, does not hold for the wireless networks considered here.

3.2 Routing Protocols

Routing protocols can be classified by different criteria. One of these are the targeted systems, e.g. wired or wireless, sensor network or MANET, mobile or static. After this coarse-grained differentiation, there are a number of criteria that follow: Is the protocol dependent on the application or not, is it data centric/content based, hierarchical or location based? The authors of [1] differentiate whether sensor network routing protocols are data centric, hierarchical or location based, and whether they offer QoS, network flow or data aggregation. In [2] a different classification is used, even though it is also a survey of routing protocols for wireless sensor networks.

For this thesis the way nodes are addressed by queries is important. Three different ways are discussed here: First, addressing the global unique identity of a node. Second, addressing all nodes which fulfill certain criteria, e.g. have measured a temperature value above a certain threshold for fire detection. Third, addressing a group of nodes in a certain area. Once the categories have been separated, different (sub-)types can be defined (table 3.4). These include Distance Vector, Source Routing, Link Reversal and Geographic protocols as well as those based on Diffusion or Agents and those that supply Real-Time guarantees.

Table 3.4: Classification of Routing Protocols surveyed in this Thesis

	Global Identities	Data centric	Location Based
Distance Vector	x		
Source Routing	x		
Link Reversal	x		x
Geographic	x	x	x
Diffusion		x	
Agent Based		x	
Real Time			x

In the following, an overview of these three categories and their representative protocols described in this thesis is given.

Routing Protocols based on Global Identities

The choice of routing protocol can have an enormous impact on the number of transmitted messages and, consequently, on the total number of bytes transmitted. If the

routing protocol always chooses the shortest path the number of data packet transmissions is minimized, no matter which routing protocol is used. The total number of bytes transmitted for one data packet depends on the routing header, though, whose size varies greatly with the chosen routing protocol. In source routing, the source knows the whole path from source to destination and includes it in the packet. This leads to different header sizes, depending on the number of hops. In distance vector approaches, each node along the path knows the way and the source only needs to include the ID of the destination and the next hop. This way, less data is transmitted but additional information needs to be kept in the intermediate nodes.

Route discovery has to be taken into account, too. Many routing protocols supply the shortest path to the destination (least number of hops) but that path has to be established somehow. The source can broadcast a route request for example, which is flooded either through the whole network or only within a certain radius (see expanding ring search in [57]). The handling of the flooded messages and the replies differs quite a lot, depending on whether the protocol in question is a source routing protocol or a distance vector protocol.

Finally, route maintenance is called for. If a node becomes unavailable due to mobility, variations in link quality or energy outage an alternative route is needed. This link break can be detected and repaired by different means. A classical approach is the use of `hello`-messages which are transmitted periodically to detect link failure. When no `hello` (or other) message has been received from a neighbor for a certain time, the link to that neighbor and all paths that use this neighbor as next hop are removed from the routing table. As this approach induces a lot of communication overhead, it has to be avoided in resource constrained networks and especially in wireless sensor networks. When a message cannot be forwarded due to link breaks, there are once again multiple ways to react. Some protocols attempt local repair, others just transmit a route error message back to the originator of the message. This behavior is often found in distance vector routing and source routing protocols respectively. A third way of reacting is the inversion of an incoming link into an outgoing one, as done by link reversal protocols (see section 3.2.17)

Table 3.5 shows the different distance vector, source routing and link reversal protocols surveyed in this thesis. It denotes the type of protocol, Distance Vector (DV), Source Routing (SR), Link Reversal (LR) or Geographic (Geo) and the way of dealing with unidirectional links (ignore their existence, remove their implications or make use of them).

Table 3.5: Protocols Based on Global Identities Discussed in this Chapter

Name	Source	Type	Unidirectional Links	Section
DSDV	[56]	DV	ignored	3.2.1
AODV	[55, 57]	DV	implications removed	3.2.2
AODV-BR	[37]	DV	implications removed	3.2.3
Route Reply Salvaging	[4]	DV	implications removed	3.2.4
NST-AODV	[17]	DV	implications removed	3.2.5
AODV-RPS	[45]	DV	implications removed	3.2.6
Unnamed	[60]	DV	used	3.2.7
DYMO	[10]	DV	implications removed	3.2.8
DSR	[28, 29, 30]	SR	used	3.2.9
DSR-DCU	[83]	SR	used	3.2.10
DSR-CSA	[54]	SR	ignored	3.2.11
LBSR	[3]	SR	used	3.2.12
Full and Partial Reversal	[8]	LR	ignored	3.2.17
TORA	[53]	LR	ignored	3.2.18
GeoTORA	[32]	LR, Geo	ignored	3.2.19

Data Centric Routing Protocols

Data centric routing protocols rely on application knowledge, which is used to identify the destination of messages. This makes them unusable for general purpose networks like the internet, but can increase efficiency in specialized networks like wireless sensor networks. A query in a data centric networks normally consists of attribute-value pairs, e.g. temperature higher than a certain threshold or an area within certain boundaries. As the routing protocols developed for this thesis are specifically designed to be general purpose and do not rely on application knowledge, only a few data centric protocols are surveyed here (table 3.6).

Location Based Routing Protocols

Location based routing protocols assume that all nodes know their physical location, and make routing decisions based on geographical distances. The actual mode of decision can be quite different. In some protocols, all nodes need to know their neighboring nodes, and decide which neighbor should forward a message depending on the distance gained

Table 3.6: Data Centric Protocols presented in this Chapter

Name	Source	Type	Unidirectional Links	Section
Directed Diffusion	[26, 27]	Data Centric, Diffusion	ignored	3.2.13
Solar Aware Routing	[67]	Data Centric, Diffusion	ignored	3.2.14
Push/ One-Phase-Pull	[24]	Data Centric, Diffusion	ignored	3.2.15
Rumor Routing	[7]	Data Centric, Agents	ignored	3.2.16
GEAR	[84]	Data Centric, Geographical	ignored	3.2.21

by that one hop. In others, the forwarding decision is made on the receiving nodes. Each calculates its distance to the destination. Nodes that are nearer forward the messages after a short delay, nodes that are farther wait longer. If a waiting node overheard the forwarding of the message it wants to transmit, it discards the message instead. In yet other protocols a direct line between the originator node and the destination is calculated, and the node that is closest to this line is chosen as forwarding node.

All geographical routing protocols have one problem in common, though. If there are obstacles somewhere between originator and destination, the message is forwarded up to that obstacle and no way further might be found. There are once again multiple ways to react and each protocol has a different preferred method of bypassing obstacles.

Table 3.7 shows the geographical protocols surveyed in this thesis. Their methods of forwarding and obstacle bypassing are discussed in the corresponding sections.

Table 3.7: Location Based Protocols presented in this Chapter

Name	Source	Type	Unidirectional Links	Section
GeoTORA	[32]	Geographical, Link Reversal	ignored	3.2.19
GPSR	[31]	Geographical	ignored	3.2.20
GEAR	[84]	Geographical, Data Centric	ignored	3.2.21
SPEED	[20]	Geographical, Real Time	ignored	3.2.22

In the following, the surveyed routing protocols that are based on global identities are described, followed by data centric protocols and, finally, location based protocols.

3.2.1 Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers

Perkins and Bhagwat propose a routing protocol for mobile nodes that builds upon the distributed Bellman Ford (DBF) algorithm [56] (also used in the internet, in the routing information protocol (RIP) [22]). DSDV proposes a solution to the major problem of DBF, the count-to-infinity problem.

The count-to-infinity problem has its origin in the way weight functions are generated in DBF. Consider a small part of a network as shown in figure 3.9(a). Nodes B, C and D are lined up, the rest of the network is connected via nodes B or D. All nodes advertise their routing tables periodically. In the first step, each node only knows itself and broadcasts its existence. After the first step, each node knows its neighbors, and knows that they are one hop distant. In the second step, this information is also propagated, and all nodes know of their two-hop neighbors and so on, until each node has gained knowledge of all other nodes in the network. The information is stored as distance vector, meaning that each node knows only the distance and the next hop for a certain other node. Node B would have an entry in its routing table consisting of (D, C, 2) (destination, next hop, distance). On node C the entry would consist of (D, D, 1).

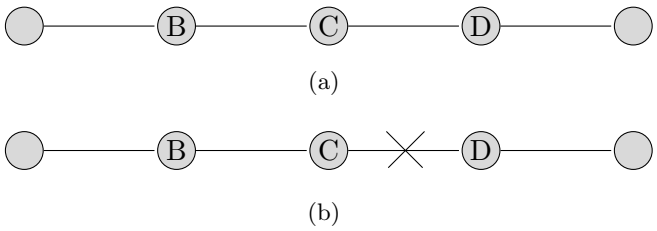


Figure 3.9: A simple subnet

The problem arises when a node dies or a link breaks and no path around the broken link exists, as shown in figure 3.9(b). Node C would detect that it can no longer transmit to node D. Therefore, it would no longer advertise the route to node D. When routing information is propagated the next time, node B still advertises the fact that it knows a route to node D of length 2, which node C would receive and write into its routing table (incremented by 1). The next routing information message from C would then contain this value of 3, leading to a value of 4 entered into the table at node B and so on until infinity. The fact that node D cannot be reached at all is only realized when infinity (i.e. in this case, the highest possible value) is reached. Until that moment, routes are still advertised and contain a routing loop between nodes B and C.

DSDV solves the count-to-infinity problem by introducing destination sequence numbers. Every time a node transmits a routing information message, it increments its sequence number. The routing information that is propagated also contains the most recent sequence number received from each destination.

The second case in which a sequence number is increased is when a link breaks. A node that detects a link break increments the sequence number for all destinations that are reached using that link and sets the weight function to infinity. An important difference between the two increment operations is the amount by which the sequence number is incremented. A node that broadcasts its routing table increments the sequence number by 2, always resulting in even numbers. At a link break, the sequence number is incremented only by one. When a node receives an update message for a destination with a higher sequence number than the stored one, it changes the entry. In the example described above node B would have an even sequence number for node D in its routing table, e.g. 50, with a distance of 2. Upon link breakage, node C would increment its sequence number for node D (it has to be equal or greater than that stored at B) by one, resulting in an odd number, e.g. 53 and a distance of infinity. Upon reception of the next update message, node B would realize that the sequence number is higher, and also enter the value of infinity into its routing table.

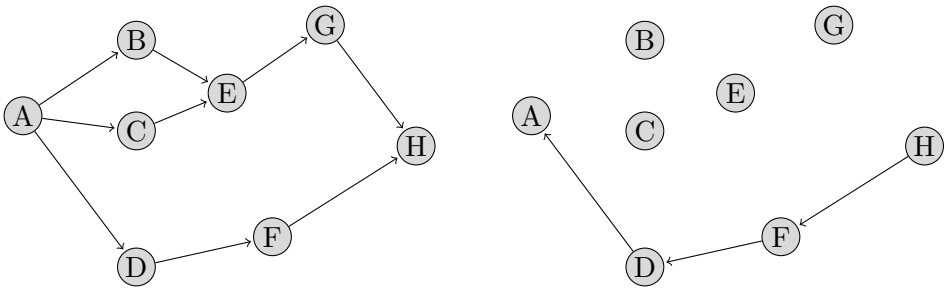
If, at some point in the future, node D would become available once more, its sequence number transmitted would be 54 or higher, resulting in an immediate replacement of the infinity value with the distance contained within the update message (plus one).

The authors of [56] also address the problem that the shortest route is not always propagated faster than all others. In such a case, propagating all routing information as soon as it changes leads to a waste of bandwidth, as the routing table is changed multiple times in each update cycle. The authors propose to delay such updates, and only propagate important changes at once. An important change would for instance be the breaking of a link and the subsequent setting of infinity as weight function. All other changes are only propagated at regular intervals.

While DSDV has solved the count-to-infinity problem, it still has a number of drawbacks. First of all, each node keeps routing information for all other nodes. This information does not occupy as much memory as in link state protocols, but still needs a lot of memory in large networks. Second, most important to this thesis, it can only operate on bidirectional links. A unidirectional link between nodes C and D on figure 3.9(b) would lead to the afore described behavior on node C's side. Even worse, node D would receive the update messages from node C and propagate the non-existing path in each update cycle.

3.2.2 Ad hoc On-Demand Distance Vector Routing

As its name suggests, AODV [57, 55] is a routing protocol based on distance vectors. The route to a destination is not stored completely in one node. Rather, a source node S knows only the next hop A on the route to a destination D and the distance to D in hops. Packets for D are sent to A, where the following hop is stored and so on. This way, the routing information is stored in a distributed manner. Opposed to source routing, routes do not have to be transmitted with the packets, which results in smaller messages. AODV is also a reactive protocol, which means that routes are only determined when they are needed.



(a) Route Request Propagation (b) Route Reply Propagation

Figure 3.10: Route Discovery in AODV

The routing process is carried out in two phases, which are route discovery and route maintenance. When S wants to send a packet to D for the first time it broadcasts a route request packet (RREQ). This RREQ is flooded through the whole network or for a specified number of hops (expanding ring search). Sequence numbers are used to realize duplicate suppression and to prevent the count to infinity problem, just like in DSDV (section 3.2.1). Figure 3.10(a) shows the propagation of a RREQ message for node H. The source node A broadcasts it to its neighbors B,C and D, which enter node A into their routing table as a direct neighbor. Nodes B, C and D rebroadcast the message in turn to their neighbors E and F, which enter them into their routing tables as neighbors. They also enter node A into their routing table, with a distance of two hops and the node the RREQ was received from as next hop. One of the messages from either node B or C arrives at node E first and is forwarded. The other one is recognized as a duplicate and discarded. Node E transmits to node G and node F to node H, which update their routing tables in the same way. Node H is the destination and acts accordingly. Node G then transmits the RREQ to node H, but node H identifies it as a duplicate and

discards it. If the whole network is flooded with the RREQ messages, every node knows all of its direct neighbors when all RREQ packets are discarded. When expanding ring search with size N is used this is only true for all nodes within N-1 hops from the source.

Figure 3.10(b) shows the path that is taken by the route reply message (RREP). When receiving the first RREQ, the destination node H sends a RREP back along the way the RREQ took. The RREP is addressed to node A and forwarded to node F from which the RREQ was received. It is then forwarded to node D and finally to node A. At this moment all nodes on the route, nodes A, D, F and H know routes to nodes A and H.

Once links are established, route maintenance is called for. There are multiple ways in which this can be triggered. Link breakage can be detected if a link - or network layer acknowledgment is requested but not delivered. Another possibility is that no `Hello Message` was received for a certain time (see RFC 3561 [55] for more details). There are different actions taken when links break, according to the previous status of the link. If the link in question was not in use it is simply marked as invalid. If it was part of an active route, i.e. packets have been sent along this link recently or the link break was detected because a packet could not be transmitted, further action is required. The breakage can be confirmed first, e.g. by sending a RREQ with the ID of the next hop and a time to live of 1. Once it is confirmed, all routes that use the broken link have to be removed. To remove the routes from other nodes, too, a route error message (RERR) is sent to all nodes that are listed in the precursor lists for the destinations of the broken link. A precursor list contains all nodes from which packets for a certain destination were received. The RERR message is then either unicast or broadcast, depending on the number of neighboring nodes in the precursor lists. Instead of this, a node may perform a local repair. It creates a RREQ for the destination with an increased sequence number, which is transmitted in the normal way. If it receives only RREP messages with a greater distance, it should send a RERR message with a flag indicating that it has a new, longer route. If the route has the same length, no RERR message needs to be sent.

This is only the basic functionality of AODV. There are many more optional features. The one that is interesting for this thesis deals with AODV operation in the presence of unidirectional links. The algorithm described above would fail if there was even a single unidirectional link in a route. Consider again the network shown in figure 3.10(a). If the link between nodes D and F was unidirectional and communication was only possible from node D to node F, the RREQ propagation would have worked exactly as described. The RREP on the other hand would only have reached node F, and after

a certain number of retries it would have been discarded. As the destination sent only one RREP, no route can be found. If the status of the link is persistent, subsequent RREQs from the source would fail too, even though another, completely bidirectional path exists.

To prevent this problem, AODV uses blacklisting of nodes. If AODV operates in an environment where unidirectional links can occur, nodes that transmit a RREP message set a special flag, indicating that they need an acknowledgment from the next hop. Nodes that do not transmit the so-called RREP-ACK message are entered into the blacklist and not considered in future routes. RREQ messages from nodes in the blacklist are simply discarded.

The route discovery mechanism that uses RREQ and RREP messages as well as the sequence numbers used for duplicate detection in AODV are the basis for parts of this thesis. Its way of dealing with unidirectional links by removing them is where the differences will be seen, as all proposed routing algorithms use them in order to increase performance.

3.2.3 AODV-BR: Backup Routing in Ad hoc Networks

AODV-BR [37] is an enhancement of AODV (section 3.2.2), that uses a mesh structure to supply multiple paths. The proposed algorithm is actually independent of AODV. It works with any distance vector routing protocol that uses a discovery mechanism based on route requests and route replies. The main achievement of the protocol is to build multiple routes without sending additional control messages. This is possible because of the broadcast character of the medium. Every node that overhears a route reply packet and is not the addressed next hop discards this packet in AODV. In AODV-BR these nodes enter the node from which the route reply was received as next hop to the destination into their routing cache. This way, a structure similar to a fish bone is constructed.

When a link breaks, the node that detected the break broadcasts the data packet it tried to send, with a flag indicating that this message should be sent using an alternate route. A neighboring node that receives this message and has overheard the route reply that created this route forwards the message to the next hop. This way, a detour of one hop is taken, which ensures the data packet's delivery where AODV would have discarded it. Also, a route error packet is transmitted to the source, so that a new and possibly better path can be established. Please note that the message still has to traverse all nodes that are on the original route.

The authors evaluate their approach in a simulation and compare it to AODV. The results of their simulation show that AODV-BR improves the performance in low traffic scenarios. In heavy load situations however, it performs worse than AODV, because the broadcasts and possible duplicate packet transmissions (over 1-2 hops) in case of a link break lead to congestion and collisions. This is partly a link layer problem, as the simulated[1] 802.11 DCF medium access control uses different protocols for broadcast and unicast. Unicast messages are transmitted using request-to-send (RTS) and clear-to-send (CTS) messages to reserve the medium for a certain time. Broadcast messages are transmitted using a carrier sense multiple access with collision avoidance (CSMA/CA) mechanism, which listens to the medium before transmitting. Another problem of the evaluation is the usage of the random waypoint mobility model, because it contains abrupt stops and sharp turns.

The proposed way of finding alternate routes is similar to overhearing, and suffers from the same problems: Unidirectional links cannot be used because the upstream nodes do not know about their existence. Still, if the alternative link that should be used for the detour is bidirectional or points at the next hop at the time of the broadcast, this protocol might improve the performance. One of the protocols proposed in this thesis, Buckshot Routing, works in a similar fashion. For details see chapter 4.

3.2.4 Salvaging Route Reply for On-Demand Routing Protocols in Mobile Ad-Hoc Networks

In [4] the usage of a so-called salvaging route reply (SRR) is proposed. When the transmission of a route reply packet (RREP) fails, it is normally discarded. In the proposed protocol, the intermediate node which could not transmit the RREP tries to find a different route to the destination of the RREP instead. It achieves this by searching for an alternate route in its route cache (for multi-path protocols) or conducting a SRR route discovery within a certain radius (for non-multi-path protocols). A SRR route discovery means that all nodes within the specified radius (one hop in the simulation mentioned in [4]) check their routing tables for alternate routes to the destination of the RREP. If one is found, the node(s) become part of the path. The authors assume that all links are bidirectional. They motivate their assumption with the fact that the authors of [45] declare that the advantages of using unidirectional links do not justify the costs. If unidirectional links are absolutely necessary, link-layer tunneling [14] should be used. The first argument is discussed in section 3.2.6, the link-layer tunneling in section 3.1.11.

[1] According to [25], current WLAN hardware does not use RTS/CTS mechanisms, so the results might have been different if the protocol had been evaluated on real hardware.

3.2.5 Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment

Gomez et al. introduce Not So Tiny - AODV (NST-AODV) [17], which is a smaller Version of AODV for low-power wireless personal area networks (LoWPANs). The protocol is based on an earlier AODV version for LoWPANs called tiny AODV, hence the name. In both protocols sequence numbers are omitted to reduce the network load. The main contributions of NST-AODV are the usage of link layer notification, buffers for packets that could not be transmitted and layer 3 retransmissions.

The AODV standard (see section 3.2.2) does not define how link breaks are detected. It proposes three possible ways. First, the usage of periodically transmitted `hello`-messages. If a certain number (default: 2) of `hello`-messages are missed, the link is defined as broken. The default value for hello packet transmission is 1 second, leading to a link failure detection time between 1 and 2 seconds, depending on whether the link break occurred directly before hello packet transmission or directly thereafter. Second, a flag in the route reply messages can be set, indicating that an acknowledgment should be sent (layer 3 ACK). Third, if available, link-layer acknowledgments can be used, where the absence of an acknowledgment signifies a link break or a unidirectional link. As only bidirectional links are used, there is no differentiation needed. NST-AODV defines link-layer acknowledgments as standard. `Hello`-messages and layer 3 acknowledgments are not used, because they present unnecessary overhead, and because the time needed to detect a link break on the link layer is much shorter (tens of milliseconds opposed to 1-2 seconds).

There are two buffers which are used to store packets when no route is available and route discovery takes place or when a link break is detected. In the second case, a route error packet would normally have been sent and the message for which the link break was detected would have been discarded. Instead it is now buffered because of the assumption that the link break may be only short lived.

The layer 3 retransmission targets exactly that case, because link layer retransmissions use a much smaller timeout. That way, if a break is only temporary, the layer 3 retransmission can deliver the packet without much additional overhead, only the buffers mentioned are needed and the packet is transmitted a few times more. This is still a great advantage, as a route error message normally leads to a new route discovery from the source, which floods the whole network.

There are a few problems with the evaluation of the protocol. The authors present experimental results, but a detailed simulation is missing. The method used to introduce link failures is of debatable quality, as nodes were simply turned off for a short period of time. In this special example, the proposed protocol was better than the basic version, because of the layer 3 retransmissions. The exclusion of unidirectional links is a common problem, but in this case there is also no evaluation for mobility of nodes. However, the proposed link-layer acknowledgments are a good way of detecting broken links.

3.2.6 Routing Performance in the Presence of Unidirectional Links in Multihop Wireless Networks

Marina and Das quantify the advantage of using unidirectional links instead of simply eliminating their impact on a routing protocol for multihop wireless networks [45]. Their results show that the gain in connectivity is nearly zero, while the shortest path costs improve. They say that these advantages are removed when the cost of hop by hop acknowledgments is accounted for, too. Their method of measuring the connectivity is to compute the average number of strongly connected components and the size of the longest strongly connected component. To create the unidirectional links they used 3 different approaches. First, two ranges were defined, long and short, and the nodes all were set to operate in one of them with the same probability. In the second approach the range was set completely at random within certain boundaries. The third approach was based on a topology control mechanism which reduces the transmitting power of a node as long as it still reaches some of its neighbors, so that nodes that are further away and could be reached using higher transmission power are reached using multiple hops instead. This was thought to conserve energy, but in fact it has been shown in [48] that the overall energy consumption in the network increases.

Their results show that there is a gain of about 15% in the size of largest component when unidirectional links are used and the short range is set to half the long one in the first approach. The communication cost is reduced by about 10%.

Because of the conclusion that it is not worth the trouble to use unidirectional links, they present a so-called Reverse Path Search Technique (RPS), which increases the success of AODV in removing the implications of unidirectional links. In RPS every node that receives a route request (RREQ) for the first time rebroadcasts it just like in AODV. The main difference is that multiple routes are used. When a node detects the loss of a RREP message, it determines that the link it wanted to use must be a unidirectional incoming one. Therefore it chooses one of its other routes back to the source. If all routes fail, a backtrack route reply (BRREP) is sent, which tells the node

from which the RREP was received that the current node has no route to the source anymore. This so-called `upstream node` then performs the same algorithm.

The evaluation method of taking the length of strongly connected components seems to be a good one, but misses the fact that links are not stable over time. All three approaches for creating unidirectional links were used in a static topology without varying the transmission range once the network was set up. The second part of the paper which describes and evaluates RPS uses only blacklisting and `hello`-messages for comparison, two methods which also eliminate unidirectional links. Even though a static topology was used, the usage of unidirectional links still brought an increase of 15% in the largest component. For a non-stable topology the value would be even higher, as the protocols that rely on bidirectional links suffer much more from topology changes than those proposed in this thesis (see chapters 4 and 5).

Under the assumption that unidirectional links should be ignored, the protocol seems to work fairly well. The overhead of building multiple routes seems to be a fair price to pay for the gain in robustness. If a bidirectional path exists, it is found in time. This makes the algorithm suitable for Mobile Ad-Hoc Networks, and even for wireless sensor networks with stronger nodes, i.e. with medium sized storage and fairly good transceivers.

The assumption that the gain in performance is not worth the overhead for using unidirectional links can be interpreted in two ways, though. The first possibility is to say that the overhead is too costly and therefore unidirectional links should be removed as the authors of [45] proposed. The other possibility is to say that the overhead has to be reduced, which is the focus of this thesis, as the overhead of the reviewed protocols ($O(n^2)$) is indeed intolerable.

The reverse path search technique could also be used in a modified version to incorporate unidirectional links. In fact, one of the routing protocols proposed in this work has some similarities to the one presented in this section, as they are both distance vector protocols and take care of the problems that arise for AODV when unidirectional links exist. The difference is, that RPS removes the implications of unidirectional links, while the protocols presented in this work use them to gain better performance.

3.2.7 A Routing Algorithm for Wireless Ad Hoc Networks with Unidirectional links

The author of [60] proposes a routing protocol based on distance vector routing. Even though prominent examples of distance vector routing protocols like AODV or DSDV cannot use unidirectional links (see sections 3.2.1 and 3.2.2), the proposed protocol can. In this protocol, every node in the network periodically exchanges beacons (explained later) with its neighbors. Each node retains a list of all neighbors a beacon was received from called `Nodesheard`, a matrix of dimension $n * n$ called `D`, where n is the number of nodes in the network and every entry consists of a tuple (sequence number, distance). The node also maintains two vectors called `To` and `From` in which the sequence number, distance and next hop are stored for each destination to which this node can send (`To`) and for each source, which sends to this node (`From`).

The beacons transmitted periodically include `Nodesheard`, which is used to distinguish between unidirectional and bidirectional nodes. If a node receives a `Nodesheard` which includes its own ID, it knows that the link to the sending node is bidirectional. If the ID is not enclosed, the link is unidirectional. Nodes also transmit their matrix `D` periodically, which is used to build routes. Because of its size the matrix is transmitted less frequently than `Nodesheard`.

This protocol is a proactive one, keeping routes from a node to every other node up to date all the time. This is fitting for the scenario envisioned by the author, as he assumes low mobility and high data traffic. He assumes that unidirectional links occur either because of the difference in battery power which causes a node with less power to diminish its transmission strength, or because of strong interference in one place. This is called a persistent phenomenon in the paper. Transient phenomena, where links change from unidirectional to bidirectional, are also mentioned, but not discussed further. Unfortunately the paper does not present any simulation or experimental results.

Another problem hinted at in the paper is the problem of cooperation with the MAC layer, as the MAC protocol used by a routing protocol for unidirectional links needs to be able to use those, too. See section 2.4 for more information about this problem.

The main performance problem of this algorithm is based on the fact that it is proactive. The matrix `D` stored on each node and transmitted periodically means that $O(n^2)$ space is needed on each node and, even worse, messages of the same size have to be transmitted. This is a major drain on energy and bandwidth. Other distance vector protocols like DSDV (section 3.2.1) and AODV (section 3.2.2) only need $O(n)$ storage and message size. This overhead disqualifies the protocol for the usage in large wireless sensor networks and reduces its usability in Mobile Ad-Hoc Networks.

3.2.8 Dynamic MANET On-demand (DYMO) Routing

DYMO [10] is the successor of AODV (see section 3.2.2). Like AODV, it is based on distance vectors and consists of two parts, route discovery and route maintenance. Routes are created using route request (RREQ) and route reply (RREP) messages, link failures are handled by transmitting route error (RERR) messages. Many features and mechanisms are the same as in AODV, therefore only the most important differences are noted here:

Optional Values or Fields Many of the values that were mandatory in AODV are now optional. The authors state that the protocol will work without all optional fields, but its performance might degrade. An example for this is the distance from a node to the destination which had to be recorded in the routing table in AODV. It can be omitted in DYMO, but the chosen routes may be suboptimal.

Weight Function While AODV uses the hop count as weight function, the choice of weight function is left to the user in DYMO. The only requirement is that it has a value of at least one.

Optional Discarding of Messages In DYMO, messages might be discarded at a node's will. The authors do not focus on possible reasons, but enable the discarding nonetheless. A possible reason might be e.g. load balancing.

Link Breakage In AODV link breaks are only detected when a node tries to forward a message over a broken link, which leads to the loss of a data message in most cases. In DYMO all nodes are required to monitor their links continuously. The mechanisms used for this monitoring are not described by the authors, but examples of possible ways, including a neighborhood discovery protocol, are hinted at.

Complexity While AODV had 23 configuration parameters, the complexity has been reduced in DYMO to only 10. But this advantage has to be put in perspective, because AODV takes care of mechanisms like e.g. blacklisting as part of the protocol, and needs parameters for this to function correctly. In DYMO the usage of blacklisting is only advertised, so the parameters are in fact there, only hidden.

For this thesis, one fact is most important: The authors state right at the beginning of the RFC that DYMO "`only utilizes bidirectional links`" [10]. This statement is strengthened by applying the keyword "MUST", which means that this is absolutely

mandatory for the operation of the protocol according to [6]. If there is a chance that unidirectional links occur, they have to be suppressed by blacklisting or other appropriate means, because otherwise persistent packet loss may occur. Still, DYMO is included here because it is the current and upcoming protocol of a family of distance vector protocols for MANETS which includes DSDV (section 3.2.1), AODV (section 3.2.2) and DYMO, all proposed at least partially by the same author(s).

3.2.9 DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks

Dynamic Source Routing (DSR)[28, 29, 30] consists of two basic parts, route discovery and route maintenance, both of which are only executed on demand. Route discovery is used to find a route from a node S to node D only when node S wants to transmit a message and does not know any valid path. Route maintenance is executed when the transmission of a message from node S to node D failed for any reason.

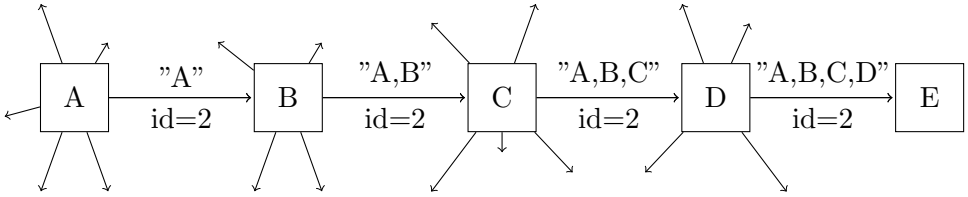


Figure 3.11: Route Discovery in DSR (taken from [28])

Figure 3.11 shows an example of the route discovery process. Node A wants to find a route to node E and transmits a route request message (RREQ). The route request contains the identity of the sender and a request id which is used for duplicate detection. All nodes that receive and forward the RREQ add their own identity to the message, resulting in a complete path that has been collected when node E is reached (A=>B=>C=>D=>E).

When the destination receives the RREQ, it sends a route reply (RREP) to the initiator, containing the accumulated list of intermediate nodes. Of course, duplicate suppression is used first: If a node receives a RREQ with a message ID it has already seen or if its node ID is already in the list of nodes the message has passed, it discards the message (loop prevention).

To transmit a RREP to the initiator of a RREQ, the destination first checks if it already knows a route to that node. If it does not, it initiates another route discovery, but this time the source route that has already been found gets attached to the

RREQ message. Otherwise, both nodes would continuously alternate in starting route discovery, never receiving a route but continuously increasing network traffic.

When DSR is used on top of MAC protocols like IEEE 802.11 which require bidirectional links on the MAC layer, this second route discovery process can be omitted. Instead, the destination simply reverses the collected source route contained in the RREQ, and sends the RREP along this bidirectional path.

Messages that could not be sent because route discovery for their destination is underway are kept in a so-called send buffer. Unanswered route requests are resent using an exponential backup mechanism; additional messages with the same destination do not cause additional route requests to be transmitted.

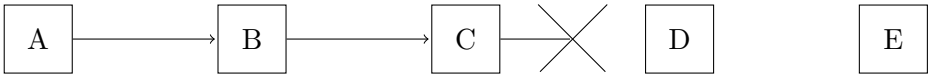


Figure 3.12: Route Maintenance in DSR (taken from [28])

Route maintenance has to be started when messages are lost. Each node is responsible for the next hop of a message in DSR, e.g. in figure 3.12 node C is responsible for the hop to node D. A node can be informed of a successful reception by the next node either through link-layer acknowledgments or through passive acknowledgments (overhearing the forwarding by the next node). If none of these options are available, DSR provides a flag that can be set in the header of a DSR packet, which causes the recipient to transmit an explicit acknowledgment.

When a node does not receive any confirmation that the message has reached its next hop, it retransmits the message up to a configurable number of times. When no confirmation is received even after these retries, a route error message (RERR) is sent back to the originator of the undeliverable packet, including the information about the broken link. The originator then removes the route containing the broken link from its route cache.

Please note that the original message has not been stored by the originator and is not transmitted back from the node which detected the broken link. Instead, from the point of view of DSR, the message is dropped. A retransmission is left to higher layers, e.g. a TCP layer. If such a higher layer tries to retransmit the message, other cached routes for the same destination are used if existing. Otherwise, a new route discovery process is started.

Both mechanisms, route discovery and route maintenance, have only been described in their basic form here. There are a number of additional features for both:

- Caching overheard routing information
- Replying to route requests using cached routes
- Preventing route reply storms when using cached routes
- Limiting route request hop count
- Packet salvaging
- Automatic route shortening
- Spreading of route error messages
- Caching of negative information

Caching of overheard routes enables a node that has been part of a communication or that has overheard a message while it was in promiscuous mode to enter a route into its route cache without ever having started a route discovery. In networks where links are fairly stable and bidirectional, this approach saves a lot of communication overhead, because the probability that a node has to start the costly route discovery process is reduced drastically. In networks with often occurring unidirectional links however, there is a high chance that the cached route is unusable or broken (see figure 3.13 for an example). Due to the unidirectional links in the network, the path from node A to node E and the path from node E to node A are different. In the figure, only the path from A to E is shown. If one of the intermediate nodes would cache a route back to node A, this route would never work. Moreover, if node C would overhear node X forwarding a message from node V addressed at node Z, it would assume that it could reach Z through X, which is wrong due to the unidirectional link between X and C.

As unidirectional links are the main focus of this work, and the authors of [28] claim that unidirectional links limit the usefulness of route caching, it is not used. Likewise, the possibility of using cached routes to answer a route request can also be useful, but relies on link stability. In networks like WSN where links change often, these so-called intermediate replies often produce stale routes and are therefore not used in this work. As they are not used, the route reply storm problem cannot occur and the mechanisms used to prevent it as described in [28] are not needed. The limitation of the distance a route request may travel can be used to realize an expanding ring search. RREQs are initially sent with a low TTL-value. If no reply is received within a certain time,

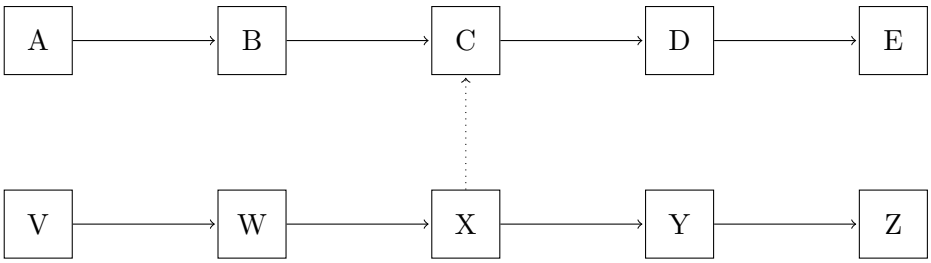


Figure 3.13: Possible Problems of Overheard Routes in DSR (taken from [28])

the TTL-value is increased for the next try. As the authors stated that they expect the network diameter (i.e. the longest possible route) to be no more than 10, the usefulness of this mechanism in large sensor networks is questionable since it possibly adds delay and can even increase the amount of messages flooded if multiple ring searches are needed.

Packet salvaging can be used when a link break is detected. The node that detected the break may search its own cache for a different route to the destination and forward the message along that path instead of dropping it when an entry is found. The packet then needs to be marked as already salvaged to avoid creating loops if the new path is broken as well. Automatic route shortening is used when a node overhears a message of which it is not the intended next hop, but featured in the future path of the message. This node then sends a gratuitous route reply to the original sender of the message, informing it of the shorter route for future use.

Route Error Messages are normally only sent to the originator of a message that could not be transmitted. If intermediate replies are enabled, it is useful to spread the knowledge about the broken link further. When the originator starts a new route discovery, it includes the route error message in its route request, thereby informing all nodes of the link break and preventing them from answering with stale information containing that broken link.

Negative information, like the information that a link is broken, can be cached, too. It is useful when a link breaks and reappears frequently. If a node knows that the link used by a certain message has a tendency to break, future route replies that feature this link may be ignored, similar to the blacklisting mechanism for unidirectional links used in AODV (see section 3.2.2).

DSR also offers support for heterogeneous networks and mobile IP as well as multicast routing. Both features are irrelevant to this thesis and are therefore not discussed here.

3.2.10 Distributed Cache Updating for DSR

The author of [83] addresses the problem of stale routes in route caches. Route caches or route tables are used to store routes that have been discovered or overheard. Stale routes are routes which are still in a route cache even though they can no longer be used due to link breakage. DSR (see section 3.2.9) and many similar protocols use timeouts to remove stale routes after a certain time. The problem with this mechanism is that sometimes routes that are still viable get removed if the timeout is set too short. If it is too long, stale routes may stay in the cache and will be used again eventually.

The second mechanism used in DSR is that of issuing route error (RERR) messages. When a node tries to forward a message and does not receive an (passive or explicit) acknowledgment that the next node has received said message, a number of retries are made. If none of these succeed, a route error message is sent to the originator of the discarded message as part of the route maintenance process.

This route error message is the focus of the cache updating protocol presented in [83]. In DSR, there are two modes of operation. In the first mode, only the originator of the discarded message is informed of the link break, regardless of the fact that other nodes may have cached paths that use this link, too. If nodes operate in promiscuous mode (i.e. listen to messages not addressed to themselves) all nodes that overhear the transmission of the RERR message also remove routes that contain the failed link from their caches.

The proposed distributed cache updating algorithm enhances the route error spreading to reach all nodes that have a route in their cache which uses the broken link. The node that finds the link break informs all its neighbors that use this link which in turn inform all their neighbors that use this link and so on. While this sounds like a flooding of the network, it is indeed different from that. Instead, a so-called cache table is introduced, which holds the information needed to identify neighbors which cached the broken link as part of a route.

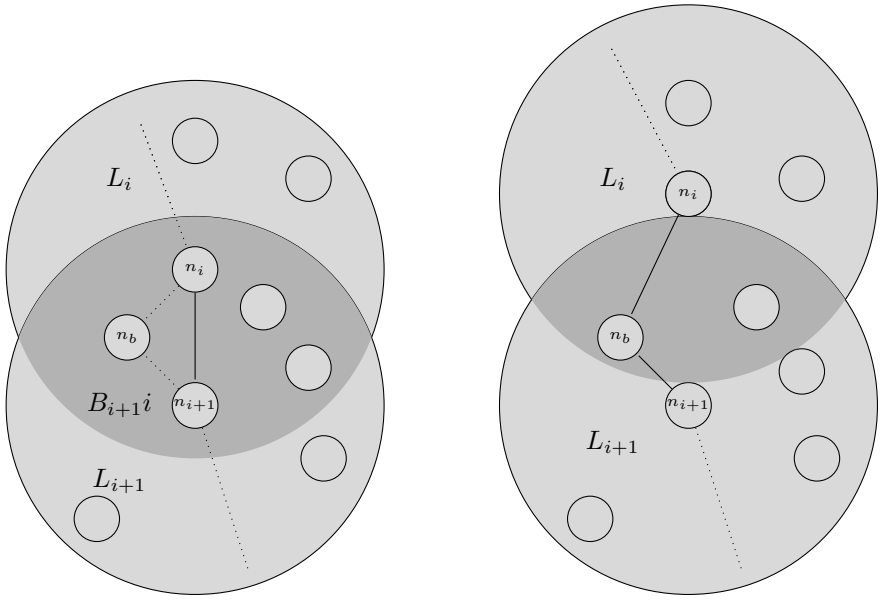
The cache table contains two different types of information. The first part informs about the caching. If it has been used only in upstream nodes, in both upstream and downstream nodes or not at all. The second information is the identity of each neighbor that has learned links through route replies. The data contained in the cache table can be collected without additional cost, as it can be deduced from the route replies and data messages a node forwards. They contain the source route.

The author claims that the distributed cache updating outperforms DSR by 19 % in standard mode and by 7 % in promiscuous mode. However, gratuitous route error messages were not used in the standard mode of DSR during the evaluation, which

would have increased its performance. Also, the size of the cache table was simulated as infinitely large, which would not be possible on real hardware. Then, a replacement strategy would have to be found, as the cache table would be constantly full in networks with a high load. This would in turn lead to nodes not being notified of updates and a decrease of performance for the distributed cache updating protocol.

3.2.11 A DSR Extension for Connection Stability Assessment in Mobile Ad-Hoc Networks

The protocol presented in [54] is a quality of service extension that has been implemented for dynamic source routing (see section 3.2.9). The author presents a mechanism to assess connection stability, where a connection can be comprised of multiple different routes that are used consecutively. This represents the view of the application, which does not care about how stable a single link or even a whole path is. Rather, it is concerned whether the packets arrive at all, or within certain time boundaries.



(a) n_b bridging node for link (n_i, n_{i+1})

(b) n_b bridges interrupted link

Figure 3.14: Bridging of broken links (taken from [54])

The stability is quantified by using what the author calls `Bridging Node Density` (BND). It can be intuitively visualized for a single link as the number of common

neighbors of the two nodes that are connected by that link. In figure 3.14 the BND for the link between node n_i and n_{i+1} has a value of 3, as there are three other nodes within the intersection of the transmission ranges of both nodes.

This value is summed up for each link on a path and divided by the number of hops to form the value for a path p. To take into account the fact that longer routes tend to break more often, and the observation made that this probability seems to grow linearly, the BND is weighted with the reciprocal of the path length:

$$wBND = \frac{1}{(p-1)^2} \sum_{i=1}^{p-1} B_{i,i+1}$$

A higher value of $wBND$ means that the end-to-end connection ought to be more stable than one with a low value. The absolute value of the BND heuristic is, however, not used to predict the lifetime of a connection as some other heuristics are. Rather, it is only used to compare connections, when different communication partners are available for the same service. Then, the one with the higher value, i.e. probably the highest lifespan, is chosen.

3.2.12 LBSR: Routing Protocol for MANETs with Unidirectional Links

The Loop Based Source Routing protocol (LBSR) [3] is based on DSR [28] and designed to use unidirectional links in the routing process. DSR is able to route using unidirectional links, too, but the method is not very efficient as the destination floods an enlarged route request. This represents much overhead as the packet flooded by the destination does not only pick up the identities of the intermediate nodes but also contains the route from the source to the destination (see 3.2.9).

LBSR eliminates the need to flood the network from a message's destination. Instead of building a route from the source S to the destination D and back, routing loops are created. A so-called Lreq message is flooded into the network by S. Every node that receives this Lreq for the first time rebroadcasts it after attaching its Identity (ID). If an Lreq is received by node S, a loop has been found. S now knows a route to each node whose ID is enclosed in the Lreq. Now it transmits (unicast) a packet along that route which enables all nodes along this route to send packets to each other and to S by following the enclosed route. If a node that is already part of a loop receives a message from a node that is not its predecessor in the loop, it adds its ID to the Lreq and forwards it along its loop.

The evaluation of LBSR focuses on the number of floodings and messages transmitted. The number of floodings is naturally only half as large, but the total number of sent messages is higher. The fact that a loop has to be passed by two messages costs time, energy and bandwidth. The broadcast character of wireless networks, i.e. that unicast messages are received by all nodes within range, even those not addressed by the message, has been ignored completely.

The main advantage pointed out by the authors is that the number of entries in the route cache is many times as large as in DSR. If that is an advantage remains debatable, though. The protocol creates as many loops as are possible. The increased number of cache entries means that there are more invalid cache lines if a node moves or a timeout occurs. Also, the cache size needed is much bigger. Finally, the protocol uses a source routing approach, which means that the packets used to create longer loops can become arbitrarily large. If the loop encompasses the whole network, its headers must hold the identities of all nodes in the network. This enormous overhead definitely disqualifies a use of this protocol in wireless sensor networks.

3.2.13 Directed Diffusion for Wireless Sensor Networking

Directed Diffusion [26, 27] uses a data-centric approach. There are four types of elements in Directed Diffusion: interests, data messages, gradients and reinforcements. Requests from users are transformed into so-called interests. Nodes that fulfill certain criteria included in the interest messages activate their sensors and transmit their results back along the reverse path the interests have taken. This path reversal of course once again means that bidirectional links are needed. Nodes that forward multiple results may aggregate them, e.g. to pinpoint the location of an event more accurately. Directed Diffusion uses named data packets to describe tasks. These are usually attribute-value pairs. When a node receives an interest, it searches its cache for a corresponding entry. If no entry exists, a new one is created. This entry includes the information contained in the interest, i.e. the desired data rate, time stamp and duration. A gradient is also created, which refers to the node the interest was received from. To realize this, locally unique node identifiers are needed.

If an entry already exists but no gradient to this neighbor can be found, the gradient with its corresponding time stamp and duration is added. If a gradient already exists, only the time stamp and duration are updated. There can be any number of gradients associated with one interest. A node X that finished processing an interest rebroadcasts it. All neighbors that receive the interest create a gradient referring to X. Note that all gradients work only locally, i.e. only the next hop is stored. Intermediate nodes

have no knowledge of the originator of the interest. After a certain time, every node has a gradient to each of its neighbors, when the interest has been flooded through the network.

Nodes that have received interests and fulfill the criteria therein unicast data messages to all neighbors from which an interest was received. These can be identified through the gradients that are stored in the interest cache. Nodes that receive a data message check if they have a corresponding interest in their cache. If they do not, the message is discarded. Otherwise the data cache is checked, and if this results in a match, the message is discarded (loop prevention). If there is no match in the data cache, an entry is created and the message is forwarded along the gradients in the interest cache.

At first, the network is flooded by interests as well as by data messages, because every node sends all messages to all of its neighbors (figure 3.15). Eventually, as many copies of the data message as it has neighbors would reach the sink. This is of course not what is wanted. Therefore, Directed Diffusion allows the sending of reinforcements. These reinforcements are transmitted along a preferred route. A sink that receives the first data message may choose to reinforce the path through the node that forwarded this message, because it is seemingly the fastest path. Other criteria like e.g. the highest data rate can also be applied. An important fact is that all decisions about reinforcement are made locally. No extra communication between nodes is necessary, and no node knows more than its local neighbors. Nodes do not even need to know the identities of source and destination.

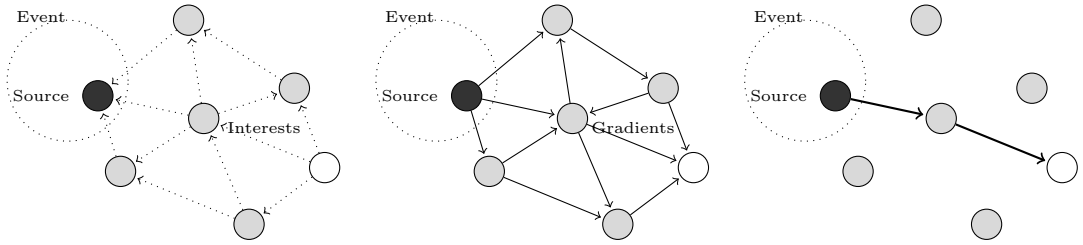


Figure 3.15: Setup of Gradients (taken from [27])

Intermediate nodes may also start reinforcements, to enable local repair. When a node on a previously reinforced path realizes that it does not receive data messages at the necessary rate any more, it uses reinforcement rules to find another empirically good path.

It is possible, that local rules result in multiple paths between source(s) and sink(s). Therefore, a way to truncate unwanted paths is also needed. An easy way to do this is to associate a timer with every gradient. If no reinforcement is received for a certain time, the gradient will simply fade. Another way is the usage of so-called negative reinforcements. If the sink has two incoming gradients, it sends a negative reinforcement to one of the corresponding neighbors, e.g. the one whose messages constantly arrive after the ones from the other. When a node receives a negative reinforcement, it degrades the corresponding gradient. If there are no other gradients left, it transmits a negative reinforcement to the node from which it received its data messages.

This protocol is one of the most cited protocols for wireless sensor networks. The fact that it is data-centric and does not require globally unique identities makes it very popular. There are, however, a few problems. Locally unique identities are still needed. This means that within two hops of a node X, no other node may use the same identifier as X. In wireless sensor networks, where neighborhoods are determined by radio range and radio range changes every so often, realizing an identity finding protocol that fulfills the necessary criteria only with local communication is hard. More problematic is the fact, that identities have to stay locally unique, even if the nodes are mobile. In most cases, this can only be guaranteed if identities are globally unique, too.

The example application sketch uses geographic information to specify the nodes of interest. If geographic information and locally unique identities are available, it is fairly simple to turn these two into globally unique identifiers that are based on the region and refined with the local identity.

The diffusion approach has a very simple way of dealing with unidirectional links: They are ignored. If an interest is propagated over a unidirectional link from node X to Y, then Y can never send back a data message along this link. X does not receive a message from Y and does not reinforce the gradient of Y. In fact, it does not even know that Y had a gradient that pointed at X. As no reinforcements arrive, the gradient will be removed after a certain timeout or any other aging mechanism.

This removal of unidirectional links is once again the major drawback of the algorithm. Directed Diffusion does not guarantee optimal paths, and indeed there may be better ones using unidirectional links.

3.2.14 Scatterweb - Low power Sensor Nodes and Energy Aware Routing

Schiller et al. describe the design of the ScatterWeb ESB nodes and the ScatterFlasher as well as an energy aware routing scheme they denote as `Solar Aware Routing` [67]. The routing protocol is an extension of directed diffusion (see section 3.2.13), and is used to differentiate between two kinds of nodes in a heterogeneous network: Energy-scavenging nodes and purely battery powered nodes.

The basic assumption behind Solar Aware Routing is that nodes which have means to generate their own power (e.g. from solar cells, vibration or temperature differences) should be preferred when routing decisions are made, in order to conserve the batteries of all other nodes, thus enhancing overall network lifetime.

The authors specify (among some other minor changes) two new fields for the interest and data messages called `Bcount` and `Scount`. When a node forwards a message, it increases one of these counters; B(attery)count for battery operated sensor nodes and S(cavenging)count for nodes that use environmental energy.

When a sink decides to reinforce a path, a new weight function has become available. While directed diffusion normally uses shortest path or highest link quality weight functions, now the path with less battery operated nodes can be chosen. If ties occur, an additional weight function, e.g. the shortest path among those tied, is applied.

As Solar Aware Routing is built upon directed diffusion, the propagation of interests and reinforcements also relies on bidirectional links.

3.2.15 Matching Data Dissemination Algorithms to Application Requirements

The main problem addressed in [24] is the choice of routing protocol in data centric networks. As the authors show, the wrong choice of dissemination protocol and its interaction with the application can introduce up to 60% overhead. But typical application developers are not system experts, and need guidance in choosing the right protocol.

The second contribution made in the paper is an introduction of two new diffusion algorithms: Push and one-phase pull diffusion. The original version of directed diffusion is labeled as two-phase pull in this paper, because it involves two floodings of the network; Flooding of interest messages to find sources and flooding of exploratory data messages to reinforce gradients.

Two-phase pull works well in scenarios where only a few sinks are present, in many applications there is just one gateway node. But for a different scenario, where communication between sensor nodes is necessary (e.g. activation messages), the number of gradients and flooded messages increases drastically. Push diffusion was designed with such a scenario in mind, where many sources and many sinks exist, but data is generated only seldom. In Push diffusion, sinks do not flood interest messages. Instead, communication is started by the sources, by flooding exploratory data messages, thus removing one of the floodings from two-phase pull.

One-phase pull works in the exact opposite way. Sinks flood their interest messages into the network (pull). Here, the second part of two-phase pull, where the sources flood exploratory data, is removed. Instead, the data is sent along one gradient directly, by answering the first message that arrived and ignoring all others. This choice is based on the fact that the message that arrives first must haven taken the fastest path, and the assumption that the other direction will be equally fast.

This assumption is made in many other protocols, and leads to the same problems for one-phase push. The authors also state that the problem of asymmetry is known, but they expect the medium access layer to identify asymmetric links.

Even though asymmetric links are mentioned, unidirectional links, as a special case of asymmetric ones, and their implication on the proposed protocols are not discussed in the paper. However, it can be deduced from the description of the protocols and their relation with directed diffusion (two-phase pull) that unidirectional links would reduce the performance drastically. While two-phase pull can ignore the implications of unidirectional links, messages sent using one of the two new diffusion protocols would not reach either the sink or the source respectively, if there was a unidirectional link on the path the fastest interest/exploratory data message uses.

3.2.16 Rumor Routing Algorithm for Sensor Networks

Rumor Routing [7] is another data-centric routing algorithm. When an event occurs, nodes that detect the phenomenon generate an agent with a certain probability. An agent is a long lived packet which travels through the network in a random way. It builds a path to the event, and may pick up paths to more events as it travels. Figure 3.16 shows an example of this.

The black and grey clouds represent events 1 and 2 respectively. An agent has been sent by a node which detected event 1 (not shown) and has passed all black nodes, which now know a route to the event. A second agent has been generated by a node which detected event 2. The beginning of its path is shown as the grey nodes. Once the agent

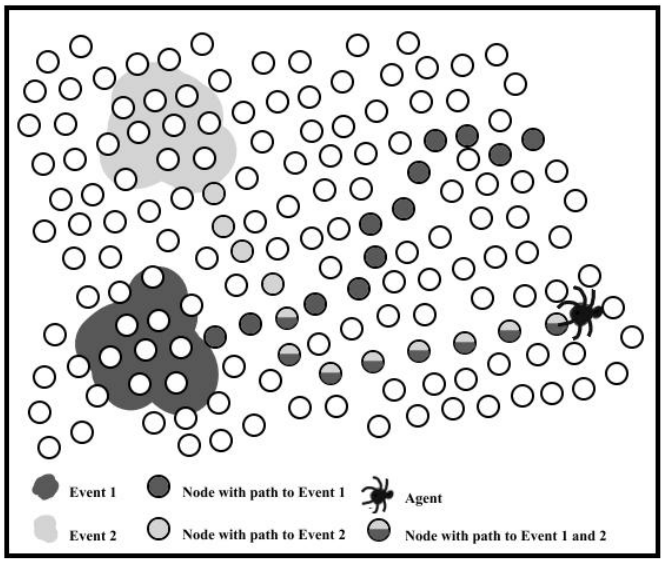


Figure 3.16: Agent propagation in rumor routing (taken from [7])

arrives at a node which has knowledge of the route to event 1, it starts propagating routes to both events. Nodes that know both routes are shown half black half grey.

As it is expected that more than one node detects an event, a probabilistic approach is used to determine whether an agent should be created or not. If more than one agent is created, their paths may overlap or collide. This is used to reduce the length of a path when an agent that knows a shorter path arrives at a node. Figure 3.17 shows a case in which a second agent reduces the path length drastically.

When an agent reaches a node that is interested in the event, that node sends an interest back along the path the agent has taken. This usage of the reversed path makes Rumor Routing unusable in the presence of unidirectional links.

3.2.17 Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks

Busch et al. analyze the performance of the full and the partial reversal algorithm [8]. These algorithms build a directed acyclic graph (DAG) that leads from any node to a fixed destination. If there are multiple destinations, one graph has to be built for each of them. When a node has to transmit a message, it simply sends it on any of its outgoing links for that destination. This way, the used route is not always the shortest, but the message will reach the destination eventually. When links break, nodes that

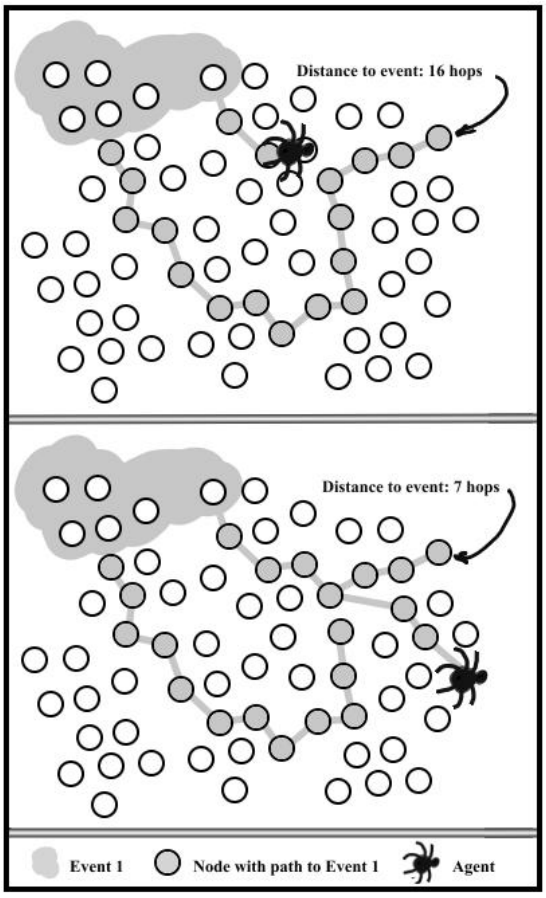


Figure 3.17: Path length reduction in rumor routing (taken from [7])

have no more outgoing links reverse all of their incoming ones (full reversal) or only a certain subset of them (partial reversal). This may cause further reversals in neighboring nodes. The results of the analysis show that the full reversal algorithm is asymptotically optimal, while the partial one is not.

While link reversal routing may be efficient for n-to-1 communication, the fact that a separate DAG is needed for each destination means that link reversal will not be used in this thesis. Realizing any-to-any routing with link reversal algorithms would result in as many DAGs in each node as there are nodes in the network. This represents too much memory consumption, computational overhead and communication cost for both wireless sensor networks and Mobile Ad-Hoc Networks. Moreover, the route reversals require all links to be physically bidirectional, even though they are used logically unidirectional.

3.2.18 Temporally-Ordered Routing Algorithm

Even though the Temporally Ordered Routing Algorithm (TORA) [53] uses a directed acyclic graph (DAG) to describe routes in the network, it still relies on bidirectional links. These are needed because TORA is based on the link reversal principle.

The focus of TORA is the reduction of the impact of link failure (e.g. due to node mobility) on routing performance. The authors state that TORA will not find the shortest path to a destination in most cases, but it will find one and find it fast, if one exists. They also state that shortest path routing protocols produce a high overhead when a link failure has to be handled, because failure notifications sometimes need to be propagated through the whole network.

To prevent this overhead, no routes through the network are stored. Instead, a node only has a certain number of incoming and outgoing links, with respect to a sink. This means that a node has multiple different routes to choose from at any given time. If an outgoing link breaks, no action has to be taken as long as there is still a route to the destination (the detection of link breakage is not part of TORA, it is left to the link layer).

When the last outgoing link breaks, the node becomes a global maximum (in terms of distance to the destination). It then has to propagate its new height level to its neighbors (not described in TORA) and reverses its known links. This reversal may cause one or more of its neighbors to loose their last outgoing link and so on. The height levels are propagated through the network, and used to detect network partitions.

Figure 3.18 shows how the failure of the last outgoing or `downstream` link on a Node i is handled by TORA. When a link failure occurred, the node simply needs to propagate a new reference level. Otherwise, if its neighbors have different reference levels, it takes the highest reference level found among its neighbors and propagates that. If all of the node's neighbors do have the same level, the reflection bit comes into play. This bit is used to show that the current level of a node has not been set by itself, but adapted from another node ("reflected"). This is necessary to prevent false positives in the partition detection algorithm. Otherwise, two nodes that have chosen the same level independently would assume a partition and start erasing routes. This reflection is used to represent a so-called sub-level. If the bit is set, the node checks whether it has been the one to define that level. If it has been, it starts the erasure process, otherwise a new reference level is generated.

There are two problems with TORA. First, it can only use bidirectional links as do all link reversal algorithms. Second, it needs to define a separate DAG for each destination in the network. This would qualify it for a scenario where all messages are transmitted

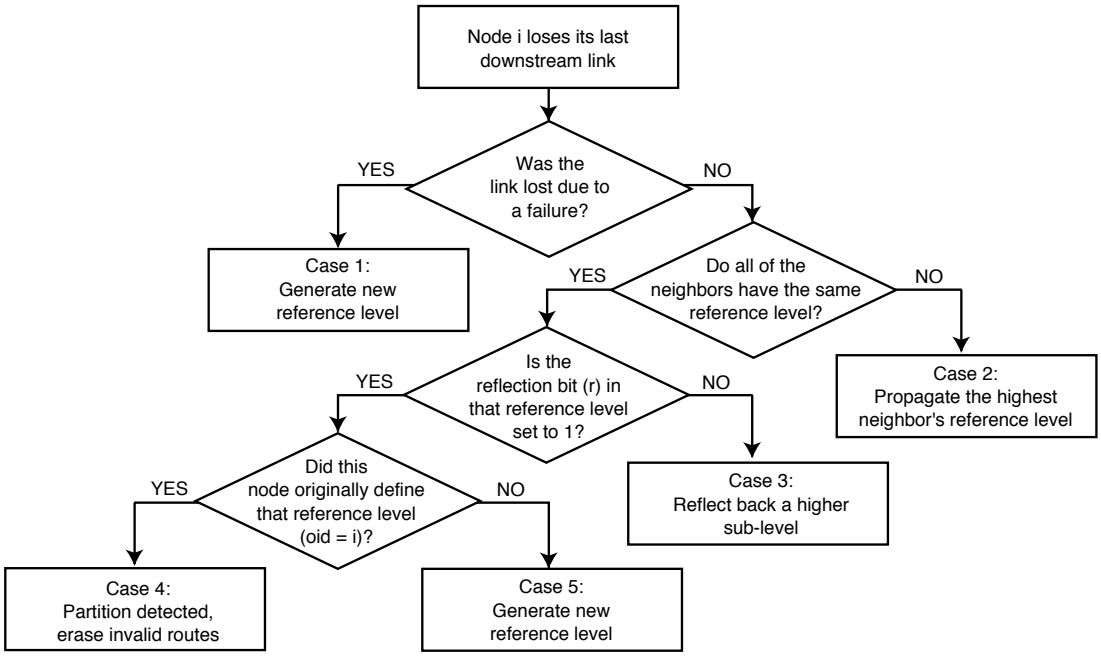


Figure 3.18: The Decision Tree for Route Maintenance used in TORA (taken from [53])

in a N-to-1 pattern, e.g. a sense-and-send application in a sensor network. But for an any-to-any communication pattern, the overhead in terms of memory consumption and protocol messages is much too high as each node would need to store as many DAGs as there are nodes in the network. Also, the logical reversal of links requires all links to be (physically) bidirectional.

3.2.19 GeoTORA

GeoTORA [32] is an extension of TORA (see section 3.2.18) that uses geographical information. More specifically, GeoTORA does not route messages to a single node, but to all nodes within a certain region (geographic multicast, also-called geocast).

The directed acyclic graph (DAG) used in TORA as routing structure for a single sink is shown in figure 3.19. It shows how a link break between nodes D and F occurs (a), and the incoming links of node D are reversed as a reaction to this breakage of the last outgoing link by D (b). As a result of this reversal, node C has no more outgoing links and reverses its incoming link from node B (c) which in turn switches its link from node A (d). Now a fully functional route exists once more.

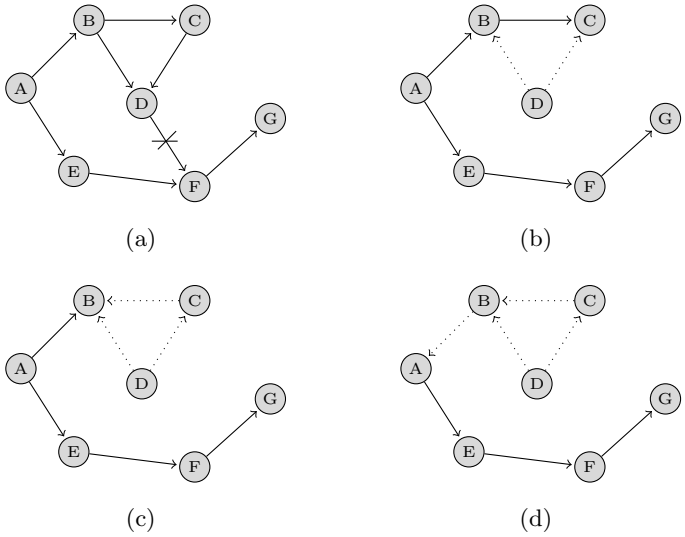


Figure 3.19: Route Maintenance by Link Reversal used in TORA, as depicted in GeoTORA [32]

GeoTORA modifies the single destination approach of TORA to enable anycast. In anycast, the destination of a message is not a single node, but any one member within a certain group of nodes. To enable this anycast, the sink concept of TORA is enlarged. Where only the single destination of a message had no outgoing links in TORA, all members of an anycast group follow this scheme in GeoTORA. Moreover, there are direction-less (logically as well as physically bidirectional) links between sinks of the same anycast group.

Figure 3.20 shows an example, where nodes A, B, C and D are in the same anycast group (a). Even though the shown links are logically unidirectional, the physical links are assumed to be bidirectional. All nodes that are not members of the group use a DAG from TORA to reach any of the members. If one of the directed links break, e.g. the link between nodes G and A (b), links outside the anycast group are reversed following TORAs methodology until the DAG is repaired (c).

GeoTORA is not an anycast protocol, as its proposed enhancements only define the first step towards anycast. Messages are assumed to be directed at a certain area for which the anycast group has been defined. Therefore, a node that is within the group floods messages in the geographic area upon reception. All non-members that receive such a flooded message discard it, all members rebroadcast it. Sequence numbers are used to implement duplicate suppression.

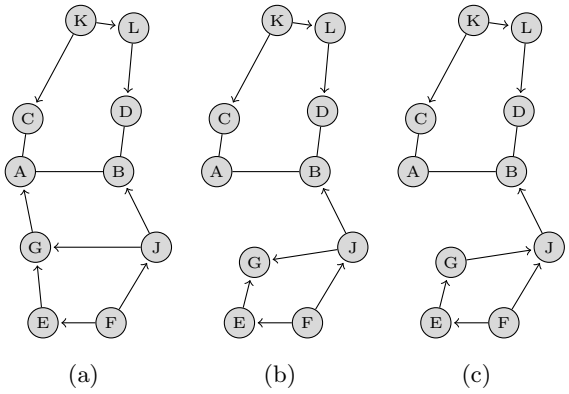


Figure 3.20: Local Repair of the DAG in GeoTORA (taken from [32])

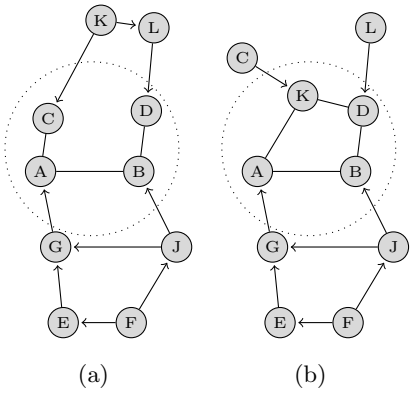


Figure 3.21: Membership Changes in a Geocast Group (taken from [32])

Figure 3.21 shows the impact of node mobility on GeoTORA. At first, nodes A, B, C and D are members of the group because they are within the geocast region. Now, node C leaves the specified area while node K enters. As node C is no longer a member of the group, it needs the DAG to forward messages, while node K is now connected to all members via bidirectional links.

GeoTORA removes the need to have a separate DAG for each possible destination, i.e. each node, from TORA as nodes are grouped in geocast regions. Also, it enables the usage of an area as destination rather than individual nodes. Still, even though the number of DAGs is reduced, GeoTORA is not usable for any-to-any routing and cannot operate on unidirectional links.

3.2.20 Greedy Perimeter Stateless Routing

Being a geographical routing protocol, Greedy Perimeter Stateless Routing (GPSR) [31] requires that each node knows its own position and, if it is the originator of a message, the position of the destination in order to make forwarding decisions. Also, each node needs to know the positions of all of its neighbors, which are transmitted as beacons at regular intervals.

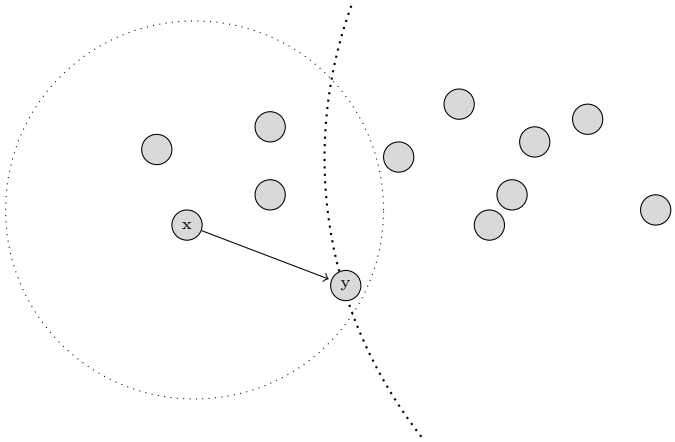


Figure 3.22: Greedy Forwarding in GPSR (taken from [31])

The basic principle of GPSR is greedy forwarding, as shown in figure 3.22. When a node X wants to transmit a message, it checks its neighbor table to find all neighboring nodes that are (geographically) closer to the destination than itself. From this set of nodes, the neighbor with the shortest distance to the destination (node Y on figure 3.22) is chosen and the message is forwarded to this node. This process is repeated by each node upon reception of the message until the message reaches the destination.

Greedy techniques suffer from one problem, though. It is possible that the message is received by a node that has no neighbors that are closer to the destination than itself for various reasons (e.g. a communication obstacle). GPSR solves this problem by using `perimeter forwarding`, a forwarding principle based on the right-hand rule for graph traversal. The message is routed along the perimeter of what the authors call the `void`. This method is only feasible for planar graphs, and suffers from crossing edges in the communication graph.

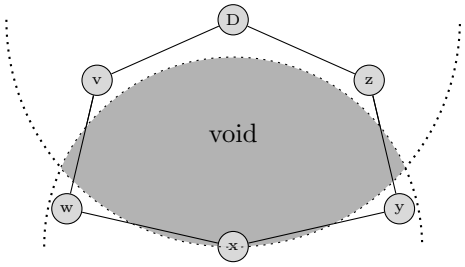


Figure 3.23: Perimeter Forwarding in GPSR (taken from [31])

Figure 3.23 shows the void of node X with respect to destination D. It consists of the intersection between two circles. The transmission range of X and the distance between X and D around D. There can be no nodes in this area, otherwise node X would have used them as forwarding node in the greedy fashion.

GPSR has a high delivery ratio if the number of obstacles is small. If it is high, the perimeter forwarding is needed more often, which degrades performance. GPSR's memory consumption is also low, as nodes only need to remember their direct neighbors instead of whole tables of destinations and the routes to them.

One of its drawbacks with respect to this thesis is that it can only use bidirectional links, as a node has no knowledge of neighbors it can reach through unidirectional links. Worse still, it will even know the position of a node that has a unidirectional link pointing to it, and try to forward messages to this neighbor which are never received.

3.2.21 Geographical and Energy Aware Routing

Geographical and Energy Aware Routing (GEAR) [84] is a data centric, location based routing protocol that was developed especially for wireless sensor networks. Its basic principle is similar to that of GPSR (see section 3.2.20), as it uses a greedy forwarding mechanism. The difference is that GEAR does not use the distance between a node and the destination alone as weight function, but rather also includes the energy consumption of candidate nodes to decide which one should forward a message. This is realized using the so-called `learned cost` of a neighbor. It is defined as a weighted sum of the distance and the energy consumption:

$$\alpha * distance + (1 - \alpha) * energy\ consumption$$

The choice of α has a strong influence on the chosen routes. If it is set to 1 or all neighbors have the same energy value, only the distance is evaluated, resulting in a purely greedy forwarding. If it is set to 0 or all neighbors are equidistant, the decision degenerates to a load splitting algorithm.

Another difference between GPSR and GEAR is the handling of voids or holes as they are called in GEAR. When a node realizes that none of its neighbors is closer to the destination, it forwards the message to one of its neighbors but also increases its own cost for the destination. As these values are propagated (infrequently), its neighbors come to know of the higher cost eventually and may select a different route.

The third difference has its origin in the data centric property of GEAR. While GPSR addresses nodes by their exact location, GEAR assumes that messages are destined for a certain region, i.e. a square area. It forwards the messages to the centroid of that square, which means that there must not be a node at that exact location. Rather, mechanisms are needed and provided to stop a message from traveling around this area "forever" (i.e. until its time to live expires).

As the learned costs of routes is propagated using local broadcast, nodes with incoming unidirectional links may calculate wrong forwarding information, depending on the used weight function. The probability of a wrong forwarding decision rises with the percentage of unidirectional links and the frequency of link changes/breaks.

3.2.22 SPEED

SPEED [20] is a geographical routing protocol for real-time applications. It offers three different kinds of delivery service: Regular unicast, area-multicast and area-anycast. It has been designed with wireless sensor networks in mind and provides a per-hop delay guarantee to meet the real-time requirements of an application.

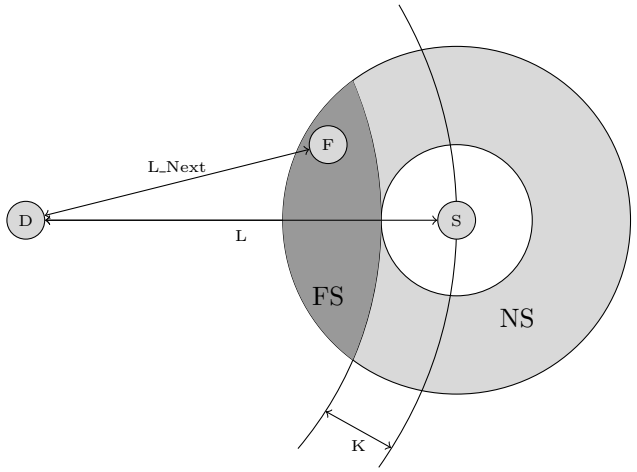


Figure 3.24: Selection of Forwarding Node in Speed (taken from [20])

Figure 3.24 shows the forwarding principle used in SPEED. Node S wants to transmit a message to node D. The white circle around S represents a geographical distance of K units. All neighbors within this area are ignored as they would not bring the message much further to the destination. K is a parameter of SPEED and set before deployment. The larger circle around S represents its communication range, every node that is within this circle but not within K distance of node S is a member of the Neighbor Set (`NS`) of node S.

L is the distance between nodes S and D. All nodes in the NS of node S that are further away than L from node D are ignored. As the message should be at least a distance of K closer to the destination after the next hop, all nodes whose distance is higher than L-k are also removed from the candidate set. The resulting set of nodes is

called the Forwarding Candidate Set (`FS`). The distance between one of those candidate nodes and the destination is shown as `L_next` in the figure.

The real-time characteristics of SPEED are realized in the next step. Nodes in the FS are divided according to the delay they induce on the communication. The absolute distance gained by one hop (`L - L_next`) is divided by this delay (`SendToDelay`). Nodes that achieve a higher score in this ranking have a higher probability of getting chosen as next hop. As each hop shortens the distance of the message to the destination by at least K, the maximum number of hops needed is L/K plus one for the starter node. In combination with the single-hop delay guarantee, this property is meant to provide an upper boundary for message delivery time.

One of the major drawbacks of SPEED is that is does not provide any mechanism for routing around a communication obstacle (`hole` or `void`). Instead, the authors argue that a formula can be found that defines the necessary network density to guarantee a void free network. Also, if the deadline for single hop delivery cannot be met, the message is dropped. This might be a mistake, if one of the later hops could have compensated the delay.

SPEED relies on a so-called `Neighbor Beacon Exchange` to compute the per hop delays. Every node broadcasts its ID, position and `receive delay` periodically. Nodes that receive such a beacon message store the data in their neighbor table and use it in their forwarding decisions. As neighbor beacons are only transmitted using local broadcast, unidirectional links lead to wrong information in the neighbor table and thus wrong forwarding decisions.

3.3 Own publications referenced in this chapter

- Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. An adaptive TDMA based MAC protocol for mobile wireless sensor networks, best paper award. In International Conference on Sensor Technologies and Applications, 2007. [41]
- Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. MLMAC - an adaptive TDMA MAC protocol for mobile wireless sensor networks. In Ad-Hoc & Sensor Wireless Networks: An International Journal, Vol.8 Nr.1-2, 2009. [43]
- Stephan Mank, Reinhardt Karnapke and Jörg Nolte. MLMAC-UL and ECTS-MAC - Two MAC Protocols for Wireless Sensor Networks with Unidirectional Links, best paper award, Third International Conference on Sensor Technologies and Applications Athens, Greece, 2009. [44]
- Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. Mac protocols for wireless sensor networks: Tackling the problem of unidirectional links. In International Journal on Advances in Networks and Services, vol 2 no 4, pages 218 - 229, 2009. [42]

Chapter 4

New Routing Protocols

As chapter 3 has shown, existing protocols for wireless sensor networks or Mobile Ad-Hoc Networks still lack the ability to handle unidirectional links in an efficient manner. Many of those protocols deal with unidirectional links by removing their negative impact on the routing tables, while some of them use unidirectional links explicitly. Making these unidirectional links usable introduces overhead, which has to be weighted against the gain. The authors of [45] have evaluated some of the existing protocols and concluded that the gain is not worth the cost. While this might have been true for their scenario and the protocols they evaluated, it is possible to have scenarios where network separation occurs when unidirectional links are eliminated. It is also possible to have protocols that induce far less overhead than the ones they considered.

The authors of [48] are primarily concerned with energy consumption in wireless networks. Even though energy consumption is not explicitly addressed in this thesis, there is one fact from the paper that should be mentioned here: Multihop communication does not save energy. One basic assumption made in some protocols is that it is cheaper to have a message travel two hops to a destination instead of one because the transmission power needed for a certain distance d scales with d^n where n is at least 2. Thus, the energy consumed by two hops of 1/2 d would be much smaller. While this is true when only the energy needed to bridge the distance is measured, real hardware has a high power consumption that is induced by the transmitter and receiver electronics (α) which is independent of the transmission power. As shown in the paper, the consumption is rather $\alpha + \beta d^n$, with α as the dominating part. Therefore, introducing an additional hop does not conserve energy, quite on the contrary. For this reason, all nodes are assumed to always transmit on the highest power level, and shorter paths are preferred in the context of this thesis.

Table 4.1: The five protocols developed in this work

Name	Type	diameter	density	optimization	requirement
Buckshot	SR	small	medium	code size	none
BuckshotDV	DV	any	medium	packet size	none
OSBRDV	DV	any	medium - high	#packets	none
ULTR	DV	medium - high	medium - high	cross-layer	neighbor info
ULC	DV	medium - high	medium - high	info gain	neighbor info

To solve the problems described in chapter 2 without introducing too much overhead, new routing protocols are needed. Because of the different application scenarios, there are a lot of variations in the requirements for the routing layer. Designing a routing protocol that delivers the best results under all of these circumstances is simply impossible. Therefore, five new routing protocols are presented in this chapter. All of these protocols use flooding at least for route discovery, making a duplicate detection mechanism necessary. The suggested method is to use the identity of the source of a message and a growing sequence number as used e.g. in AODV (see section 3.2.2).

A very brief overview of the protocols is given in table 4.1: It shows the name of the protocols, their type (Source Routing or Distance Vector), the intended network diameter and node density as well as the optimizations that distinguish them from one another. Please note that all of these protocols have of course been designed to make use of unidirectional links, and the corresponding column has been removed as all entries would have been the same. However, two of the designed protocols, ULC and ULTR, require neighborhood information that can be supplied either by a neighborhood discovery protocol or by a MAC layer (see below).

Four of the described protocols assume a minimum hop routing character, meaning that for each hop taken a counter is incremented by one. While some argue that minimum-hop routing protocols are likely to have lots of packet losses and therefore less throughput (e.g. [12]), the proposed protocols have been designed to be resolute against such losses. Furthermore, the weight function can easily be replaced by a different one without changing the routing protocol. Examples for different weight functions include delay, residual energy or load balancing mechanisms. All of these can be integrated by replacing the described increments with different values. For the routing protocols, the meaning of these values is irrelevant, only the difference is used to find the shortest (or cheapest) path.

Buckshot Routing (section 4.1) has deliberately been kept simple, yet still achieves a good delivery ratio. It has been designed to require the least amount of memory on the used sensor nodes.

BuckshotDV (section 4.2) is a distance vector version of Buckshot Routing. It has been designed for networks with a large diameter, in which the messages of Buckshot Routing would get too large because of its source routing character.

OSBRDV (section 4.3) was designed to reduce the number of packets that are transmitted, making it usable in low-bandwidth scenarios.

ULTR (section 4.4) has been designed to use cross-layer information provided by a TDMA MAC protocol, namely the information about the two-hop neighborhood of a node.

The fifth protocol, ULC, enables the usage of specialized weight functions by collecting information about the links traversed by a route request message. The information about the links' status is then provided to the destination node, which can make different decisions according to the chosen weight function. Also, the collected information can be made available to the application, easing network monitoring and enabling adaptive applications (see section 4.5).

4.1 The Buckshot Routing Protocol

Buckshot Routing [58] is based on the source routing principle also used in e.g. DSR [28]. When a node A wants to send a message to a node B it looks up the path in its routing table. If there is an entry, the whole route is attached to the message and the message is transmitted. If no entry is found, a route discovery is started. The big difference to DSR lies in the way messages are forwarded.

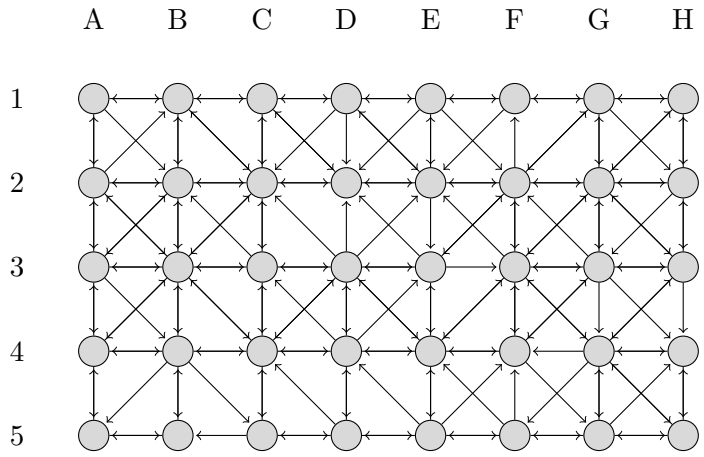


Figure 4.1: A Communication Graph containing Unidirectional Links

Figure 4.1 shows a possible communication graph with unidirectional links for a small sensor network consisting of 40 nodes. Parts of this communication graph will be used throughout this chapter to illustrate the workings of Buckshot Routing.

In Buckshot Routing, a node that wants to find a route to another node floods a route request (RREQ) message into the network. Every node that receives such a RREQ checks if it is the destination of this RREQ. If it is not, it appends its ID to the route request before retransmission. Please note that the RREQ naturally also travels over unidirectional links. Figure 4.2 shows an example from the communication graph depicted above, where node A3 searches for a path to H3. The route most likely taken by a flooded route request message includes a unidirectional link. Also, it only needs seven hops which is the minimum number of hops required in this example network to get from one side to the other. Consequently, RREQ messages arriving later at H3 that have taken a path with the same or a higher length will be ignored.

When a RREQ message reaches its destination, the whole path that was taken by the RREQ is enclosed and can be stored in the routing table in reverse order (table 4.2), just like in DSR when bidirectional links are used. After storing the reversed route, the

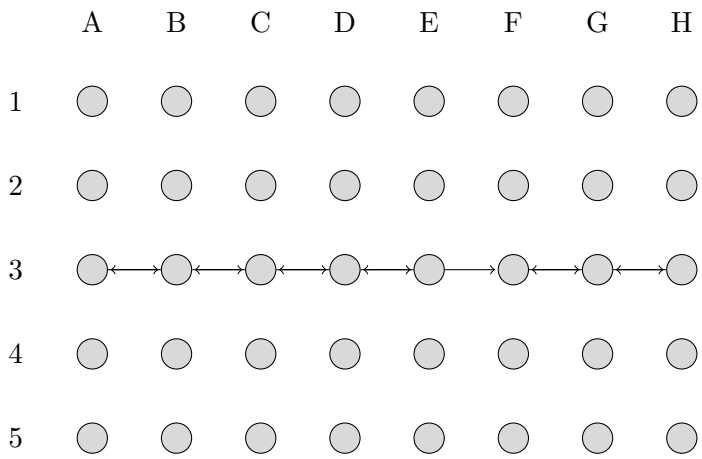


Figure 4.2: Path taken by the fastest RREQ Message

Table 4.2: Routing Table of Node H3

Destination	Path
A3	{G3, F3, E3, D3, C3, B3, A3 }

destination transmits a route reply (RREP) message. Now, the neighborhood tables get involved.

In Buckshot Routing all nodes also remember their neighboring nodes in a neighborhood table. This table is maintained without additional communication overhead, simply by listening to the medium and recording the last hop of all received messages. When a node receives a message (RREP or data) that is not addressed to itself, it parses the message header to identify the intended next but one hop. If that node can be found in the current node's neighborhood table, the message is forwarded.

Figure 4.3 shows all links used by RREP messages that reach their intended destination. The unidirectional link between nodes E3 and F3 that has been used by the RREQ message can of course not be used in the reverse direction. But it is passed by on two different sub-paths of length two: One containing the node with the incoming unidirectional link and another which bypasses that node completely. Both sub-paths are created by nodes that are not on the original path, but have the next-but-one hop in their neighbor table. This is called *pseudobidirectional links*. Bidirectional links are used as always, unidirectional ones are used in one direction and passed by on the way back if there is a one-hop detour available.

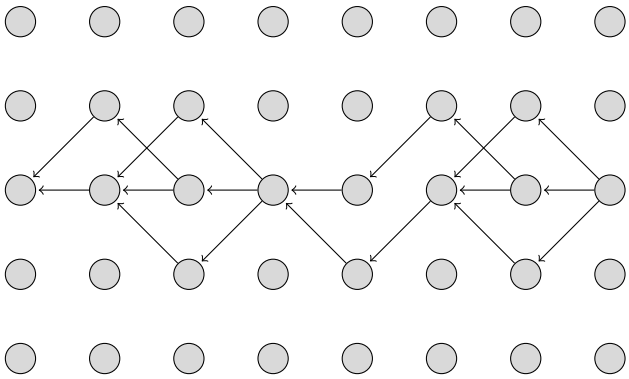


Figure 4.3: Multiple Paths taken by a RREP Message

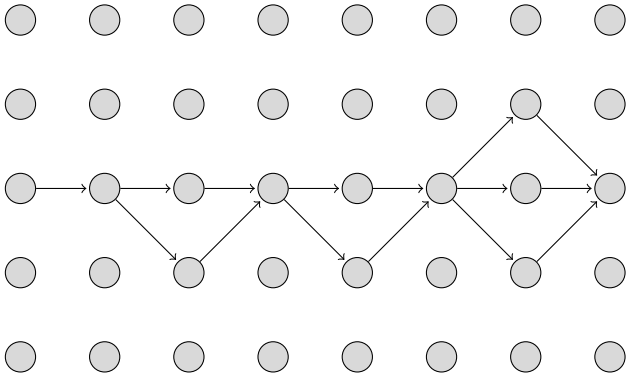


Figure 4.4: Multiple Paths taken by a Data Message

Figure 4.4 shows the paths taken by the Data packet. As the forwarding mechanism for data packets is the same as for RREP messages, the data packet is also forwarded around obstacles. Please note that, as the data packet should take the same path as the answered RREQ message, it could theoretically be forwarded directly, without using the obstacle evading mechanism. However, this would only work if the logical topology of the network was stable, with no link changes. As the experiments described in chapter 2 have shown, this is often not the case. Therefore, the obstacle evasion is also used for data packets, even though it increases the network load. What is more, the mechanism increases the robustness not only against unidirectional links, but for all cases of packet loss. The presented mechanism also enables Buckshot Routing to operate without retransmissions, which are typically used in other protocols.

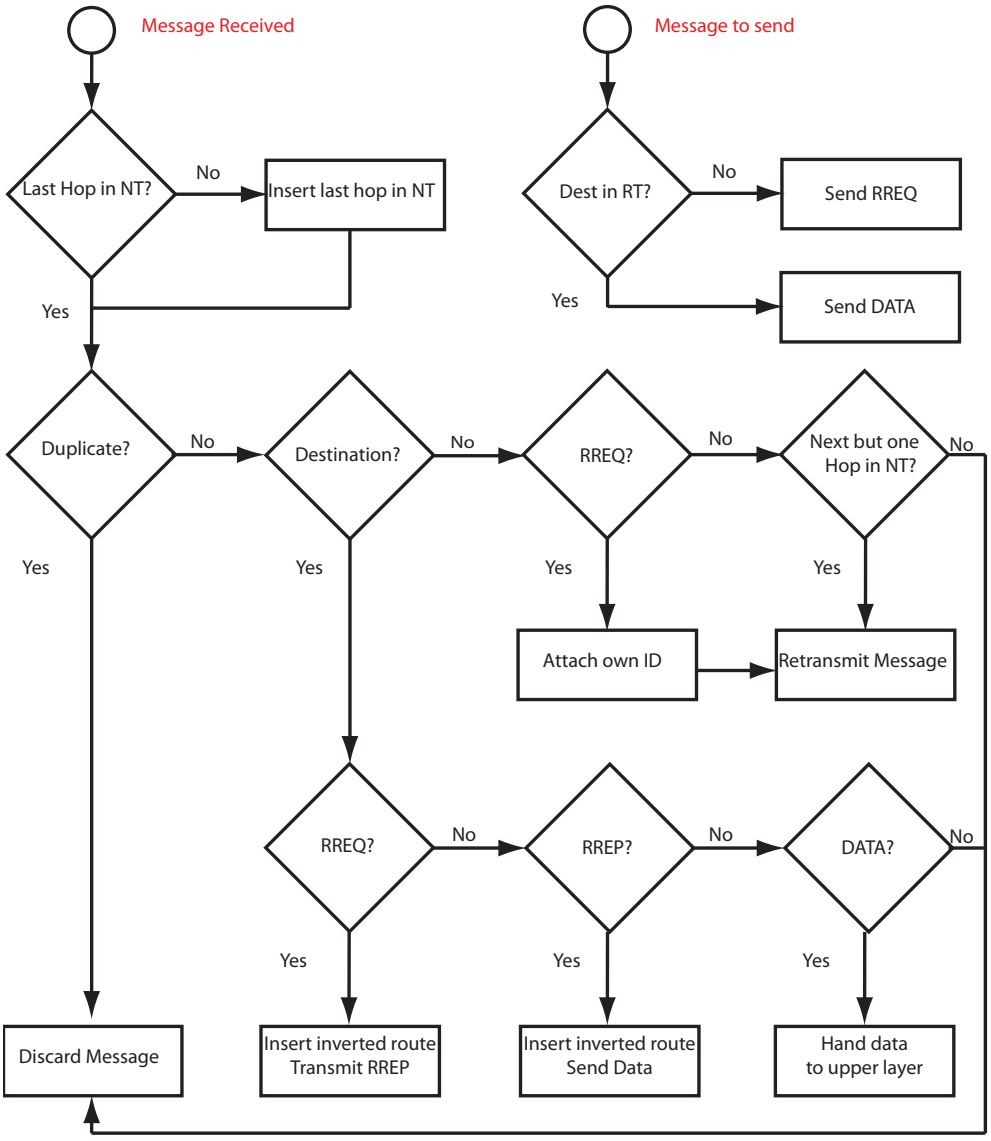


Figure 4.5: Message Handling in Buckshot Routing

The way Buckshot Routing reacts when a message is handed to it from the lower (*Message Received*) or the upper layer (*Message to send*) is depicted in figure 4.5. When a message needs to be delivered to another node, the algorithm checks for a route to that node in its routing table. If one is found, a DATA message is sent. Otherwise, a RREQ message is flooded.

When a node receives a message, it updates its neighbor table by inserting the ID of the last hop if that was not already contained. After that, duplicate detection takes place. If the same message was already received from another neighbor (and thus handled before), it is silently discarded. If it was no duplicate, the node checks whether it is the destination of this packet. If it is, the message type is determined and the message handled accordingly:

The reversed route from a Route Request is entered into the routing table as route to the source of the RREQ, then a RREP message is sent along this reversed path. The path taken from a RREP message is used to transmit a DATA packet along the newly found route after it is inserted into the routing table. Data from a Data packet is simply handed to the upper layer. If this node is not the destination, it checks if the message is a RREQ. If it is, the node appends its own identity to the message before broadcasting it again. If it is a RREP or DATA message, the path it should take has already been determined previously. Therefore, the node checks if it has the next-but-one hop (the one after the next) in its neighbor table. If it has, it retransmits the message after increasing the pointer to the next hop by one.

4.1.1 Message Types

Buckshot Routing uses only the three types of messages described above: RREQ, RREP and DATA. The message format of each is shown in the following.

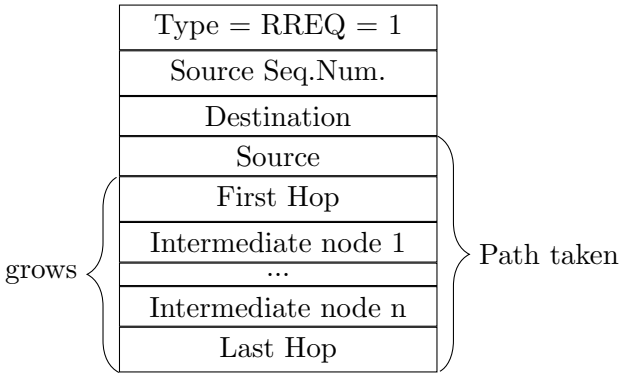


Figure 4.6: RREQ Packet Format in Buckshot Routing

Figure 4.6 shows the structure of a RREQ message. It contains the type information and the identity of the destination which are necessary for the ongoing route discovery. The source sequence number in combination with its identity are used for duplicate suppression. These four entries are all set by the source of the RREQ when it starts the

route discovery. They also define the initial size of the RREQ packet. Once it has been transmitted to the neighbors of the source, it starts to grow. As each node that receives the RREQ appends its own ID, the RREQ grows by the size of one ID per hop. When the RREQ reaches the destination, its size equals $n*sizeof(ID)+sizeof(initialRREQ)$ where n is the path length from source to destination.

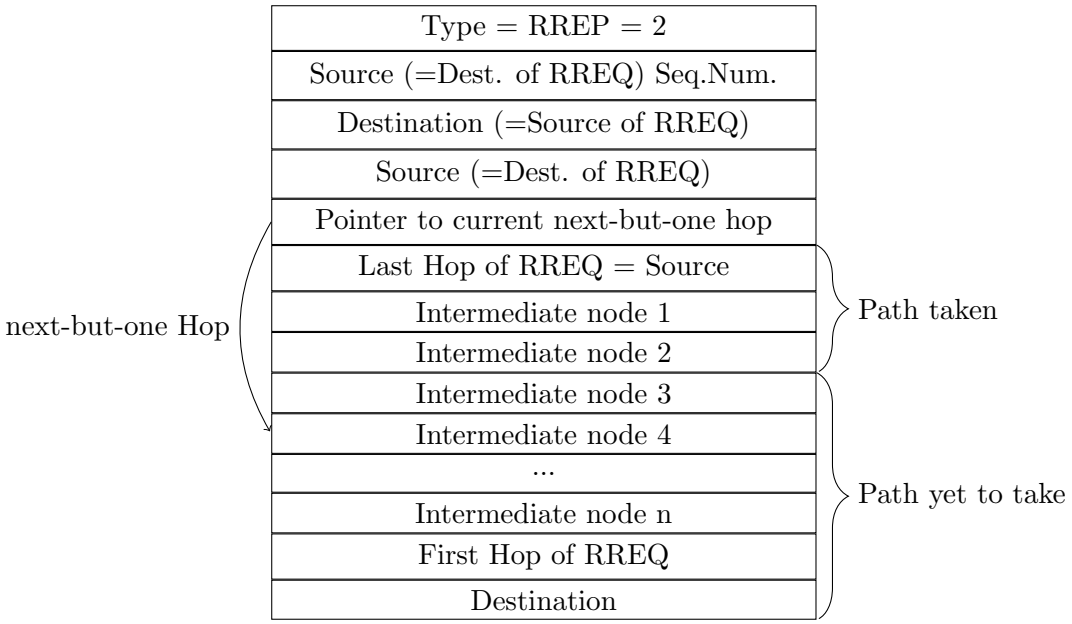


Figure 4.7: RREP Packet Format in Buckshot Routing

The format of a RREP message in Buckshot Routing is shown in figure 4.7. It consists of the type information, followed by the ID and sequence number of the node that generated the RREP message, i.e. the destination of the corresponding RREQ. This information is once again used for duplicate suppression. After this, the whole path that has been taken by the RREQ is included, just as it is in the RREQ itself. The only addition made is a pointer to the current next-but-one hop, which is needed for the forwarding algorithm and which increases with each hop.

Figure 4.8 shows the format of a DATA packet as used in Buckshot Routing. It contains the type of message followed by the sequence number and the identity of the node that generated this message. Then, the necessary routing information is enclosed: The identities of all nodes on the path from the source to the destination and, the pointer to the next-but-one hop which is once again used in the forwarding mechanism. The application data can be found at the end of the message for alignment reasons.

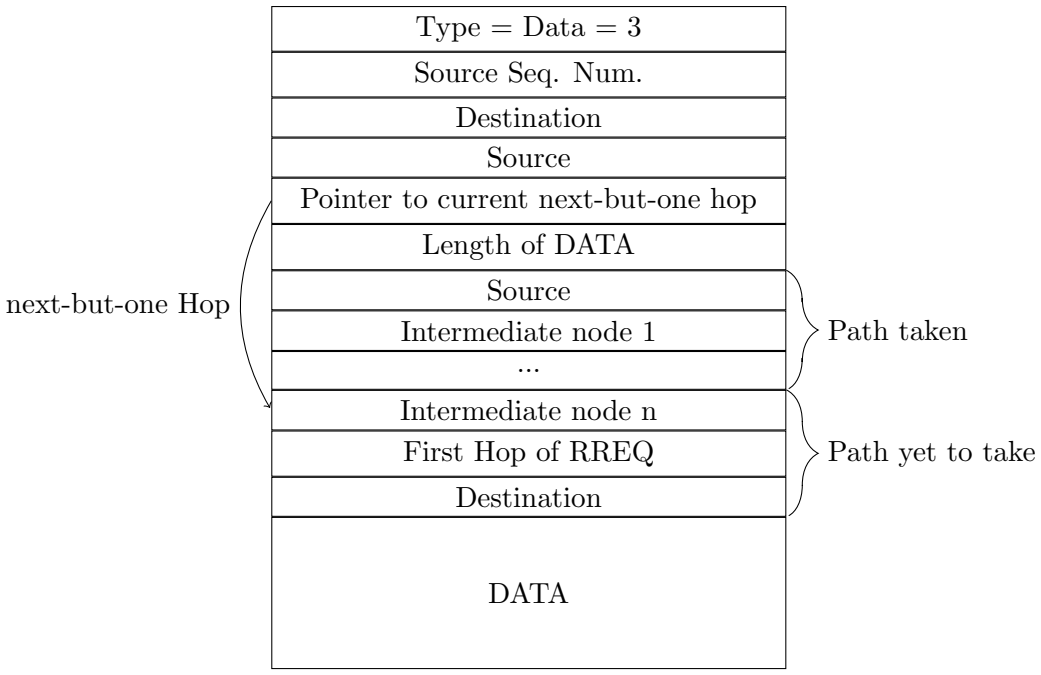


Figure 4.8: DATA Packet Format in Buckshot Routing

4.1.2 Variations

There are three interesting possibilities of improvement to the operation of Buckshot Routing. The first one is to use route shortening, the second one is header shrinking and the third one are highly dynamic routes.

When a node receives a RREP or DATA message and it is not the intended destination, it checks if it has the next-but-one hop in its neighbor table. If it has, it retransmits the message. This mechanism could be easily changed to check not only for the next-but-one hop, but rather for any other node that the packet should pass in the future. This way, the message reaches the destination much faster. By removing all hops in between from the packet header, it would even be possible to inform the destination of the shortened route. This would reduce the network load produced by future packets.

Another possible extension which is also used in some other source routing protocols is based on the fact that DATA messages contain the path from source to destination, but it is only used on the path. Once the destination is reached, the path is irrelevant. This is already true for the path up to any intermediate node. Once it is reached, the path taken to it is not used anymore. This leads to the conclusion that the packet

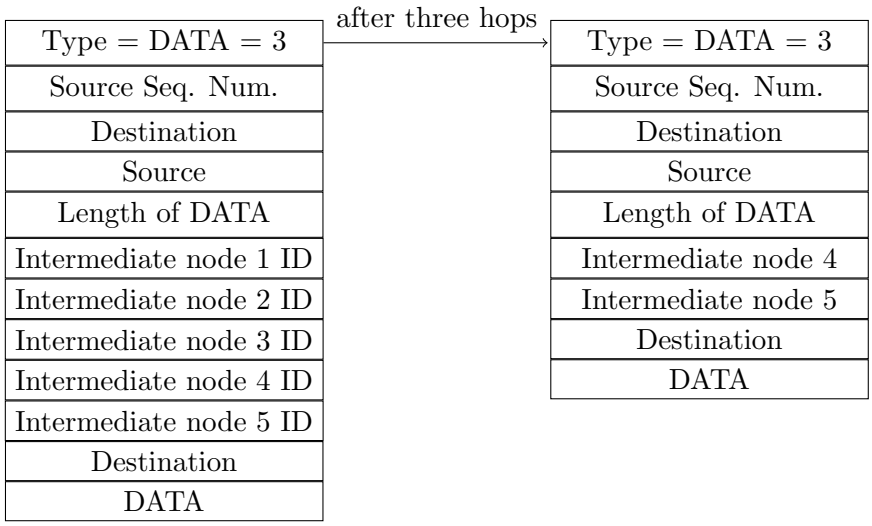


Figure 4.9: Hop-by-Hop Reduction of DATA Packet Size

header can be shortened with each hop the packet takes. As a result of that, there is also no need for a pointer to the next hop anymore, because only nodes that have not been passed already remain mentioned in the header. Figure 4.9 shows an example where the initial path consists of six nodes. Every time the message passes a node, the current hop information is removed, shrinking the message by the size of one identity. A major drawback introduced by using this extension is that it disables passive learning of routes and cache updates.

When following the basic principle of Buckshot Routing, the actual route that is taken by the data packets differs, depending on the link status. The route description enclosed in the messages (RREP, DATA) is always the one collected in the RREQ or its reversed equivalent, though. This opens up the possibility to use highly dynamic routes: Instead of always using the same path, nodes that forward a RREP or DATA message may replace the ID of the next hop as shown in the message header by their own before forwarding the message to the next-but-one hop. This way, messages always contain the route that has actually been taken, and the destination can replace the path previously in its routing table (if any) with the new one. In this way, routes are checked and repaired each time a message is transmitted. This is especially useful for highly dynamic networks (mobile nodes) or when the communication is assumed to involve a lot of messages transmitted in both directions over a longer period of time.

One problem remains with these extensions, though: They cannot all be used at the same time, as e.g. shortened routes would disable the highly dynamic ones.

4.1.3 Advantages and Disadvantages

Buckshot Routing in its basic form was designed with simplicity in mind. Many other protocols that enable the usage of unidirectional links are complicated and introduce a large overhead. The computational overhead and memory consumption of Buckshot Routing are small. It is expected to provide a good delivery ratio even with an increasing number of unidirectional links (see chapter 5 for actual numbers), without having the need to flood the whole network twice as e.g. DSR does. Moreover, Buckshot Routing is able to deal with changes in the topology much better, because an exact route is never needed. In many cases where other protocols need to repair routes and handle errors, buckshot still works fine, because it is inherently fault tolerant [58].

As described in the variations, route shortening is also possible. On the downside, the packet headers are large due to its source routing nature. This could somewhat be diminished with header shortening, but is still a problem, which will be addressed in the next section.

Buckshot Routing is similar to AODV-BR (see section 3.2.3) as both protocols use a one-hop detour. But AODV-BR needs to detect message loss first, and inquire neighboring nodes for alternate routes afterward, while Buckshot Routing uses detours implicitly, removing the communication overhead for a recovery. Also, the neighboring nodes are inquired for a path to the destination in AODV-BR. In Buckshot Routing they only need to know the next-but-one hop, due to its source routing character. AODV-BR is based on distance vectors and does not have the information about the next-but-one hop available in each node. Moreover, Buckshot Routing can circumvent dead nodes as well as broken links, because the messages are not forwarded to the next hop, but to the next-but-one hop. Finally, Buckshot Routing can already use this forwarding mechanism for RREP messages, enabling the usage of unidirectional links on the path.

4.2 Buckshot Routing with Distance Vectors

Buckshot Routing is expected to simplify the routing process and still have a good delivery ratio. One problem that remains is the network load, as Buckshot Routing is based on a limited directional flooding, which means that more packets than absolutely necessary are transmitted. As these transmissions are needed to provide the robustness against unidirectional links, node failures and message losses, the protocol described in this section will focus on reducing the packet size, thus reducing network load without losing robustness. To realize this, the source routing based character of Buckshot Routing will be replaced with a distance vector version.

Table 4.3: Routing Tables in BuckshotDV

Destination	Next_but_one Hop	Hop Count
D	B	3

In traditional distance vector routing algorithms like DSDV (see section 3.2.1) or AODV (see section 3.2.2), each node maintains a routing table, with entries consisting at least of the ID of the destination, the distance, and the next hop. Using the same entries in Buckshot Routing with Distance Vectors (BuckshotDV) is simply not possible. As described in section 4.1, Buckshot Routing needs to know the next-but-one hop, which means that this value has to be kept in the routing table, too. Table 4.3 shows an example of a routing table for BuckshotDV. Instead of a whole path with many node identities, it contains a single ID: The next_but_one hop. But, this has to be determined somehow, requiring changes to the way route request (RREQ) messages are built in distance vector protocols.

4.2.1 Message Types

In BuckshotDV a node enters its own ID along with the ID of the node from which it received a RREQ message before retransmitting it, previous entries are overwritten. A node that receives a RREQ message now knows its neighbor's neighbor, and thus the next-but-one hop on the reversed path, which it enters into its routing table in the form (Source of RREQ, next-but-one hop, distance). This entry is based on the fact that in Buckshot Routing the "real" next hop is never important, only the next-but-one hop. The only exception to this is the source/destination, which does not have a next-but-one hop. To compensate this the source enters an illegal ID when creating a RREQ message.

The first value in the RREQ is the type of message, followed by the sequence number of the originating node and its identity, which are used for duplicate suppression and to build the reversed route. The destination ID is of course necessary to terminate the route discovery once the destination has been reached. All of these values are fixed throughout the lifetime of a RREQ message.

The first value being subject to change is the hop count which is incremented by one on each hop. Please note that of course any other weight function like, e.g. energy, would also be possible. The hop count is followed by the identities of the previous and the current hop, which of course change with each hop the message takes.

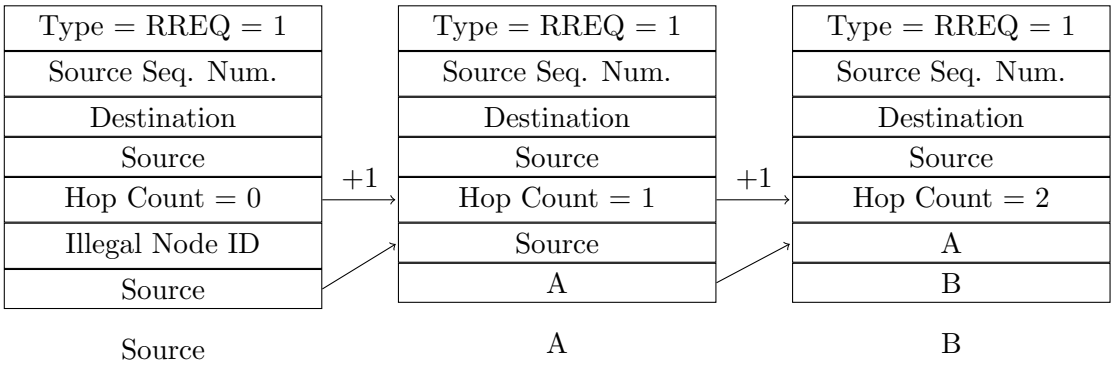


Figure 4.10: Per Hop Changes in a Route Request Message in BuckshotDV

Figure 4.10 shows an example of a RREQ message that is transmitted from its source to node A and then to node B. The changing values are initialized with 0 for the hop count, an illegal value and the source's ID before the source transmits it RREQ message. Upon reception of this message, node A enters the source with a distance of 1 and next-but-one hop: the illegal value into its routing table. This is necessary to prevent all other neighbors from rebroadcasting a message from A to the destination over and over again. After creating the routing table entry, node A increments the hop count of the RREQ message and enters the last hop (the source) and its own ID before retransmission. On node B the procedure is the same. If some node C received the message from B it would create a routing entry consisting of the source, a distance of 3 and node A as the next-but-one hop.

When a node receives a RREQ and determines that it is the destination of this packet, it creates a routing entry for the source of the RREQ message and transmits a route reply (RREP). RREP messages contain the ID of the node from which the RREP was received and the identity of the next-but-one hop in BuckshotDV. The next-but-

one hop is needed to find the route to the source of the RREQ message, the identity of the previous node is needed to build the backward route. Thus, contrary to Buckshot Routing in its basic source routing variant, RREP messages are also used to built new routes. Nodes that receive a RREP message check their neighbor table for the next-but-one hop listed there, which is the next hop from their perspective. If and only if there is an entry, they look up the next-but-one hop from their perspective in their routing table, adjust the values in the RREP message and retransmit it.

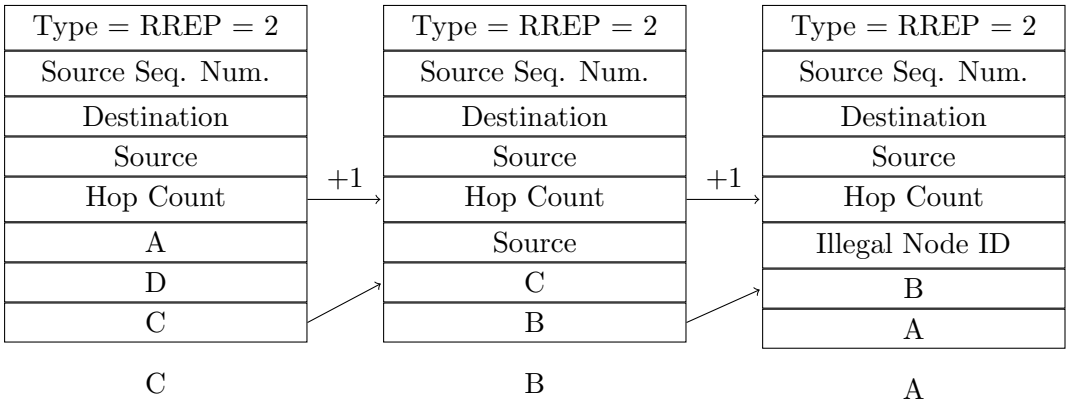


Figure 4.11: Per Hop Changes in a Route Reply Message in BuckshotDV

In Figure 4.11 an example of the way RREP messages are handled in BuckshotDV is given. The RREP message consists of four values that do not change and four that do. The type, sequence number, source ID and destination ID are used in exactly the same way as before. In the varying fields, the hop count has been reset by the destination of the RREQ before retransmission and now denotes the distance of each node that receives the RREP message from the destination of the RREQ. Following the hop count, the identity of the next-but-one hop is inserted, which is used on the receiving nodes to decide whether they should forward the message, following Buckshot Routing's forwarding mechanism. The other two varying fields are the same as in the RREQ message: The identity of the last and current hop. They are used to build routing table entries for the way to the destination (of the RREQ message). In the example, node A enters an illegal value into the next-but-one hop field, because it is a direct neighbor of the destination and no next-but-one hop exists. Please note that node A does not know that, i.e. it retrieves this value from its routing table. No special case handling is required, because the illegal value had been present as next-but-one hop in the RREQ due to which this routing entry was made.

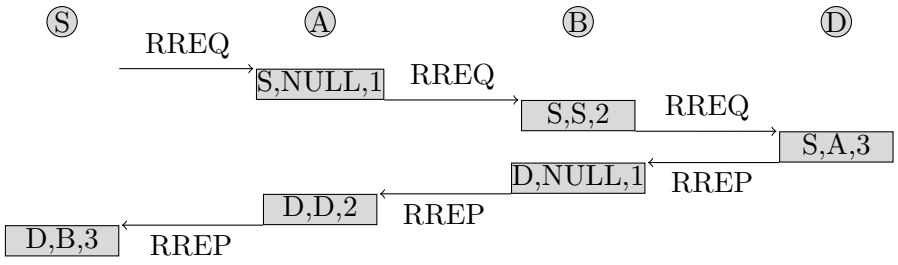


Figure 4.12: Routing Table Entries Generated by BuckshotDV

An example of the way routing table entries are created in BuckshotDV can be seen in figure 4.12. Node S, the source, searches for a route to node D, the destination. It transmits a route request message as described above. Upon reception of this message, node A enters node S into its routing table with no next-but-one hop (illegal value: NULL) and a distance of 1. When node B receives the (modified) RREQ, it enters node S with next-but-one hop node S and a distance of 2 into its routing table. Finally, node D receives the RREQ, creating an entry consisting of node A as next-but-one hop and a distance of 3 for node S. This concludes the building of the backward route. Now the forward route has to be established by the route reply message, which is transmitted by node D. Node B receives it and enters node D with no next-but-one hop and a distance of 1 into its neighbor table. For node A the entry consists of node D as next-but-one hop and a distance of 2. Node S enters node B as next-but-one hop and a distance of 3 into its routing table.

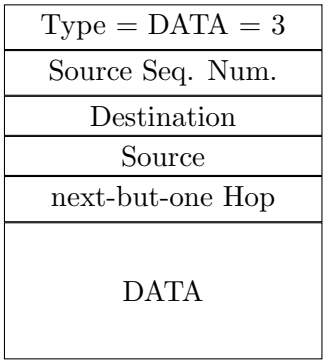


Figure 4.13: DATA Message Format in BuckshotDV

Once the RREP message has arrived at the source and the routing table entry has been created, the DATA packet can be transmitted. As no new route needs to be learned from a data packet, the identities of the previous and current hop are omitted in DATA packets.

The data packet format used in BuckshotDV is shown in figure 4.13. Just like when forwarding a RREP message, each node that receives a DATA message checks its neighbor table for the next-but-one hop listed in the message and replaces it with its own next-but-one hop for the listed destination if and only if it has found the neighbor in its neighbor table.

4.2.2 Variations

A possible variation of BuckshotDV concerns the DATA messages. In the basic version, they contain only one entry that changes with each hop: The next-but-one hop which is used for routing decisions. It would be possible to include the current and previous hop, to learn about the path that has been taken by the DATA message. Then, a node that receives a message could update its routing table entry for the source of the data message.

4.2.3 Advantages and Disadvantages

When compared to pure Buckshot Routing, BuckshotDV is complicated and requires more computation and copying on each node. Still, when comparing it to protocols like AODV, it remains simple. Its main advantage compared to Buckshot Routing is its scalability. In Buckshot Routing, as in all source routing protocols, the message headers grow with increasing network diameter. In BuckshotDV the header size is constant for each type of packet, making it usable in large networks. However, where Buckshot Routing is able to use route shortening if some of the intermediate nodes move closer to the source, BuckshotDV is not. The only point where BuckshotDV could use route shortening is when the destination becomes a direct neighbor of one of the intermediate nodes, which could then find it in its neighbor table.

4.3 Overhearing Supported BuckshotDV

In the previous section (4.2) BuckshotDV was introduced, which focuses on reducing the size of the messages transmitted by Buckshot Routing. While a reduction of the number of bytes that need to be transmitted is of course a reduction of energy consumption and network load, its gain is only limited for small networks. This is due to the fact that only a few node identities can be removed. But even in larger networks, the number of messages will presumably be the most important factor, as each routing message may also be fitted with a MAC header. Even if it is not, some of the current hardware used in sensor networks needs quite long preambles to synchronize sender and receiver, thus increasing bandwidth consumption drastically. For this reason, Overhearing Supported BuckshotDV (OSBRDV) focuses on possibilities to reduce the number of transmissions while keeping messages nearly as small as in BuckshotDV.

The basic idea behind OSBRDV is to delay messages on nodes that are not on the original route, and refrain from sending them if the node on the original route forwarded the message. This can be determined by overhearing the next-but-one hop node (as listed in the message, i.e. the direct successor of this node) sending the message in question to a node further along the path. The problem that arises with this new approach is that nodes now need to know whether or not they are on the original path. This leads to the need of transmitting not only the identity of the next-but-one hop inside a data message, but also including the direct next hop. Thus the message size is increased again, by the size of one identity. But as the size of an identity is assumed to be small in wireless sensor networks, it can be safely assumed that reducing the number of messages only by a fraction will already be enough to even out the additional identity transmitted.

In an optimal case, if all links on the path and those to all neighboring nodes are bidirectional, only nodes that are on the path would transmit the message. But this case is very unlikely as the experiments described in section 2 have shown. Normally, there will be unidirectional links around which messages have to be routed by OSBRDV.

4.3.1 Message Types

OSBRDV uses the same three message types BuckshotDV uses: Route Request, Route Reply and DATA. The Route Request message format stays exactly the same, only the other two are changed. However, the handling of a RREQ message is slightly different: Instead of storing only one node identity upon reception of a RREQ or RREP, two are stored in the routing table now (table 4.4).

Table 4.4: Routing Tables in OSBRDV

Destination	Next Hop	Next_but_one Hop	Hop Count
D	A	B	3

Figure 4.14 shows the message formats for RREP (l) and DATA messages (r). The RREP now contains four node identities in the variable area: The next-but-one hop is used to determine whether a node is a candidate for forwarding or whether it has to discard the message right away. If the value found in the field "next hop" equals the identity of the node that is working on the message, the message is retransmitted without delay, otherwise it is delayed for a certain time, depending on parameters like MAC protocol, per-hop-delay and similar. The current and previous hop identities are needed to build the reversed route on this node and on the next one, just like in BuckshotDV.

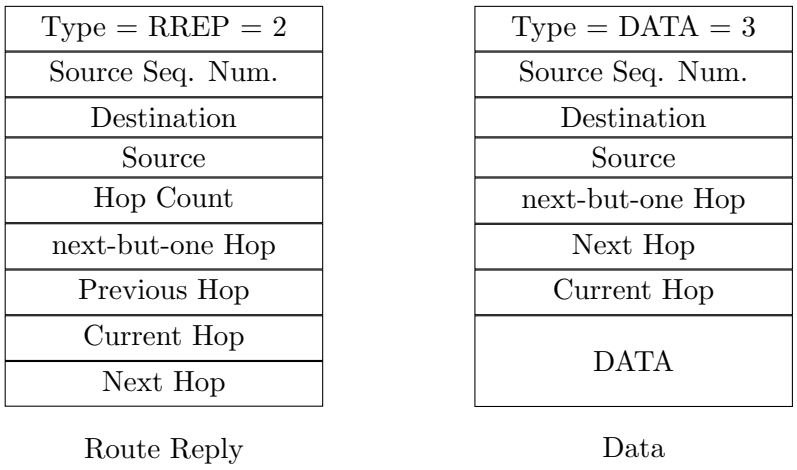


Figure 4.14: RREP and DATA Message Format in OSBRDV

The DATA message only contains three varying fields, the next-but-one hop, the next hop and the current hop. The first two fields are used to determine if the node that has received the message is a forwarding candidate or even on the direct path. Depending on this, the forwarding is delayed or not. The current hop is used upon message reception to determine if the message has already been entered into the list of deferred messages during its previous hop.

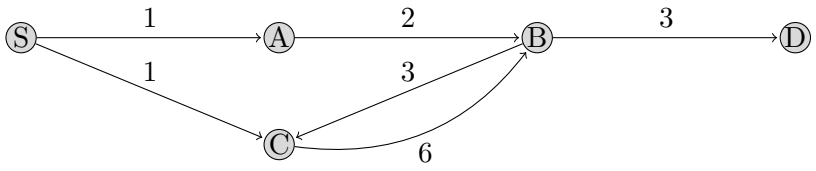


Figure 4.15: Transmission of a RREP or DATA Message in OSBRDV

Figure 4.15 shows an example transmission of a RREP or Data message in OSBRDV. In the first step, the source (S) transmits a message for the destination (D) with next hop A and next-but-one hop B. This message is received by nodes A and C. Node A is the intended next hop and forwards the message in step 2, after the current hop, next hop and next-but-one hop are adjusted. Node C, on the other hand, is not on the direct path. It stores the message in its list of deferred messages. When node B receives the message, it acts exactly like node A and forwards the message after the necessary adjustments.

Now there are two different scenarios: Either node C receives the message from node B in step 3, or it does not. If node C receives the message, it searches in its list of deferred messages for the source and sequence number. In the stored message, the next-but-one hop is denoted as node B. If this entry matches the current hop of the received message (and it does), the received message has already traveled closer to the destination. Therefore, node C discards its stored message, thus reducing network load.

If node C does not receive the message from node B in step 3 (or later), it assumes that the intended forwarder was unable to forward the message, and transmits the stored message after a certain timeout. In the example, a timeout of five time units has been chosen, therefore the message is forwarded in step 6.

4.3.2 Variations

To keep the routing tables up to date, it could be possible to learn routes from DATA messages. But that would require including the previous hop, as in the route reply messages. Also, it could be possible to replace the handling of messages by nodes that are not on the direct route: A probabilistic forwarding could be used. Using a random number generator, two thresholds (F and D) could be customized. If the random number R was below F, the message would be forwarded directly. If R was between F and D, the message would be discarded. If R was higher than D, the message would be deferred and forwarded if no retransmission by the next hop was overheard.

The first case is used to increase the chance of successful message delivery to the next but one hop. The second case reduces the network load while the third case represents the normal operation of OSBRDV and the redundant transmission needed to guarantee a certain delivery ratio.

Actual values for F and D should be chosen according to the average route length, frequency of changes and frequency of unidirectional links.

4.3.3 Advantages and Disadvantages

OSBRDV needs to transmit less messages than the previous two versions of Buckshot Routing. How much less depends strongly on the network topology and the number of unidirectional links involved. On the other hand, it gets more complicated, since a timer has to be set at configuration time and evaluated at runtime. If the configured timeout is too low, more messages than necessary will be transmitted. If it is too high, long delays will be introduced on lossy networks. The additional identity that is needed to determine whether or not a node lies on the original path must be stored in the routing tables and transmitted in data packets, thus increasing local storage and message size. But the fact that less messages need to be transmitted could be worth these side effects (see chapter 5).

4.4 Unidirectional Link Triangle Routing

While the previously described protocols focused on using existing unidirectional links implicitly, the two following routing protocols need to know about the existence of unidirectional links beforehand. This is realized using a neighborhood discovery protocol which is described in the next subsection.

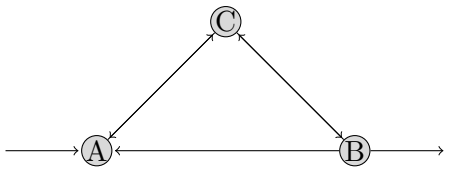
In unidirectional link triangle routing, a neighborhood table entry on node A consists of the ID of the neighbor (e.g. B) , the status of the link to that neighbor (bidirectional, unidirectional-incoming or unidirectional-outgoing) and, if the link is unidirectional-incoming, the identity of another neighboring node (e.g. C), which can be used to forward data to the node in question (node B). Table 4.5 shows an example for all three kinds of links.

Table 4.5: Routing Tables in ULTR

Destination	Next Hop	Link Status	Forwarder
D	A	bidirectional	none
E	B	incoming	C
F	G	outgoing	none

When a node wants to transmit a message to another node that is not included in its neighbor table or its routing table, it starts a route discovery by transmitting a route request (RREQ) message. This message is flooded through the network and creates routing entries for the source on all nodes it passes. The entries include only the next hop and the distance, resulting in a distance-vector protocol like e.g. AODV.

However, the handling is different once the destination has been reached and transmits the route reply. When a node receives a message that is not flooded, i.e. a route reply (RREP) or DATA message, it checks its routing table to find out which of its neighbors is the intended next hop just like in AODV. Unlike AODV, there is another step after that one. Once the node knows the neighbor that has been chosen to forward the message, it checks its neighbor table to see if the link to that node is *currently* an unidirectional-incoming one. If it is, and a detour of one hop is possible, the node forwards the packet first to the detour node which in turn retransmits the message to the intended node. Otherwise the message is silently discarded. Please note that broken links can be treated just like unidirectional-incoming ones.



ID, Status, Forwarder
B, incoming, C
C, bidirectional, none

A

ID, Status, Forwarder
A, outgoing, none
C, bidirectional, none

B

ID, Status, Forwarder
A, bidirectional, none
B, bidirectional, none

C

Figure 4.16: Neighbor Table Entries in Unidirectional Link Triangle Routing

Figure 4.16 shows a small part of a network and the corresponding neighborhood table entries used in this protocol: The nodes A, B and C from the example above are connected bidirectionally, with the exception of the link between nodes A and B which is unidirectional, enabling only transmissions from B to A. The neighborhood table of node A consists of two entries, a bidirectional one for node C and a unidirectional-incoming one from node B, with node C denoted as designated forwarder. The neighborhood table of node B contains node A, which would not be possible without a two-hop neighborhood discovery protocol, as node B does not receive any messages from node A. The link is marked as unidirectional-outgoing, and thus does not need any forwarder. The second entry features node C with a bidirectional link, needing no forwarder either. Finally, the neighborhood table of node C contains nodes A and B, both marked as connected through bidirectional links and not needing any forwarders.

Due to the fact that the unidirectional link and the detour that is taken on the way back form a triangle, this protocol is called unidirectional link triangle routing (ULTR).

ULTR is similar to the link layer tunneling mechanism proposed by the unidirectional link working group of the IETF (see section 3.1.11), but does not require multiple interfaces on the nodes to communicate. Also, depending on the used neighborhood discovery protocol, it may even be able to work with triangles which include more than one unidirectional link, which the link layer tunneling mechanism cannot handle. Moreover, ULTR works completely on the routing layer, the link layer is not involved. This is an advantage when timeouts are used, because the extra hop and thus longer delay are not hidden from the routing layer.

4.4.1 Neighborhood Discovery

The neighborhood discovery protocol needed for ULTR can be quite simple and needs only be started on a node once it receives the first message from a neighbor, i.e. when the first route request message is flooded into the network. Once it has been started, the neighborhood discovery protocol regularly transmits a message containing the IDs of all nodes from which this node has received messages recently and the status of its links to and from them. When a node receives such a hello message, it checks whether its ID is contained therein. If it is not, the receiving node knows that it is on the receiving side of a unidirectional link.

In other protocols, where unidirectional links are not used, a lot of overhead would now be necessary to inform the upstream node (the sender of the hello message) of the unidirectional link. In this protocol, the upstream node does not need to know about its existence. The receiving node only marks the link as unidirectional-incoming in its neighbor table.

When a node A receives a hello message via the bidirectional link from node C in which the upstream node of the unidirectional link is listed and the link to that node (from C to B) is marked as bidirectional, node A enters the sender of the hello message (node C) as a forwarding neighbor into the corresponding neighbor table entry (for node B). Please note that this would also be possible if there was a unidirectional link from C to B, but the proactive detection of this special case would probably introduce a large overhead and solve only one special case: If there is a unidirectional link from C to B and no other neighbor of A has a bidirectional link to B.

When a message (RREP or DATA) is sent the reversed way, it needs to be forwarded along a one-hop-detour. This message can be used to inform the upstream node of the link, which is then entered into the upstream node's neighborhood table as unidirectional-outgoing. Please note that for the routing alone this information would not be necessary, indeed it would be easy to hide the fact that the message has taken a detour. But for the sake of timers that can be used for retries on MAC- or routing layer it helps to know that the delay could be twice as high. In this case, the information about this special link can be acquired "for free" and could be used to solve the problem described above. The information about the unidirectional-outgoing link can also be used by the MAC layer not only for retries, but also to determine the right two-hop neighborhood of a node, which is a mandatory information for TDMA protocols.

4.4.2 Message Types

ULTR uses the same three message types as all other protocols described in this chapter, Route Request, Route Reply and DATA. Figure 4.17 shows an example for each of them:

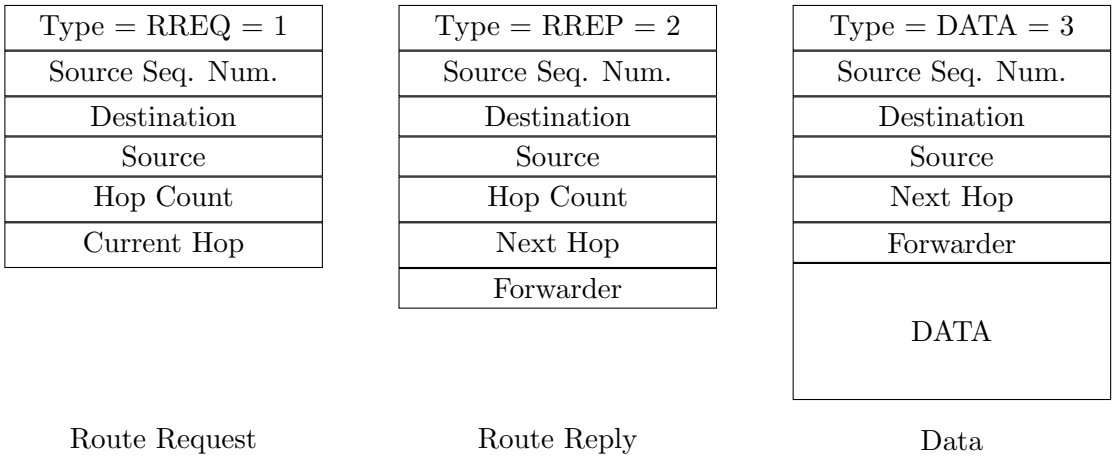


Figure 4.17: Message Types Used in ULTR

A RREQ message contains the identity and sequence number of the source which are used for duplicate detection, followed by the identity of the destination. The hop count is incremented by one on each hop as usual, and the identity of the last hop is used to build the backward route. A Route Reply message contains sequence number and identity of the source for duplicate detection as well as the identity of the destination. For forwarding purposes the next hop and, if necessary, the forwarding node are included. The DATA packet contains the sequence number and identity of its source as well as the identity of its destination and, of course, the application data. This is followed once again by the identities of the next hop and, if suitable, the forwarding node.

4.4.3 Variations

ULTR relies on a neighborhood discovery protocol, which might introduce too much overhead, depending on the protocol, its configuration and the application scenario. In order to get rid of this overhead, a variation without neighborhood discovery can be used: Nodes that try to forward a message and do not get a confirmation that the next hop received the message (either through link layer acknowledgments or overhearing) can use a localized version of BuckshotDV (see section 4.2) instead. This way, detours of a configurable length would be possible.

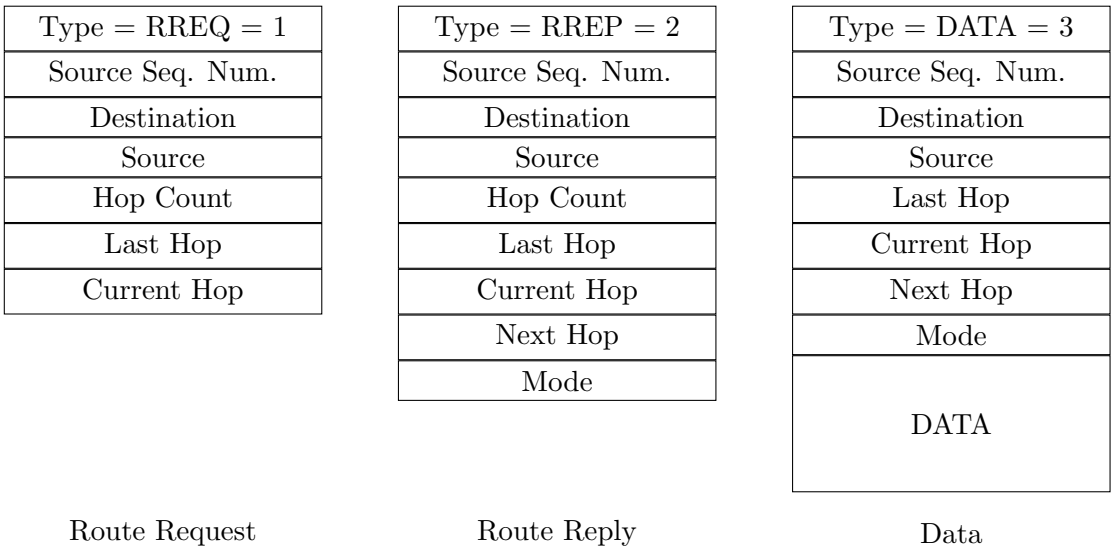


Figure 4.18: Message Types in ULTR Without Neighborhood Discovery

When this variation is used, some modifications to the message types are necessary (see figure 4.18). Information about the last hop would have to be included in RREQ messages, in addition to the current hop. Both node IDs are stored in the routing table. A node decides which entry to use depending on the overheard status of the link. If the next hop is assumed to be connected by a bidirectional link, the normal next hop is used. Otherwise the message is set to Buckshot mode and the next-but-one hop is used. The last hop is also used for implicit link detection: If a node overhears the transmission of a message in which it is denoted as last hop, it knows that the link between itself and the current hop denoted in the message is currently bidirectional.

A RREP message contains three node IDs instead of only two: The last hop ID and current hop ID are used to build the backward route for normal and for Buckshot mode just as they are used in the RREP. The next hop ID is used for forwarding. However, the RREP also contains a flag denoting the mode of transmission, which can take on the values ”normal” and ”buckshot”. It is evaluated upon message reception to decide if a node shall forward the message or not. In normal mode it only forwards the message when it is denoted as next hop in the message, in Buckshot mode it also forwards the message if it has the next-but-one hop in its neighbor table.

The DATA message features the same three node IDs that are present in the RREP message. For routing purposes alone, the last hop ID would not be needed, but it is nevertheless included for link status detection. The mode flag is also present again, to enable the usage of one-hop Buckshot Routing if the status of the next link is unknown or known to be unidirectional-incoming.

4.4.4 Advantages and Disadvantages

This protocol is by far the most complex protocol presented yet. The complexity is the price for the reduced number of data packet transmissions, as no flooding of DATA packets, not even a limited one, is used. Periodic updates of the neighborhood table ensure that the link status information it holds is always up to date, which also enables implicit local repair. Altogether this should lead to a higher delivery ratio. On the downside the usage of hello messages also leads to more protocol overhead, as these messages can be quite large in dense networks. Therefore, the typical tradeoff between actuality and network load has to be made when setting the hello period, which makes configuring the protocol harder. On the other hand a new option for cooperation between MAC and routing arises.

Like all routing protocols that use unidirectional links, ULTR also needs a MAC that can transmit over unidirectional links. The information about the existence of the unidirectional links probably needs to be collected to a certain extend anyway, depending on the MAC protocol used. So either this can be retrieved from the MAC without additional cost, or the MAC protocol can query the routing layer for it using an appropriate interface. More information about the cooperation options between MAC and routing is provided in section 4.6.

4.5 Unidirectional Link Counter

In the protocols previously described unidirectional links were only used implicitly (in the Buckshot variants) or with local detours (in Unidirectional Link Triangle Routing). In none of these protocols information about unidirectional links was made available to distant nodes. In the case of ULTR this could even lead to a suboptimal choice of route if the route taken by the RREQ consisted mostly of unidirectional links. Then, the reversed route would still work, but messages sent along it would use up to twice as many hops and take a much longer time. Furthermore, if no local detour exists, messages will get stuck.

4.5.1 Message Types

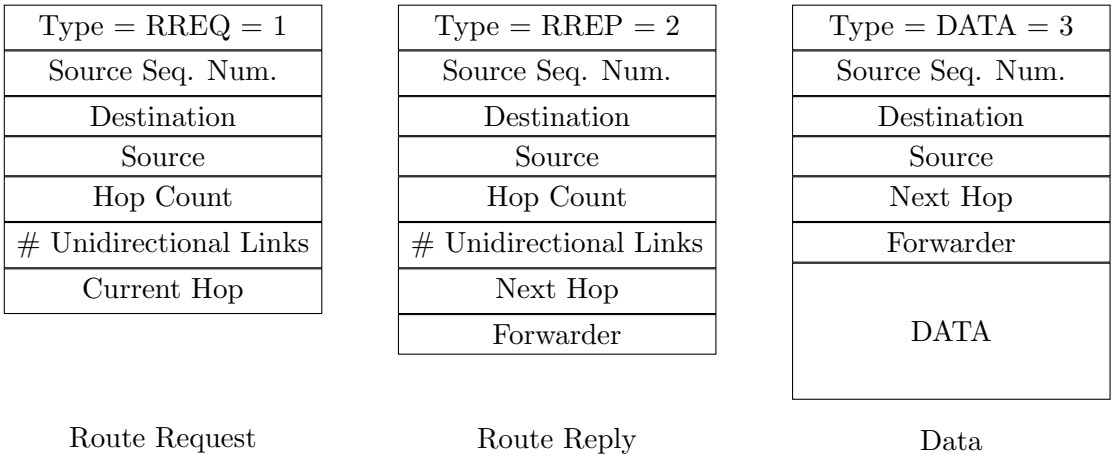


Figure 4.19: Message Types used in ULC

In Unidirectional Link Counter (ULC), these two problems are solved. When a node receives a route request message, it checks its neighbor table for a way back to the last hop. This can be a direct way (if the link is bidirectional) or a local one-hop detour. If the way back is bidirectional, the hop count of the route request is increased by one and it is retransmitted. If the link is marked as an unidirectional-incoming one but a one-hop detour to the last hop is known, the number of hops and the number of unidirectional links passed by this RREQ are both increased by one. Please note that this requires the new field "Number of Unidirectional Links" to be present in the RREQ and RREP messages (figure 4.19). The DATA message stays exactly the same as in ULTR.

As this inclusion of the number of unidirectional links in the RREQ and RREP is essential to this protocol, the protocol is called *Unidirectional Link Counter* or ULC. When a destination receives a route request message, it can read the number of hops the message has taken to reach it in the hop count as usual. But in this protocol it can also calculate the number of hops the route reply will probably need by adding the number of unidirectional links to the hop count. Once multiple RREQs have been received by the destination, a number of different weight functions can be used to determine which one of these routes should be used.

4.5.2 Possible Weight Functions

Which weight functions are available depends heavily on the intended scenario. They include, but are not limited to: Least number of hops, least number of hops with unidirectional links added or best average route. Once the weight has been calculated, it can be entered into the routing table. Table 4.6 shows an example of a routing table for ULC.

Table 4.6: Routing Tables in ULC

Destination	Next Hop	Route Weight
D	A	4

Least Number of Hops When using this weight function, the shortest path from source to destination is chosen. This can be very useful when the main traffic is expected to flow from the source to the destination with only a marginal amount going the other way. An example for this behavior would be a sensor network with a sense-and-send application where periodic data is sampled automatically and acknowledged only occasionally.

Least Number of Hops with Unidirectional Links Added The addition of both values delivers the maximum path length used in communication between source and destination of the RREQ. This can be important when the packet loss rate for a single hop is high. As the probability of successful transmission along a path (P_{path}) is the potentialization of the probability for a single hop (P_{hop}) by the number of hops (x), the number of hops is the only variable and thus most important factor: $P_{path} = {P_{hop}}^{x}$.

Another scenario that uses this weight function might be the usage of a time sensitive application. If messages are expected to travel not only from the source to the destination but also vice versa within certain time boundaries, the maximum length of one direction is set.

Best Average Route When the number of expected transmissions from source to destination is about as high as the number of transmissions expected from destination to source, it is useful to keep the average length of the paths small. This is achieved by simply adding half the number of unidirectional links to the hop count, as the local detours are needed only in one direction. This weight function is also useful when the application involves a query-response traffic pattern, meaning that messages always need to travel from the source to the destination and back. For such scenarios, the round trip time, i.e. the number of hops needed to travel both directions once, is most important.

4.5.3 Neighborhood Discovery

The neighborhood discovery protocol must fulfill the same requirements as the one described in ULTR. Nodes need to know about their incoming unidirectional links, and about the existence of a one hop-detour to the source for every unidirectional link. Therefore, the same protocol can be used, which regularly transmits the identities of all neighbors and their link status. The handling of incoming hello messages is also the same.

4.5.4 Variations

The neighborhood discovery protocol used in ULC is necessary, because every node needs to know the status of its links before the first RREQ message is received, in order to increase the counter for unidirectional links if the message was received over such a link. If this information is not provided or not used by the underlying MAC protocol as well, the induced overhead can make ULC unfitting for a number of application scenarios. Therefore, a variation has been designed, which does not require a neighborhood discovery protocol:

Nodes that receive a message over a link for the first time set the status of that link as unidirectional-incoming. If a node overhears the forwarding of a message it has already transmitted by a neighboring node, the link to that node is set as bidirectional.

This passive evaluation of links leads to three drawbacks in link information quality: First of all, unidirectional-outgoing links can no longer be detected. However, this is not such a big problem, as it is only necessary to route around unidirectional-incoming links. Second, the protocol needs a certain settling time. When the first RREQ message in the whole network is transmitted, all links it passes will be listed as being unidirectional. But once a few messages have been transmitted from different sources, the neighborhood tables should be much more accurate. The third drawback is that link changes can only be detected by using a timeout for all links and removing neighbor table entries after a certain time. This raises the need to learn about neighbors once more, when new messages are transmitted.

When a Route Reply or DATA message is transmitted and a node that should forward it finds the connection to the next hop listed as *unidirectional-incoming*, the node switches to a limited usage of BuckshotDV for the next hop of the message.

The usage of passive link status monitoring once again makes some modifications to the message format necessary. Figure 4.20 shows the the message types:

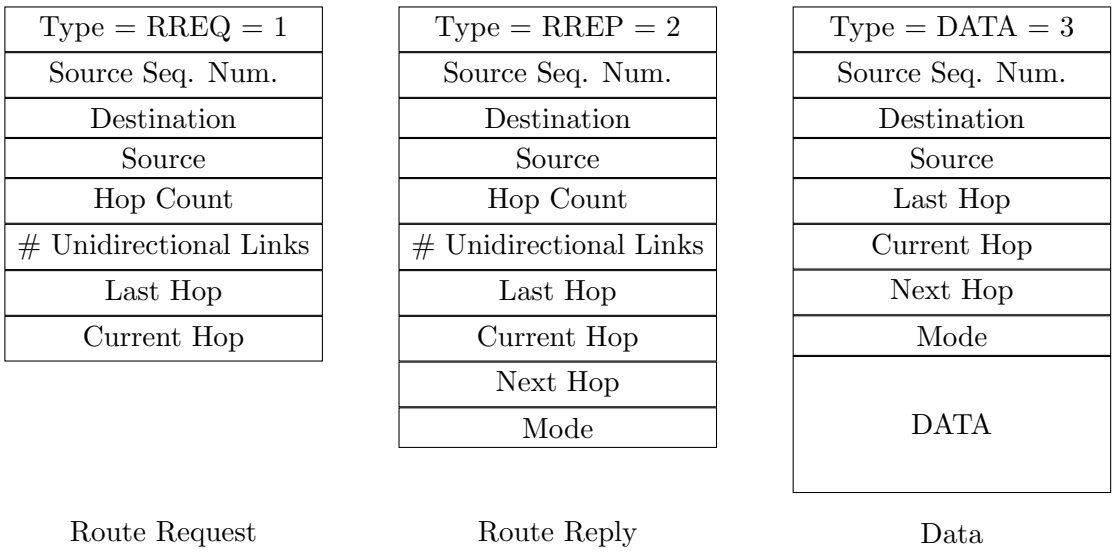


Figure 4.20: Message Types in ULC Without Neighborhood Discovery

Route Request messages now contain the last hop in addition to the current hop. This is used to detect bidirectional links: If a node receives a message in which it is listed as last hop, it enters the link to the current hop denoted in the message as bidirectional into its neighbor table.

RREP and DATA messages use three node IDs: last hop, current hop and next hop. Last and current hop are used for link detection and to build the inverted route just as in the RREQ message. The next hop is used to decide if the node that received the message should forward it. Here the mode flag is evaluated first. It can once again take on the values ”normal” and ”buckshot”. In normal mode the field ”next hop” contains the ID of the intended forwarder, all other nodes discard the message. In Buckshot mode the field actually contains the next-but-one hop, and all nodes that have this next-but-one hop in their neighbor table forward the message.

Table 4.7: Routing Tables in ULC

Destination	Next Hop	Next_but_one Hop	Route Weight
D	A	B	4

Nodes that forward a RREP or DATA message look up the next hop in their routing table and check the link to it in their neighbor table. If the link is listed as being bidirectional the message is sent in normal mode, otherwise Buckshot mode is used. Enabling the usage of Buckshot mode requires including the next_but_one hop into the routing table of ULC (table 4.7).

4.5.5 Advantages and Disadvantages

The unidirectional link counter offers a number of possible weight functions, which can be chosen according to user specifications. The fact that the RREQ messages collect information about the unidirectional links they have passed means that this information is freely available to other protocols. For example, it can be combined with a monitoring protocol like Sensorium [50], and used for passive network monitoring. ULC can also cooperate with the MAC layer when detecting unidirectional links and one-hop-detour neighbors (see section 4.6). Upstream nodes of unidirectional links do not need to be informed explicitly of their existence as in ULTR. They learn of their outgoing unidirectional links only when a route reply or DATA packet is sent via the detour, and only supply the information to possibly existing retransmission protocols.

4.6 Cooperation with the MAC-Layer

All protocols described in this chapter were designed specifically to utilize unidirectional links. This makes it imperative to use a MAC layer that can also transmit over unidirectional links. Any protocol that uses the standard ”request to send” - ”clear to send” mechanism is completely unsuitable, as no clear to send message will ever be received over an outgoing unidirectional link. Moreover, nodes with an outgoing unidirectional link will never know that they could be disturbing the communication between two other nodes (see section 2.2). There are some improvements that allow contention based protocols to work with unidirectional links, e.g. ECTS-MAC [42, 44] (see section 3.1.4). Some of the MAC protocols that utilize unidirectional links route their link layer acknowledgments back to the upstream nodes. For this, the neighborhood table used by ULTR (section 4.4) and ULC (section 4.5) could be reused.

Plan based MAC protocols need to know the two-hop neighborhood of each node to identify the collision domain. Within this domain, the varying parameter (e.g. frequency (FDMA), code (CDMA) or slot (TDMA)) needs to be unique for each node. Therefore, a neighborhood discovery protocol is needed which finds these two-hop neighbors. The protocols used for ULTR or ULC could easily be enhanced to deliver this information. Otherwise, if the MAC protocol already has its own neighborhood discovery protocol, it only needs to make the gathered information available to the chosen routing protocol.

The usage of such a neighborhood discovery protocol would also implicitly solve the ”special case” of a unidirectional link triangle with more than one unidirectional link, enabling the unidirectional link triangle and unidirectional link counter protocols to make use of such links as well.

This usage of a single neighborhood discovery protocol for both MAC and routing reduces communication overhead and memory consumption by far. It also ensures that both layers work on the same data. If they would use different algorithms, different storage sizes or replacement strategies, lots of problems could result, as described e.g. in Murphy Loves Potatoes [36] (see section 2.1.9).

4.7 Own publications referenced in this chapter

- David Peters, Reinhardt Karnapke, Jörg Nolte "Buckshot Routing - A Robust Source Routing Protocol for Dense Ad-Hoc Networks", Ad Hoc Networks Conference 2009, Niagara Falls, Canada, 2009. [58]

- Stephan Nürnberger, Reinhardt Karnapke, Jörg Nolte "Sensorium - An Active Monitoring System for Neighborhood Relations in Wireless Sensor Networks", Second International Conference on Ad Hoc Networks, Victoria, BC, Canada, 2010 . [50]

- Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. Mac protocols for wireless sensor networks: Tackling the problem of unidirectional links. In International Journal on Advances in Networks and Services, vol 2 no 4, pages 218 - 229, 2009. [42]

- Stephan Mank, Reinhardt Karnapke and Jörg Nolte "MLMAC-UL and ECTS-MAC - Two MAC Protocols for Wireless Sensor Networks with Unidirectional Links", Third International Conference on Sensor Technologies and Applications Athens, Greece, 2009. [44]

Chapter 5

Evaluation

To the best of the author's knowledge, no sensor network routing algorithms exist, that make use of unidirectional links. Therefore, the protocol most used in deployments was chosen for comparison: A tree routing based approach with retransmissions, which is quite common in sense-and-send applications where all nodes transmit their data to the sink regularly (e.g. [71, 13, 21, 36]). As this may seem to be an unfair comparison, two protocols from the MANET area were also chosen as competitors: DSR in the version that uses unidirectional links (see section 3.2.9) and AODV-BR (see section 3.2.3). AODV-BR does not use unidirectional links, but has an interesting way of detecting them and salvaging the data message that caused the detection, which is somehow similar to the forwarding mechanism used in Buckshot Routing. As fourth reference protocol, `Flooding` is included. While it is known that `Flooding` induces a lot of overhead, it can still deliver valuable insights. In the simulations, `Flooding` is used to determine the upper limit of messages that could reach the destination. In the real world experiments carried out for this work, the network load it generates is used to understand the performance of the MAC protocol supplied by the hardware in use.

The distance measured in hops is taken as weight function (minimum hop routing), but other weights, e.g. residual energy, could also be used with the same result, as all protocols would work on the same values. Routes with a lower weight replace older ones with a higher value in the routing tables.

The authors of [49] propose a combined evaluation method that uses experiments with real hardware, emulation and simulation techniques in order to speed up the deployment of new protocols. The combination of all three methods enables the developer to identify which problems occur and shows him/her where further investigation is necessary. The routing protocols AODV, DSR and OLSR were used to evaluate the proposed approach to protocol monitoring.

The authors found that latency and timing are crucial to the performance of reactive protocols like AODV and DSR, because of buffering times. The queue-ups that can result from this buffering were apparent in the experiments, but not in the emulations.

In conclusion of this paper it can be said that all three methods of evaluation have their own gain for a protocol developer, if they are used correctly. For simulations, the choice of the underlying communication model is crucial. The emulation can be fed with real world connectivity data, and can be used to evaluate the implications of the network stack used on the real devices. Experiments are needed to generate this connectivity data.

It is important that for all three methods exactly the same implementation of the protocol is used, and that this implementation is the one that can be used directly on the hardware which is used in the real experiments.

Following this approach and the advice from Stojmenovic [69], the same implementation was used for both simulations and real world experiments in this evaluation (see appendix A for details). In the next section, the methods used in the simulations are described. They enable the evaluation of the algorithms and their ability to handle unidirectional links under controlled circumstances (section 5.1). The general principle of the real world experiments, including the chosen locations, is described in section 5.2.

After these two methods of evaluation have been described, the results of connectivity measurements conducted on the different real world locations is described (section 5.3). The actual evaluation of the routing protocols designed in this thesis and those chosen for comparison is presented in sections 5.4 to 5.6, sorted by the application scenario in use. Following these evaluations, the interaction between duplicate suppression mechanisms and routing protocols is discussed(section 5.7) before a summary of the achieved results is given in section 5.8.

5.1 Simulations

All simulations were performed using the discrete event simulator OMNeT++ (see appendix A). The simulated networks consisted of four different sizes of grids: 100 nodes (10x10), 400 nodes (20x20), 900 nodes (30x30) and 1600 nodes (40x40). A grid alignment was chosen to represent applications that need area coverage, where each node is equipped with sensors that have a range of one distance unit. But, as will be seen below, the exact placement of the nodes is not important, because connectivity is determined using a connectivity matrix (see section 5.1.1). The different numbers of nodes represent network sizes ranging from small to huge networks, and thus increase the number of hops needed to communicate from one end of the network to the other. This determines the route length, which has a tremendous impact on the performance of all routing protocols.

All simulations are restricted to the usage of a ”perfect behavior” MAC. While it is of course true that the choice of medium access control protocol can have a strong influence on the performance of the routing layer, the goal of the simulations is the evaluation of the ability of the routing protocols to work in the presence of unidirectional links, not of their interaction with the MAC layer. Also, many of the effects of a MAC layer, e.g. the available neighbors for each node, would be the same for all evaluated routing protocols. The effects could only differ between protocols, when they are depending on the generated network load, as different protocols transmit different types of messages with different sizes and in different frequencies. But all of these are highly dependent on the application, and it is not possible to evaluate all possible application scenarios.

As simulation results are never 100% accurate, real world experiments have been conducted, too. Details about the methods of evaluation used for the real world experiments are shown in section 5.2. This section follows Stoijmenovic's advice [69], and uses a simple model in order to keep side influences small and results interpretable.

5.1.1 Connectivity between Nodes

To simulate a certain connectivity between nodes, thousands of connectivity matrices were generated before running the simulations. The same generated matrices were used for all protocols. The large number of matrices is necessary to simulate the constantly changing nature of wireless links. As the largest networks, consisting of 1600 nodes, needed to be simulated for the longest time, they also needed the highest number of connectivity matrices: For a single simulation 17761 connectivity matrices were needed.

In each of these matrices, a (directed) link from node A to node B exists with a probability of α/d^6 where d is the distance between node A and node B. The inverse link, from node B to node A exists with the same probability. Therefore, the link is bidirectional with a probability of $(\alpha/d^6) \times (\alpha/d^6)$, unidirectional (in any one direction) with $\alpha/d^6 \times (1-(\alpha/d^6))$ and non existing with $(1-(\alpha/d^6))^2$. The quotient (d^6) reflects the dampening induced by the distance between nodes while α represents the probability that a link between geographically adjacent nodes exists.

Nodes were arranged on a regular grid to reflect application scenarios which need area coverage, e.g. vehicle tracking. As all nodes were arranged on a grid, nodes that are directly above, below, right or left of a node are called direct neighbors and their distance was defined as 1. α was varied between 0.9, 0.95 and 1, and for each value of α ten sets of matrices with different seeds for the random number generator were generated, leading to 30 sets of matrices per network size, and a total of 996120 connectivity matrices containing between 10.000 and 2.560.000 entries.

Please note that due to the fact that the matrices were generated randomly, there is no guarantee that there always was a path from sender to destination. Therefore, no upper limit can be calculated, but `Flooding` is used as reference protocol: The number of application messages delivered by `Flooding` is taken as 100% and the delivery ratio of all other protocols calculated accordingly.

5.1.2 Application Settings

In each simulation, each node wanted to transmit a total of 110 messages to one or more destinations, depending on the scenario. After the initialization phase of the network, one message was transmitted every 100 milliseconds. To ensure that route discovery was finished, the logging remained inactive until all nodes had started the transmission of their fifth message. The connectivity matrices were changed every second. Please note that the absolute values of the time units are not important for the simulation, only their relation (1:10). They could also have been set to 6 seconds and one minute yielding the same results.

5.1.3 Protocol Performance

In the simulations, logging only began once each node had started the transmission of its fifth message. Therefore the theoretical optimum of delivered messages could be calculated, if connectivity could be guaranteed. But the connectivity matrices were generated randomly, therefore network separation could be possible. `Flooding` delivered close to the theoretical optimum, and is used as maximum for the simulations. For all simulations, the delivery ratio of a protocol is defined as the number of messages delivered by the protocol divided by the number of messages delivered by `Flooding`.

5.2 Real World Experiments

To evaluate the influence of medium access control and the properties of real hardware, all nine protocols (five new ones and four from related work) were evaluated on the eZ430-Chronos [15] sensor nodes (see appendix A).

All protocols use the same sensor nodes on the same locations, meaning that node 0 used to evaluate `Flooding` is the same piece of hardware on the same location as node 0 used in the experiments evaluating BuckshotDV and so on. Depending on the application scenario, the experiments were conducted on some or all of the locations described below. Each protocol was evaluated using a freshly charged set of batteries.

5.2.1 Application and Logging

In the real experiments, each node wanted to transmit a message every minute. The experiments ran for one hour each, therefore 60 messages were transmitted by the application on each node. In all experiments, 36 nodes were placed in a square of six times six. Each node recorded the number of application messages it received, and all nodes recorded the number, type and size of all messages they transmitted or forwarded.

Like in the simulations, it was once again possible that nodes were disconnected from the network and suffered from network separation. Also, sometimes nodes failed due to hardware problems. Therefore, the type of messages transmitted by a node was evaluated, too. When a node only transmitted route request messages and not a single data message, it did obviously not find any route to the sink.

5.2.2 Protocol Performance

In the real world experiments, logging began at once. Therefore the theoretical optimum of delivered messages could be calculated, if connectivity could be guaranteed which is never the case in real world deployments. In contrast to the simulations, `Flooding` could not be used as reference protocol because it did not always deliver the highest number of application messages. Therefore, the delivery ratio is defined as the number of application messages delivered to their destination divided by the number of application messages transmitted.

5.2.3 Program Size

The size of the programs deployed on the eZ430-Chronos is shown in table 5.1. Please note that the values were measured for scenario 1 (sense-and-send, section 5.4), but differ only marginally for the other scenarios as the main components (system and routing protocol) are always the same. Only the application differs from scenario to scenario, but its influence on the program size is marginal.

It can be seen on the table that DSR has by far the largest memory footprint, concerning both flash ("text") and RAM ("bss"). The lowest footprint can be seen on `Flooding`. It needs only about 500 Bytes flash and 200 Bytes RAM more compared to the system without routing, most of which is needed for the duplicate suppression.

Table 5.1: Total size of the deployed systems for different routing protocols in Byte

protocol	text	data	bss	dec
AODV-BR	14590	0	2260	16850
Buckshot	14576	0	2424	17000
BuckshotDV	13954	0	1920	15874
DSR	17760	0	3586	21346
`Flooding`	12444	0	1644	14088
OSBRDV	14882	0	2536	17418
`Tree Routing`	13234	0	1990	15224
ULC	14718	0	2066	16784
ULTR	14550	0	2066	16616
System without routing	11918	0	1418	13336
Basic System	8612	0	994	9606
Connectivity Evaluation	11368	0	3632	15000

The basic system, including only the operating system Reflex (see appendix A) without any scenario specific parts (no routing protocol, no application) is also shown for comparison. It needs 8612 Bytes of flash and 994 Bytes of RAM. Most of the RAM consumption is due to the 10 network buffers with 64 Bytes each.

DSR did not fit on the microcontrollers with the settings used in the simulations, therefore some of them (e.g. the number of messages that can be stored) had to be reduced to make it fit. As DSR has the largest memory footprint, all other protocols had no problem fitting on the micro controller when using the same settings (see appendix A for details).

5.2.4 Experiment Locations

Four different locations were used for the real world experiments:

- On a Desk
- Affixed to Poles
- Placed directly onto a lawn
- Placed directly onto stones

Desk Experiments This deployment is a single hop layout, where each node is able to receive messages from each other node. The nodes lay directly next to each other. An old set of batteries was used without re-charging them, because range did not really matter in these experiments. They were used to validate the correct operation of the protocols.

(a) affixed to poles (b) placed on the lawn (c) on a stone pavement

Figure 5.1: A modified eZ430-Chronos Sensor Node

Poles For the pole experiments, small poles were deployed on the lawn in front of the main building of our university, with about one meter distance between each of them. Then, the sensor nodes were affixed to them using cable straps, at a height of about 20 cm (figure 5.1(a)). The pole placement was usually used at 8am.

Lawn After the pole experiments were finished and evaluated, the nodes were reset and placed on the ground directly next to the poles as shown on figure 5.1(b). The resets were done by disconnecting the batteries and reconnecting them directly afterwards. The same set of batteries as before was used on each node without charging. When using all four locations, the lawn experiments were started at about 10 AM.

Stones After the lawn experiments, the nodes were disconnected, and poles as well as nodes and batteries collected. The experiments on the stones were always started at about 1 PM, using the same set of 72 AA batteries used in the morning without recharging, but the pairing of batteries and nodes might have changed, i.e. the batteries that were connected to node 4 in the pole and lawn experiments might be connected e.g. to node 27 in the stone experiments. These experiments were conducted on the stone pavement on our campus (figure 5.1(c)).

5.3 Connectivity Evaluation

To get a feeling for the behavior of the real hardware and to keep the possibilities of application errors to a minimum, the first experiments were made using a fairly simple application. 36 sensor nodes were deployed on the lawn outside of the university's main building, spaced one meter from each other [38].

As only the connectivity should be measured, the "application" consisted of a simple flooding with duplicate suppression. Node 0 was connected to a laptop via USB and transmitted 50 messages, with a pause of one minute between messages. Each node that received a flooded message first logged the neighbor from which it received the message. After that, the node checked if it had already handled a message with this sequence number. If it had, the message was discarded, otherwise the node changed the field "last hop" to contain its own ID and rebroadcast the message.

Even this simple application ran into two problems: The CC430 uses a so-called CCA Medium Access Control, which is basically a CSMA/CA scheme. A node that wants to transmit a message waits for a random time (backoff) before sensing the medium. If it is free, the message is transmitted. Otherwise, the radio waits for a random time before trying again. The used hardware was not able to receive messages during the backoff, which meant that even in an experiment with 3 nodes (0, 1, 2) node 2 was never able to receive messages from node 1, because it was still in its backoff when node 1 transmitted. To solve this problem for the connectivity evaluation, a software delay was introduced. The software waited between 1 and 13 milliseconds before handing the message to the hardware. This delay could be tolerated, because application knowledge was available (node 0 transmitted a new packet only every minute).

Retrieval of data was induced by sending a message to a node, telling this node that it should transmit its gathered neighborhood information. Sadly, the nodes were unable to receive any messages after a seemingly random time. Sometimes, nodes would function only for a couple of minutes, while others ran for more than a day and still responded. The influence of stray messages on the application could be ruled out due to precautions in the software. The problem seemed to exist in the state machine of the radio. To remove this problem, a watchdog timer was introduced which reset the radio every five minutes if the application did not receive any messages during that time. If it did receive a message, the watchdog was restarted. While this could lead to problems if nodes radios failed during the experiment, it was mainly used to gather the results, once the sensor nodes were collected and returned to the office. Two different radio channels and four placements (lawn, stones, poles and trees) were evaluated. For each placement, the initial connectivity graph is shown here.

5.3.1 Lawn Experiment, Channel 0, Sink (Node 0) connected to Laptop

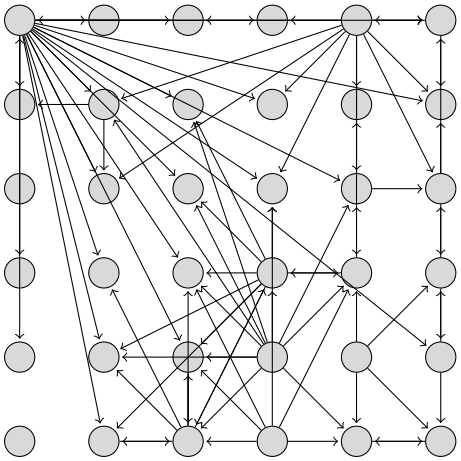


Figure 5.2: First Connectivity Graph obtained on the lawn, channel 0

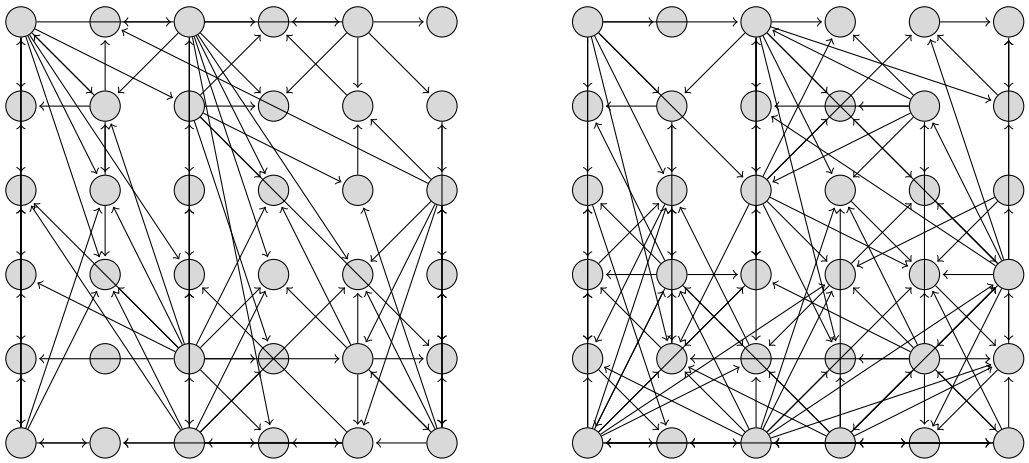
The first experiment was conducted on the lawn in front of our university. Figure 5.2 shows the connectivity graph obtained for the first of the fifty messages that were flooded into the network. One of the nodes, node 30, had a defective battery contact and did not participate at all. Four other nodes, nodes 12, 27, 28 and 33 suffered a complete reset during transportation, leading to loss of the connectivity data they gathered. Still, a lot of information could be obtained.

Node 0, which was connected to a laptop using a USB cable, was heard by lots of nodes, even those far away like node 11, node 29 or node 31. This shows that the transmission strength of the nodes, while it was set to 0dBm for all nodes, still depends on the power supply, i.e. the batteries. In deployments where a sink node connected to a fixed power supply such as a computer should be used, the longer reach of the sink node might well be a problem. This problem would for example manifest, when a tree routing approach is used, and the sink floods a message through the network to establish initial father and child connections between nodes. Most of the nodes would assume node 0 as their father, but be unable to transmit directly to it. Also, the results show that even though the nodes were only one meter distant from each other, bidirectional links are rare and unidirectional links are common. Counting all links, 3018 unidirectional and only 403 bidirectional links have been recorded. If the unidirectional links from the nodes that have failed during transport are excluded (560 seemingly unidirectional ones), the ratio is still 2458 unidirectional links against 403 bidirectional ones.

To remove the influence of the higher transmission strength of the "sink" (node 0), all links to and from node 0 can be removed from the equation. But even then, the result seems pretty obvious: 1477 unidirectional links stand opposed to 355 bidirectional ones (ratio 4.16 : 1). As for the theory of stable links, 7019 link changes were recorded during the 50 minute deployment.

5.3.2 Lawn Experiments, Sink (Node 0) connected to batteries

To remove the influence of the USB cable connected to node 0 completely, the experiment was repeated. This time, and in all subsequent experiments, node 0 used a normal battery pack like all other nodes. The experiment was conducted on two different channels, namely channel 0 and channel 3. The initial connectivity graphs are shown in



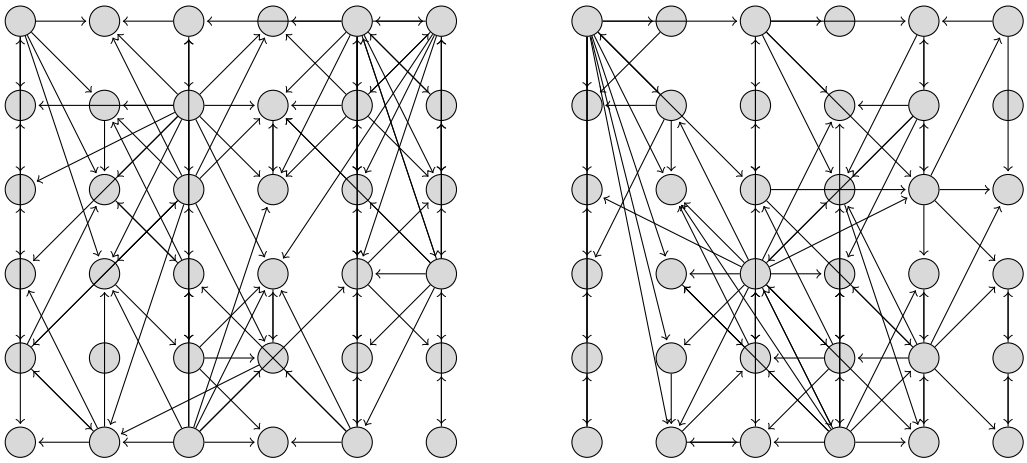
(a) Channel 0 (b) Channel 3

Figure 5.3: First Connectivity Graph obtained on the lawn with node 0 connected to batteries, for both channels respectively

figure 5.3(a) for channel 0 and in figure 5.3(b) for channel 3. Even though precautions were taken, one node (node 25) still suffered a reset before the gathered data could be retrieved during the experiment on channel 0. The application was the same, with 50 flooded messages. 4039 unidirectional links as well as 818 bidirectional links with 7019 changes were recorded on channel 0, if the links from node 25 are removed that still leaves 3912 unidirectional ones opposing 818 bidirectional links (4.78 : 1 ratio) over the length of the whole experiment. On channel 3, as much as 4411 unidirectional links and 757 bidirectional ones (ratio 5.83 : 1) and 7103 link changes were measured.

5.3.3 Stone Pavement Experiments

To evaluate the influence of the ground on which the sensor nodes were placed, the experiments were repeated again, but this time the nodes were placed on the stone yard of the university. Figure 5.4(a) once again shows the first connectivity graph obtained on channel 0. Altogether 3570 unidirectional links and 851 bidirectional ones were



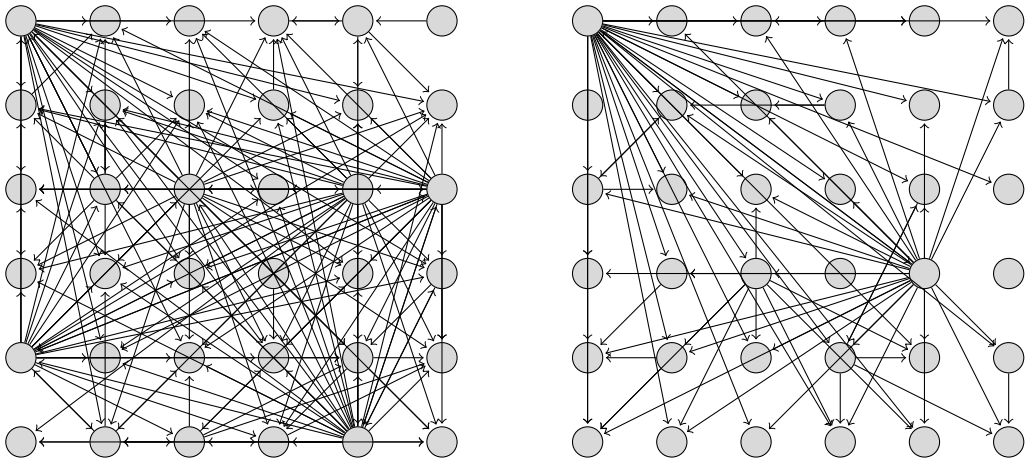
(a) Channel 0 (b) Channel 3

Figure 5.4: First Connectivity Graph obtained on the stone pavement for each channel respectively

measured on channel 0, resulting in a ratio of 4.19 : 1. Also, 6589 link changes occurred. The initial connectivity graph obtained on channel 3 is shown in figure 5.4(b). 3508 unidirectional links and 712 bidirectional ones were detected (ratio 4.93 : 1), with 5528 link changes occurring. This is nearly the same as the ratio obtained in the previous row of experiments. The average ratio seems to be between 4 and 5 to 1 for all experiments, even though individual values vary between 2.40 and 11 to 1.

5.3.4 Pole Experiments

The previous experiments were all conducted with sensor nodes that lay on the ground, which is a safe assumption for many deployments. However, if the nature of radio communication is taken into account, the nodes should be placed with a certain distance from the ground, to increase the communication range and reception. Therefore, the 36 sensor nodes were connected to wooden poles and placed about 20 cm above the university lawn in these experiments.



(a) Channel 0 (b) Channel 3

Figure 5.5: First Connectivity Graph obtained on the poles on channels 0 and 3 respectively

Figure 5.5(a) visualizes the first obtained connectivity graph for channel 0. Altogether 5150 unidirectional links and 492 bidirectional ones (ratio 10,47 : 1) with a total of 7146 changes were measured. Interestingly, the better radio characteristics increased the number of unidirectional links far more than the number of bidirectional ones. The ratio of unidirectional ones to bidirectional ones increased up to 18 :1.

The connectivity graph obtained at the start of the experiment on channel 3 is visualized in figure 5.5(b). Even though the figures seem quite different at first glance, the properties of the following 59 for each channel show that the basic connectivity characteristics are similar: Lots of unidirectional links, a few bidirectional ones and many link changes. Altogether 4761 unidirectional links and 225 bidirectional ones (ratio 21.61 : 1) with 5541 changes were measured.

5.3.5 Tree Experiment, Channel 0

To evaluate the connectivity at an even higher elevation, the sensor nodes were affixed to a five times five tree arrangement on the campus of our university. Please note that the absolute value for links does naturally decrease, as only 25 nodes are used in this scenario, instead of 36.

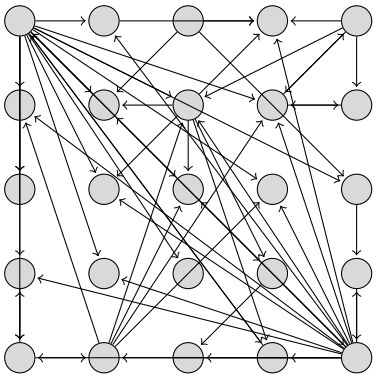


Figure 5.6: First Connectivity Graph obtained on the trees on channel 0

Figure 5.6 shows the initially measured connections. A total of 2977 unidirectional links and 330 bidirectional ones were measured (ratio 9.02 : 1) with 3329 link changes occurring during the experiment. While the increase in height caused better transmission characteristics, the trees made much larger obstacles than the poles, resulting in communication characteristics that were somewhat in between those measured in the placements on the ground and those on the poles.

5.3.6 Absence of Link Stability in all Environments

The connectivity measurements have shown that unidirectional links occur even more often than literature suggests, and confirmed that the height of the placement of nodes does influence the communication range as expected. More specific, the number of unidirectional links increases stronger than the number of bidirectional ones.

Figure 5.7 visualizes the results on the example of the lawn experiments on channels 0 and 3 in detail. Each round represents one flooded message, with one minute passing between rounds. The figure shows the number of unidirectional (U) and bidirectional (B) links as well as the number of changes between the previous round an the current one (C). Each change of a single link is counted separately, meaning that a unidirectional link that appears or disappears counts as one, a bidirectional one that turns unidirectional

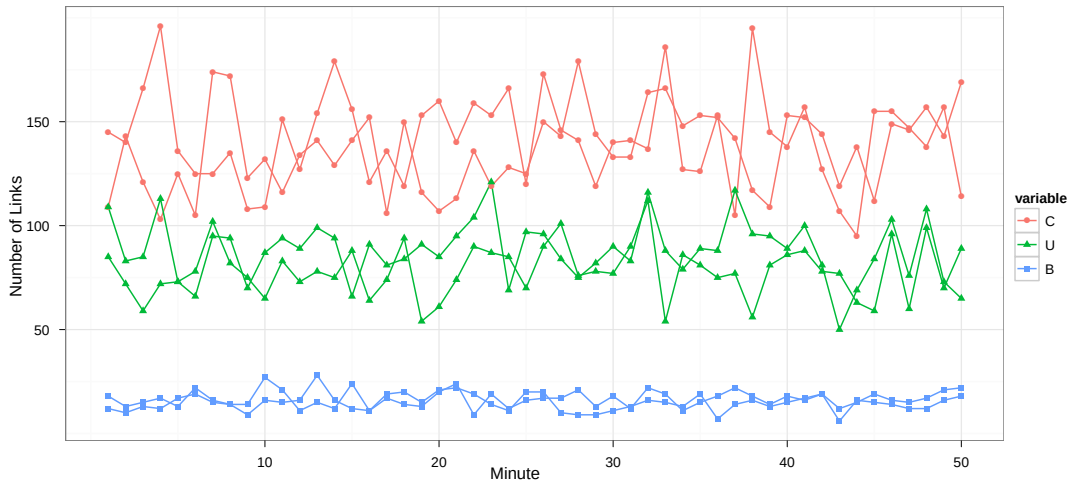


Figure 5.7: Measured Links on the lawn for Channels 0 and 3: (C)hanges, (U)nidirectional Links, (B)idirectional Links

is also counted as one but a bidirectional link that appears or disappears counts as two changes, one for each directed link contained therein. It can be seen that the number of link changes is often higher than the number of unidirectional links. This is due to the fact that when one unidirectional links disappeared and another appeared, two changes occurred. The ratio of unidirectional links compared to bidirectional ones changes frequently, but there are always far more unidirectional than bidirectional links present. The ratio varies between 3 to 1 and 91 to 1, with an average value of 8.69 to 1 over all presented experiments. This high number of unidirectional links supports the cause of this thesis, namely the necessity of using unidirectional links in routing protocols.

Figure 5.8 shows a box plot of the number of link changes per minute for each placement and channel. It can be seen that apart from the tree environment which only featured 25 nodes instead of 36, the number of changes seems to be always high, fairly independent of the environment and channel.

When considering the networks consisting of 36 nodes, an average number of 108 link changes per round (minute) can be recorded. This high number of link changes in a very short time makes it highly improbable, that a path which has been measured at one point in time will exist long enough to transmit a high number of application messages over this exact path. Other forwarding mechanisms which can react to such changes implicitly are required. Protocols should be able to react to these changes without

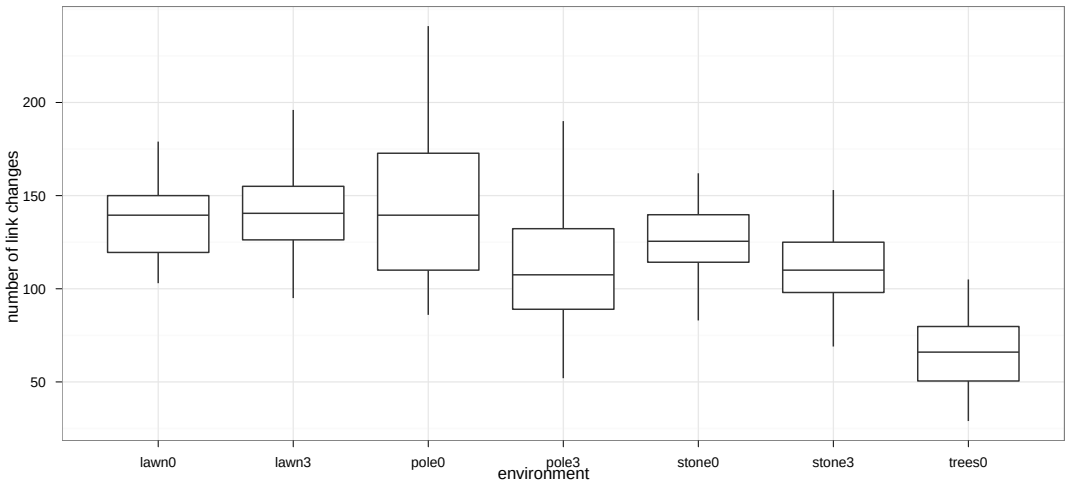


Figure 5.8: Box plot of average number of link changes per round for each location and channel

creating route error messages and restarting of route request flooding. The protocols developed in this thesis (see chapter 4) are good examples and should work well under the observed circumstances. The results obtained by evaluating them and the reference protocols, both in simulations as well as in the real world experiments, are presented in the next sections.

5.4 Application Scenario 1: Sense and Send

The application implemented for scenario 1 represents a sense-and-send behavior that is often found in sensor networks: All nodes within the network wanted to transmit all their messages to the same destination.

5.4.1 Simulation Results

The destination (sink) was fixed within a single simulation, but multiple simulations with different destinations were evaluated. For the network containing 100 nodes, all nodes in the upper left quadrant were chosen (25 destinations), for the network with 400 nodes this quadrant contained 100 nodes. Evaluating only one quadrant was chosen because of the symmetry of the network, and because of run time limits (a single simulation of `Flooding` in a network consisting of 1600 nodes took about 27 hours to complete). For the networks containing 900 and 1600 nodes a whole quadrant would have meant too many simulations, therefore only the 20 most interesting nodes (the corners and the middle of each quadrant) were chosen (20 destinations).

As 30 different connectivity change lists were used for each destination in each network size, 4950 simulations with run times between 5 minutes and more than a day were necessary for each protocol.

Buckshot Routing

Four different versions of Buckshot Routing were simulated, with two parameters that varied: The maximum route length allowed and whether or not caching of overheard routes was used. The variation in route length was evaluated for DSR as well (see below), because both protocols use source routing, meaning that the complete path has to be included in the messages, leading to pretty large messages. With a route length of 15 and a nodeIdType uint16, the header of a Buckshot Routing message can have a size of more than 30 Bytes. As a route reply in DSR contains two paths (source $\rightarrow$ destination and destination $\rightarrow$ source), the header size is already more than 60 Bytes when a route length of 15 is allowed. On the eZ430-Chronos (see appendix A) the radio hardware allowed only message sizes of 64 Bytes. Caching of overheard messages reduces the number of messages transmitted during route discovery drastically, and is especially important in the real experiments, where the network load affects the used MAC layer (see section 5.4.2).

Figure 5.9 shows the number of application messages that arrived at the sink in percent of the number of messages delivered by `Flooding` for four different versions of

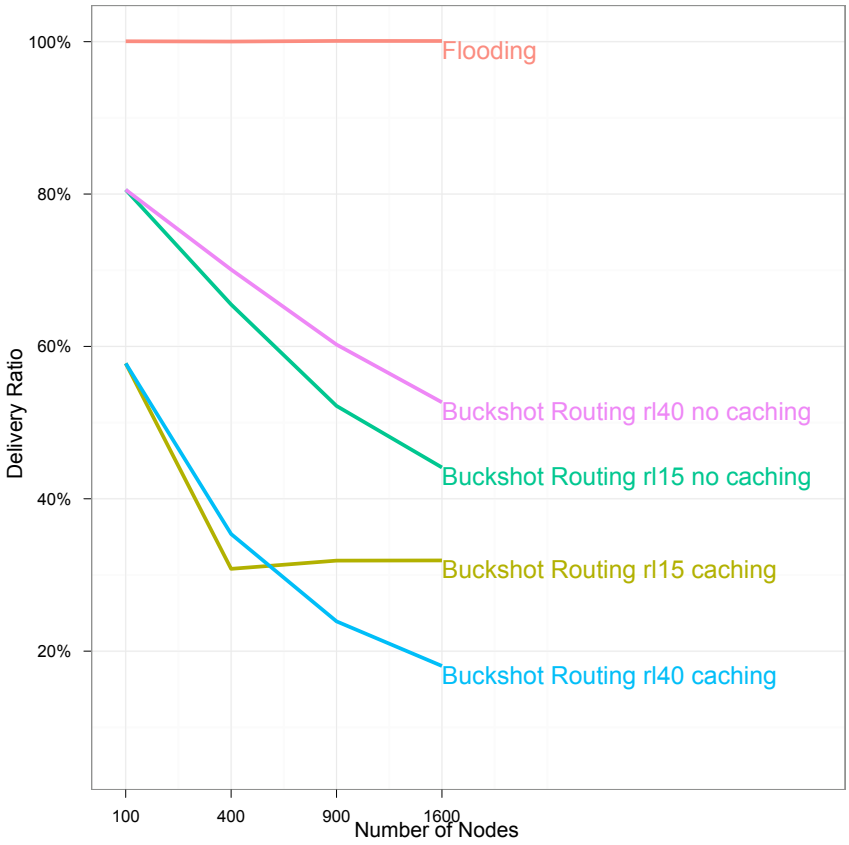


Figure 5.9: Delivery ratio of `Flooding` and four Buckshot Routing versions, Scenario 1

Buckshot Routing. It can be seen that the influence of the allowed route length was much smaller than the influence of caching of overheard routes. This is due to the sense-and-send application scenario used in the simulations: When Buckshot Routing transmits a data message, the path the message actually takes is collected on the way and replaces the message header. Therefore, the destination always learns the currently best path, and enters it into its routing table. When it transmits a message in the opposite direction, this newer, better path is used and the originator of the first message learns the path that has been taken just as the destination did. But in the evaluated scenario communication was only one way, the destination never transmitted messages back to the nodes (except for route replies). Therefore, a once overheard and cached bad path stays active throughout the simulation.

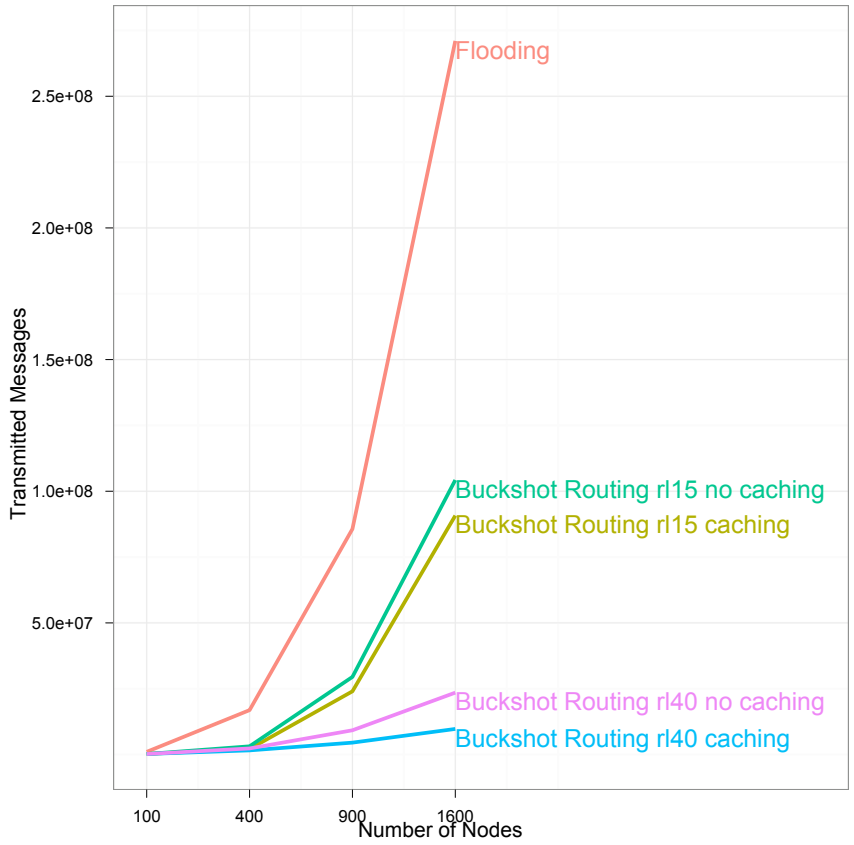


Figure 5.10: Number of transmitted Messages, `Flooding` and four Buckshot Routing versions, Scenario 1

The number of messages transmitted by each of the protocols can be seen in figure 5.10. The limit for the maximum route length has a strong influence on the number of messages transmitted once the networks get bigger. For networks with 20 times 20 nodes, a route length of 15 seems to be just about right. However, when the hop distance gets larger, many nodes do not find any route to the destination. This can be seen in the reoccurence of route discovery for every application message, which always leads to a new flooding (for 15 hops) of route request messages. As the hardware used in the real experiments dictates a maximum message size and therefore a maximum route length for source routing protocols, it also limits the possible network size.

The cost of delivering a single application message to the sink measured in transmitted messages is depicted in figure 5.11. The figure shows that even though the version

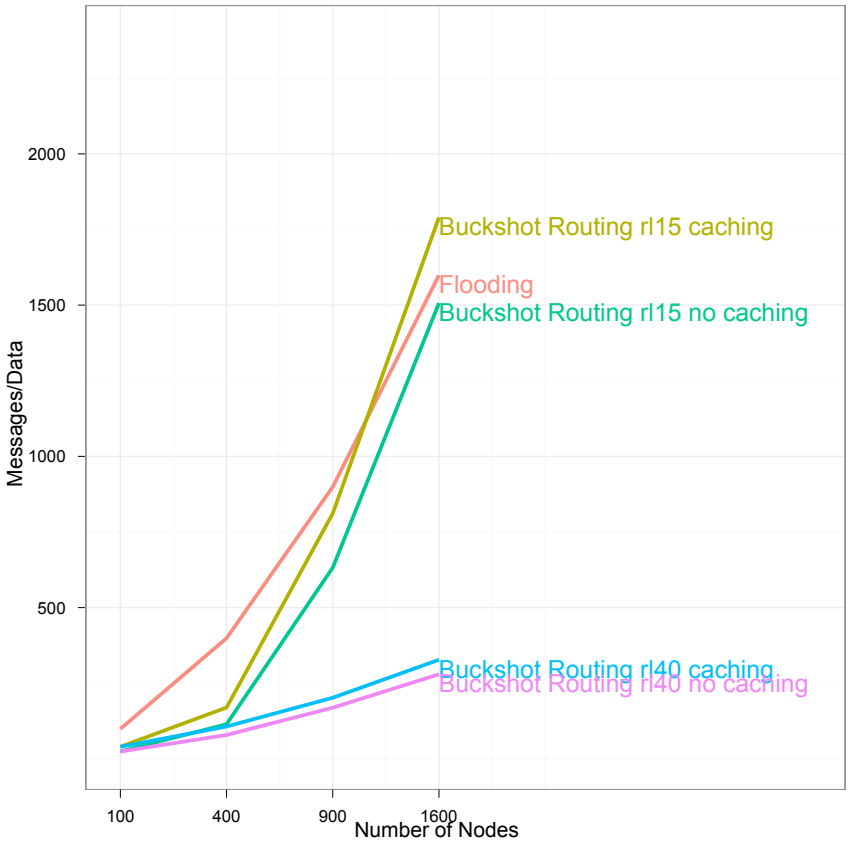


Figure 5.11: Number of transmitted Messages to deliver a single data message to the sink, `Flooding` and four Buckshot Routing versions, Scenario 1

of Buckshot Routing that uses caching and route length 40 delivered the least number of messages, its ratio is fairly good. This is due to the large number of route request floodings that can be spared when caching of overheard routes is used and will become important later on in the real world experiments.

Related Work Protocols

The number of data messages received at the sink for the reference protocols is shown in figure 5.12. It can be seen that none of the other protocols gets anywhere near the performance of `Flooding`, with DSR performing worst. Even in the smallest network consisting of 10 times 10 nodes, both DSR versions deliver only about 10% of the messages.

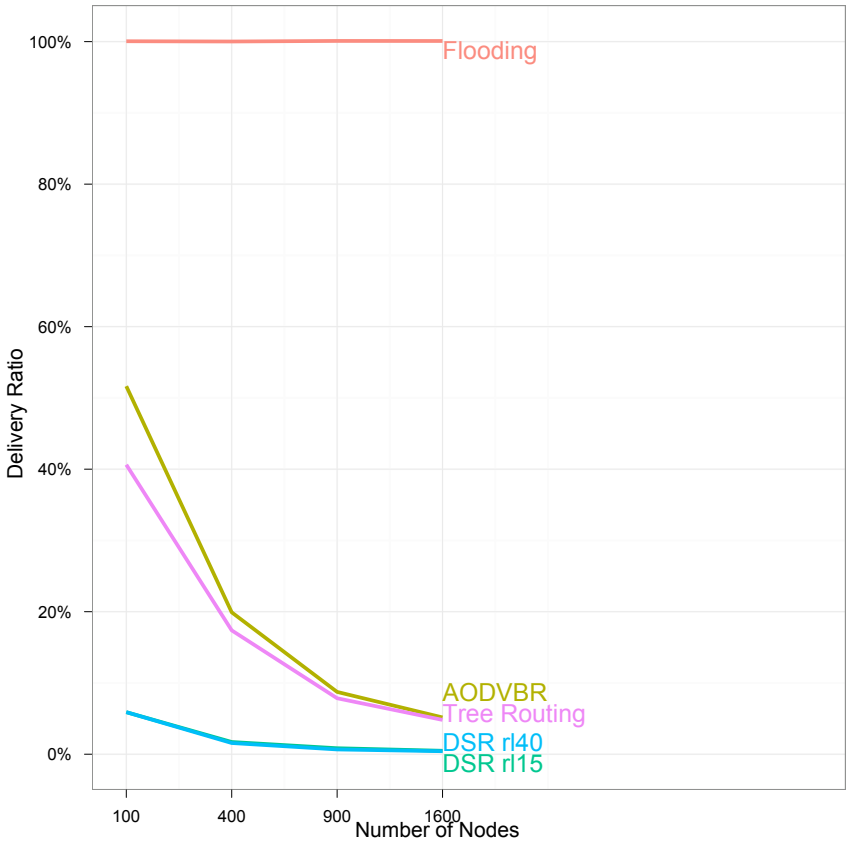


Figure 5.12: Delivery ratio of AODV-BR, `Tree Routing` and two DSR versions, Scenario 1

The other two protocols perform better in the small network, but show a steep decline in delivery ratio for the larger networks. This decline is due to the fact that even though the number of nodes in the network and therefore the maximum possible number of delivered messages increases drastically, the total number of delivered messages increases only marginally.

The absolute number of messages received at the sink for the two versions of DSR is shown in figure 5.13. It can be seen that DSR delivers a nearly constant number of messages, independent of the network size. While the number of nodes and thus the number of application messages handed to the routing protocol is multiplied by 16, the number of application messages that arrive at the sink increases only marginally. This is due to the fact that DSR suffers heavily from link changes and longer routes change

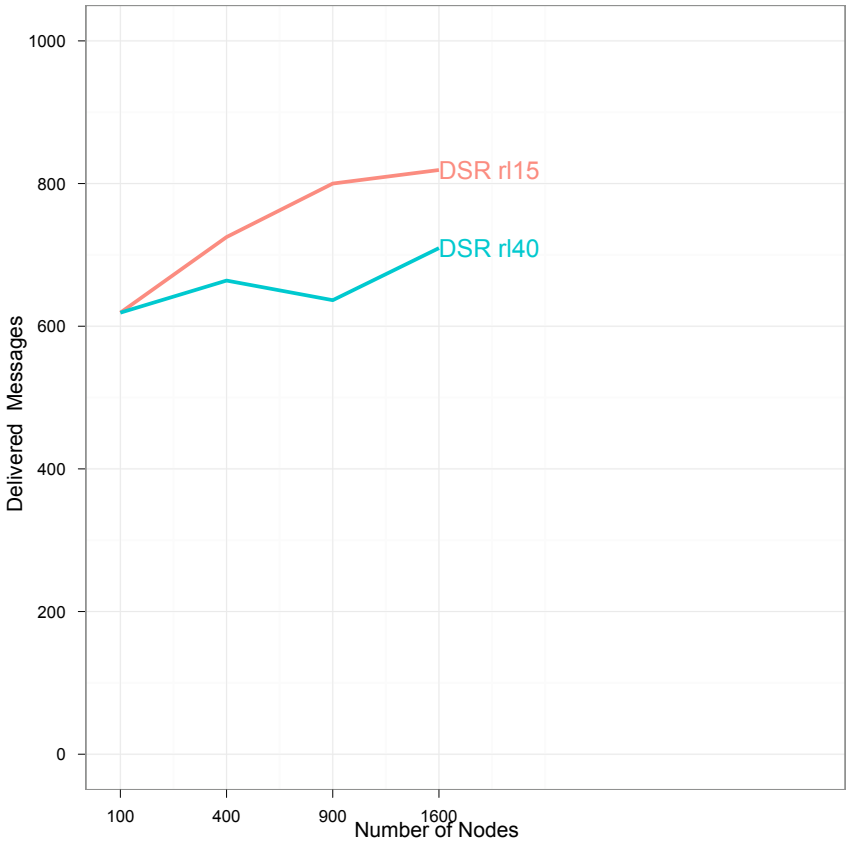


Figure 5.13: Total number of Data messages at the sink, two DSR versions, Scenario 1

more often. Another interesting fact about DSR is that the version with route length limited to 15 delivers more messages than the one which allowed route lengths up to 40 hops for all larger networks. The reason for this seemingly strange behavior can be seen when investigating nodes that are about 15 to 17 hops from the sink. Please remember that the hop distance changes as links change. Therefore, nodes might have a distance of more than 15 during their first route discovery, and less during a later one. When only short routes are allowed and no route is found, the messages are stored until a later route discovery finds a route containing 15 hops at max. Then, all stored messages are transmitted at once. These messages have a higher chance of being delivered, as the route information is current and the path is shorter. When using the 40 hop limit, these nodes choose the first, long path that is found. But longer paths have a higher probability of message loss, leading to fewer messages being delivered in total.

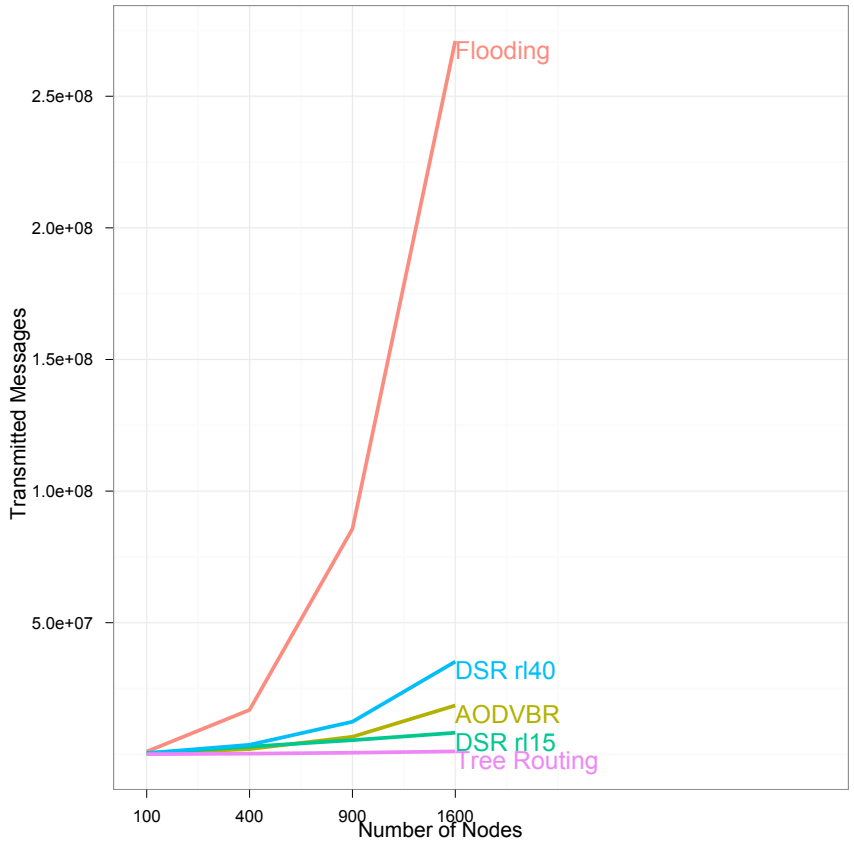


Figure 5.14: Number of transmitted Messages, `Flooding`, AODV-BR, `Tree Routing` and two DSR versions, Scenario 1

The total number of messages transmitted by each of the protocols chosen for comparison is shown in figure 5.14. `Flooding` naturally transmitted the most messages by far. Also, it can be seen that `Tree Routing` transmitted very few messages, and DSR with route length 40 transmitted much more messages than the version with route length 15.

The number of messages transmitted in order to bring a single application message to the sink is shown in figure 5.15. Even though DSR with route length 40 delivered nearly the same amount of data as DSR with route length 15, the high number of transmitted messages makes it the most costly related work protocol by far. Interestingly, even though it transmits a large number of messages, the high number of delivered messages make `Flooding` the second best. Only `Tree Routing` performs better. This is due to

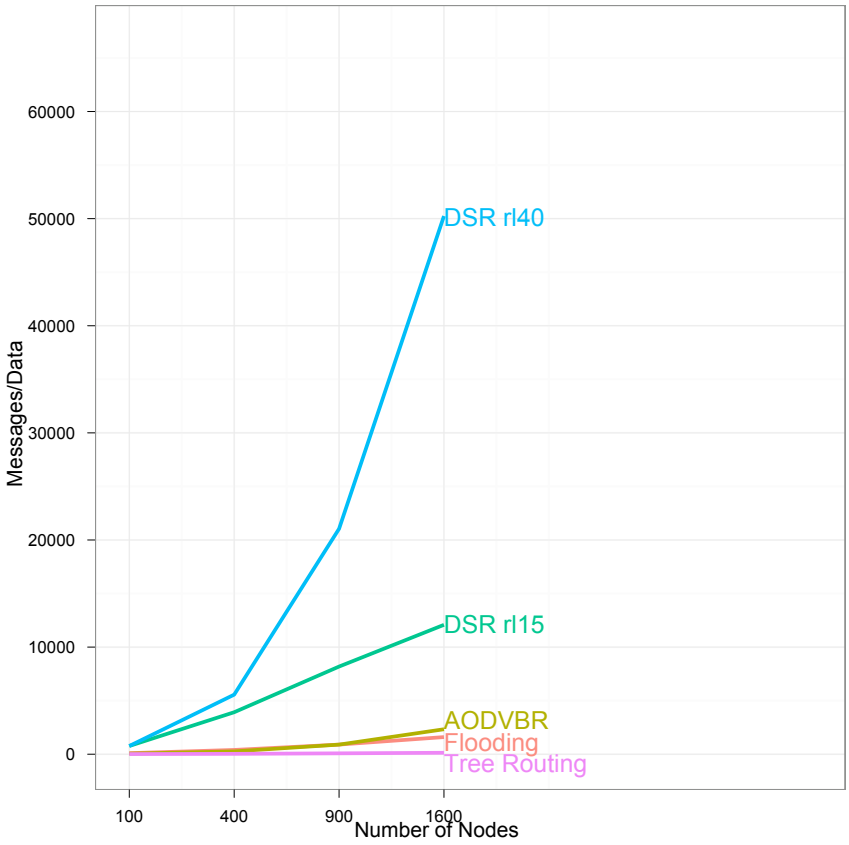


Figure 5.15: Number of Messages transmitted to deliver a single application message, `Flooding`, AODV-BR, `Tree Routing` and two DSR versions, Scenario 1

the fact that `Tree Routing` has very low costs for delivery failures. Nodes close to the sink are often able to deliver their messages. Nodes that are farther apart transmit their messages, and try two retransmissions if the message is not forwarded by the next hop. But, contrary to the other protocols, no route error messages are generated and no new route discovery is initiated when the retransmissions are not successful.

BuckshotDV, OSBRDV, ULC and ULTR

The delivery ratio of the four other protocols developed in this work, BuckshotDV, OSBRDV, ULC and ULTR is compared to `Flooding` in figure 5.16. Note that the scale starts at 80%.

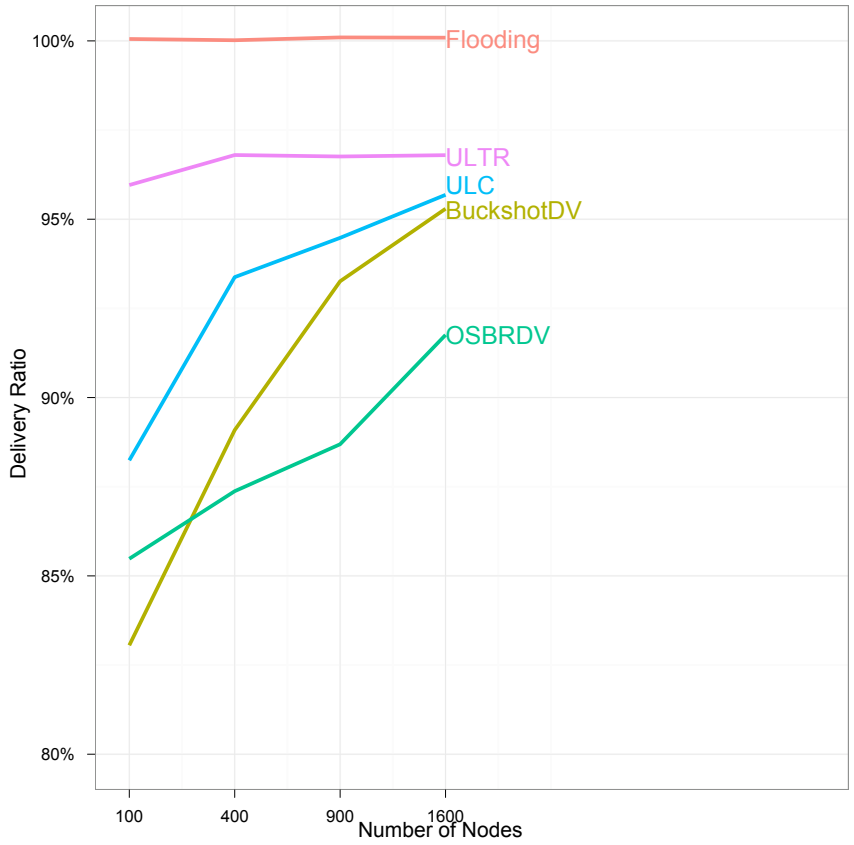


Figure 5.16: Delivery Ratio of BuckshotDV, OSBRDV, ULC, ULTR and `Flooding`, scale starts at 80%, Scenario 1

What catches the eye right away on the figure is that the delivery ratio of all four protocols is very high and increases with network size. ULTR seems to have reached its maximum at 97% already in networks consisting of 20 times 20 nodes, while the other protocols still get better, more or less gradually. Also, it seems that ULC and BuckshotDV converge to the same value of about 97%, but to be sure more simulations with larger networks would be necessary. These were not done for this thesis for two reasons: First, the simulation run time would be very high. A single simulation of `Flooding` in the 40 times 40 network took more than a day, and 600 of these were necessary. In 50 times 50 networks the value would be much higher. Second, the largest network that was simulated, 40 times 40, already contains 1600 nodes and it is unlikely that such large sensor networks will be deployed for a real application in the

near future. If larger networks are deployed, it is likely that a logical partitioning of the network would be realized on application level, and multiple sinks would be used.

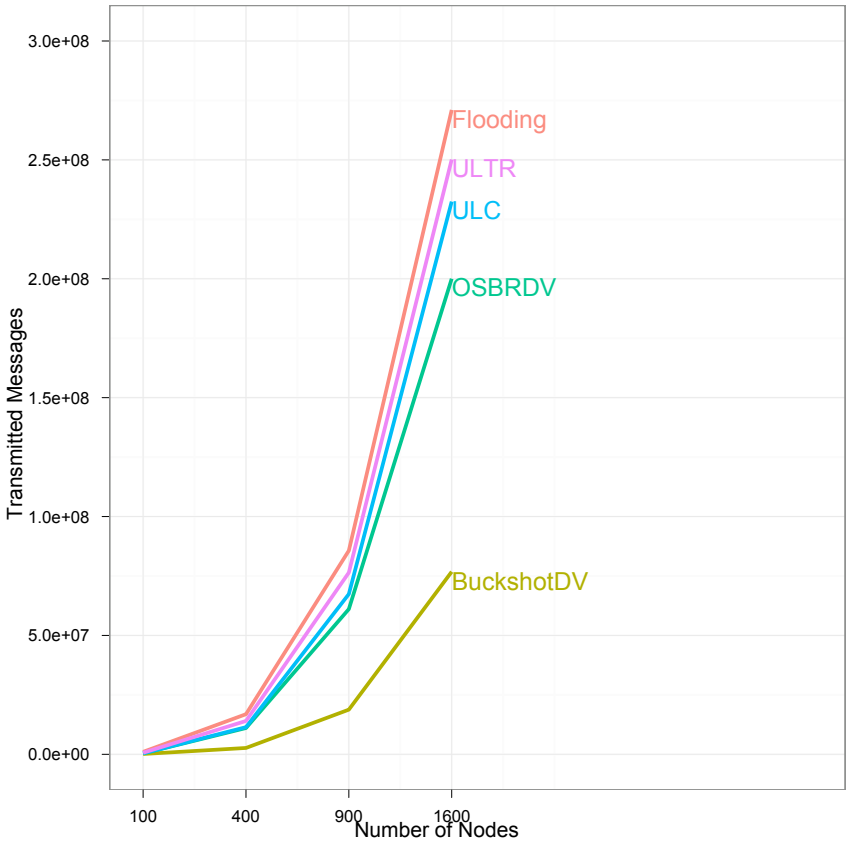


Figure 5.17: Number of Messages transmitted by BuckshotDV, OSBRDV, ULC, ULTR and `Flooding`, Scenario 1

The number of messages transmitted by BuckshotDV, OSBRDV, ULC, ULTR and `Flooding` is shown in figure 5.17. The figure shows that ULC and ULTR need a lot of message transmissions to compensate for the missing neighborhood discovery protocols: As both protocols were designed with the assumption that either a neighborhood discovery protocol or the used MAC layer would supply link information, they suffered from the absence of accurate information. The passive overhearing that was implemented instead can only detect bidirectional and unidirectional incoming links, which makes the explicit usage of unidirectional links all but impossible. Therefore, both protocols try to find bidirectional links, or, if these are not available, switch to buckshot mode

for one hop, which increases the network load very much, when it is initiated too often. Another problem for these protocols was timing: Passive detection of links only works when messages are transmitted, but links change more often than messages are transmitted. Therefore, both protocols often worked on outdated information.

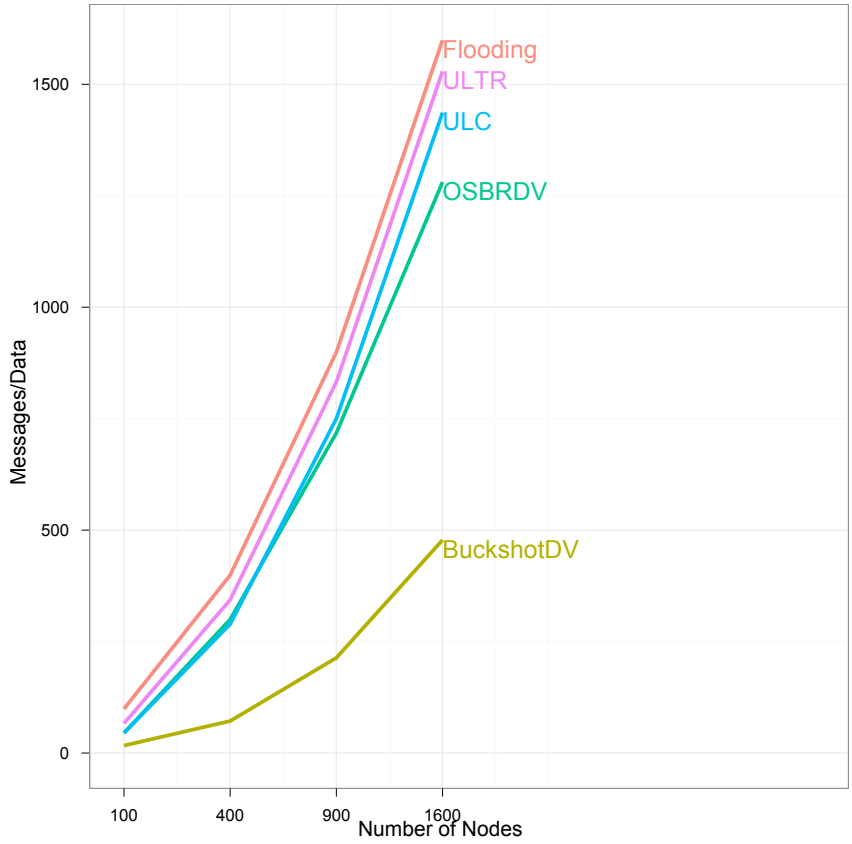


Figure 5.18: Number of Messages transmitted to deliver a single application message, BuckshotDV, OSBRDV, ULC, ULTR and `Flooding, Scenario 1`

Much more surprising is the performance of OSBRDV, which was designed to reduce number of messages transmitted by BuckshotDV by storing messages and maybe discarding them if their forwarding by the next hop is overheard. Obviously, this did not work, as OSBRDV transmitted more than twice as many messages as BuckshotDV. The reason for this increase in number of transmitted messages lies in the highly dynamic network topology and the delayed transmissions. Due to the high number of unidirectional links, nodes often did not overhear the forwarding of their message by

the intended next hop and forwarded the message after the timeout. But in the meantime, the logical topology of the network had changed, and additional nodes received the message, once again storing it and forwarding after a certain delay.

The costs that the delivery of a single data message caused are shown in figure 5.18. ULTR produces almost the same costs as `Flooding` with more than 1500 messages in the network consisting of 1600 nodes. It is closely followed by ULC, as both protocols were subject to the same problems. The limited directional flooding used in BuckshotDV seems to work, though. It needed to transmit the lowest number of messages per application message by far.

Comparison between all Protocols

Concluding the evaluation of these simulations it can be said that all protocols developed for this thesis have achieved a much better delivery ratio than the protocols used for comparison. Only `Flooding` delivered more messages, which is why it was used as reference, and the delivery ratio of a protocol defined as the number of messages delivered by that protocol divided by the number of messages delivered by `Flooding` (figure 5.19).

Even though the simulations did not feature MAC layer elements, it can already be seen that the protocols chosen from related work are not able to work in an environment with many unidirectional links and often changing links in general. On the other hand, the results clearly show that the developed protocols have achieved their design goals, namely resistance against often changing links and usage of unidirectional links. Only `Flooding` delivered better results in the simulations, and it is known that `Flooding` runs into huge MAC layer problems when it is used on real hardware. Moreover, the protocols developed in this thesis work better in larger networks, except for the source routing variant of Buckshot Routing. But source routing should only be used in small networks anyway, due to the resulting header size.

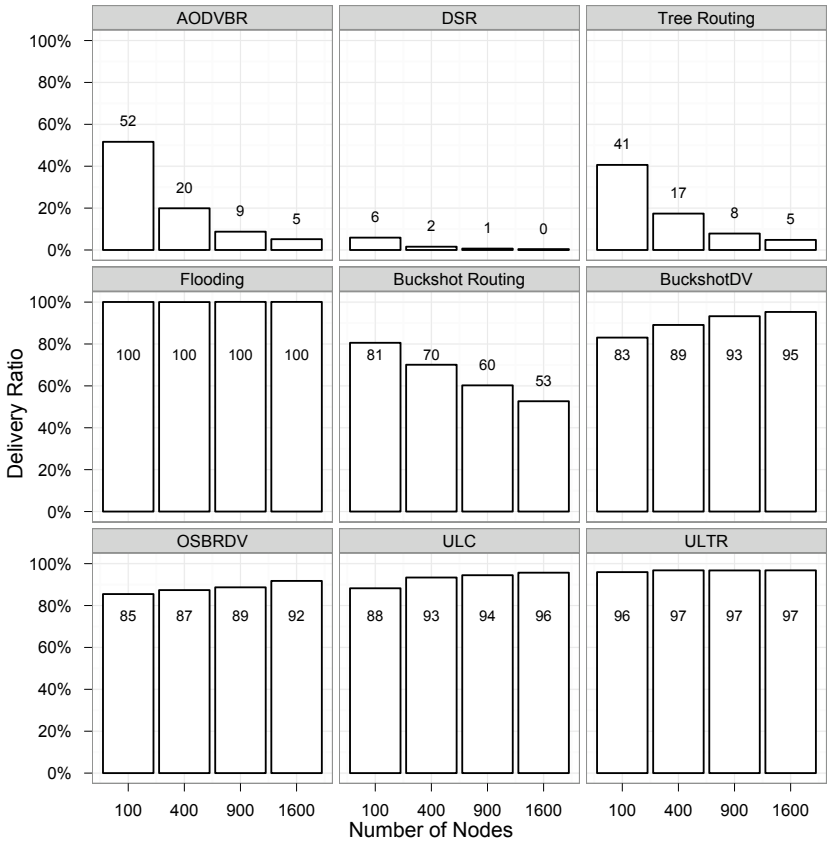


Figure 5.19: Delivery Ratio of all Protocols for different Network Sizes, Scenario 1

5.4.2 Real World Experiment Results

For the real world experiments of scenario 1, all four different locations described in section 5.2.4 were used. Each protocol was evaluated on each location, with node 0 in the lower left corner as destination (sink).

The delivery ratio of each protocol is shown in figure 5.20, sorted by protocol and location. For most protocols, the number of delivered messages for the desk and pole locations is roughly the same, as these two locations differed only marginally. The desk location is one hop, while the pole location contained between one and two hops on average. The figure also shows that `Flooding` delivers a nearly constant number of messages for the pole, lawn and stone environments.

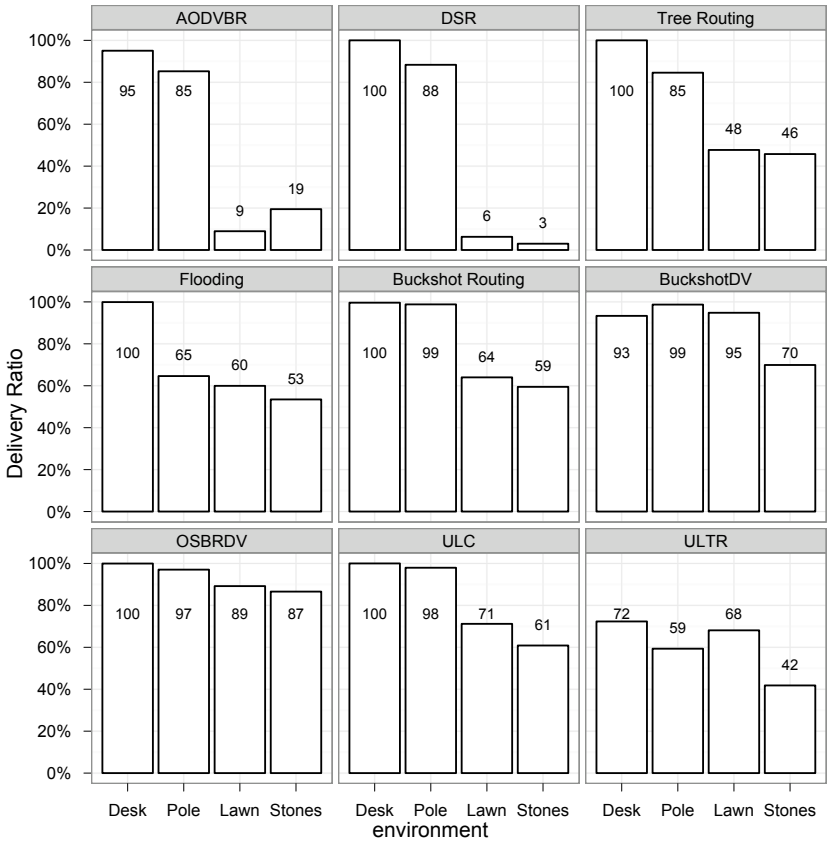


Figure 5.20: Delivery Ratio of each Protocol achieved in the real experiments, Scenario 1

The other three reference protocols, AODV-BR, DSR and `Tree Routing` show a steep decline in delivered messages for the lawn and stone pavement placements. All protocols developed for this thesis were able to deliver more messages to the destination, except for ULTR on the stones. The reason for the bad results from ULTR lies in its dependency on accurate link information. As described in section 4.4, ULTR tries to route messages around unidirectional links explicitly. But in order to build this triangle, neighborhood information is needed. The current implementation of ULTR tries to obtain this information passively, by overhearing forwarded messages. For a rapidly changing environment this approach is bound to fail. It would be interesting to see, how a protocol implementation that uses a neighborhood discovery protocol or neighborhood information provided by the MAC-layer would perform.

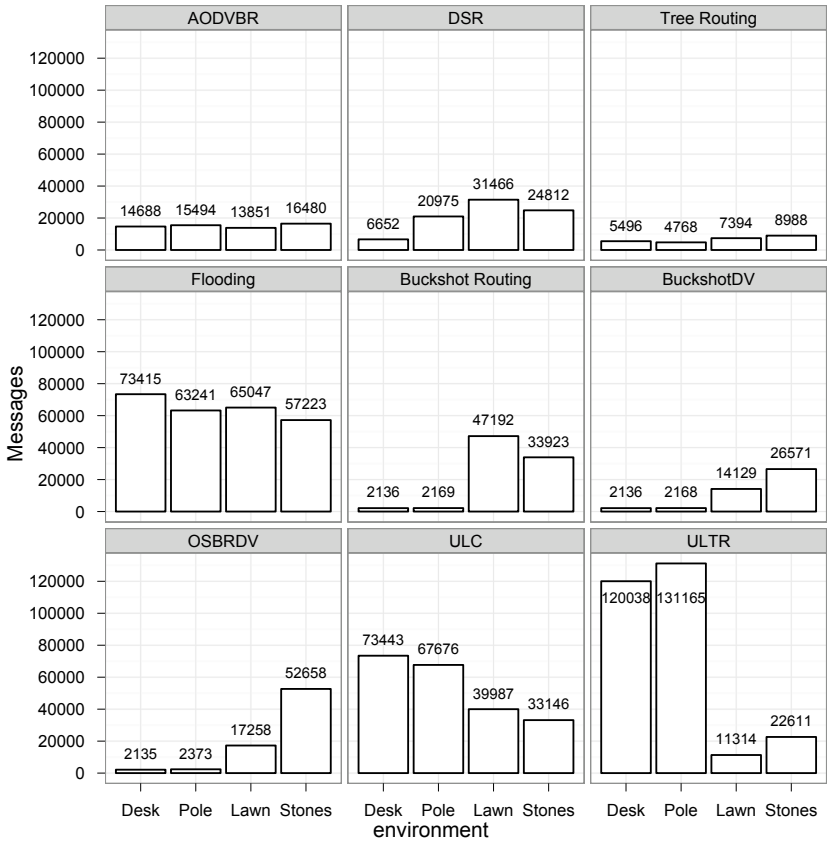


Figure 5.21: Total Number of Messages transmitted by each Protocol, Scenario 1

The number of protocol and data messages transmitted by each protocol can be seen in figure 5.21. Once again, `Flooding` remains fairly stable throughout the locations. While all other protocols transmitted more messages in the last two locations (lawn, stone pavement), the number of messages transmitted by ULC and ULTR declines. This is once more due to the absence of accurate link information. Both protocols were designed with the assumption that link information would be available either from a neighborhood discovery protocol or from the MAC layer. Using only overheard messages instead does not work in the first two locations: When all nodes can transmit directly to the sink and the sink never answers, all links are assumed to be unidirectional and buckshot mode is induced. But when buckshot mode is started by every node, the network load rises close to that of flooding. This can be seen by taking a closer look at the type of transmitted messages: A lot of the transmitted messages were data messages.

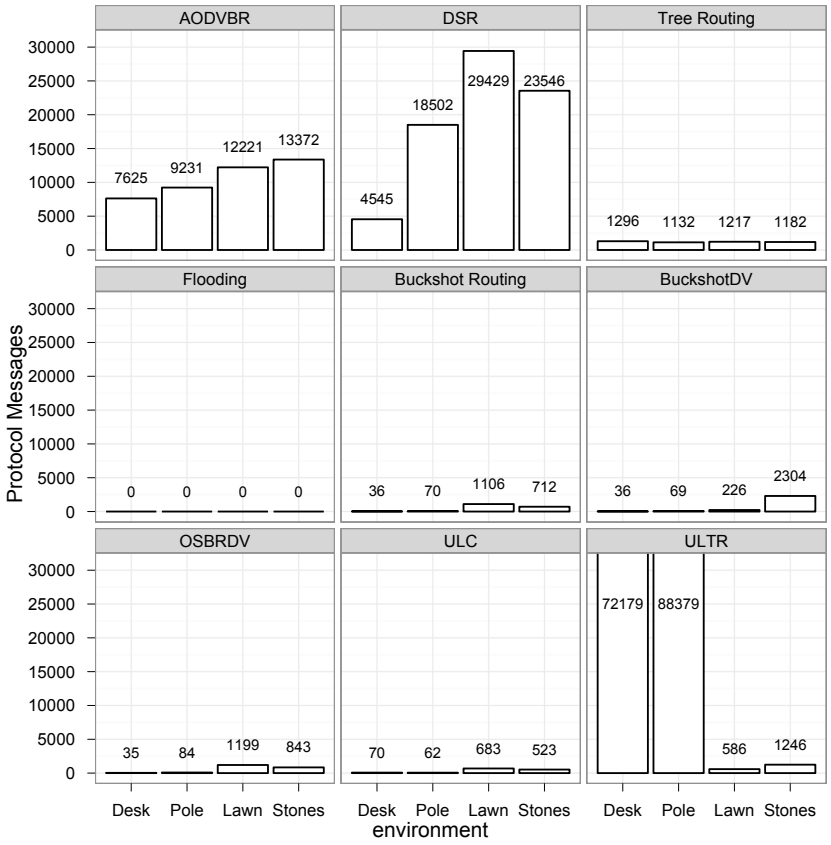


Figure 5.22: Number of Protocol Messages transmitted by each Protocol, Scenario 1

An even more interesting fact that can be seen in the figure is that the passive neighborhood discovery mechanism starts to work in the multihop environments. When paths are more than 1-2 hops in length, forwarded messages are received more often and the nature of the links can be observed. Therefore, even though it might seem strange, ULC and ULTR need to transmit fewer messages in networks with a larger diameter.

The number of protocol messages, i.e. non-data messages, transmitted by each protocol can be seen in figure 5.22. As already seen above, ULTR suffered badly from the missing neighborhood discovery protocol. Where ULC only incremented a counter for unidirectional links, ULTR always switched to buckshot mode. The huge number of protocol messages transmitted by ULTR consisted mainly of route request messages. In fact, a route discovery took place for nearly each data message generated by the application in ULTR.

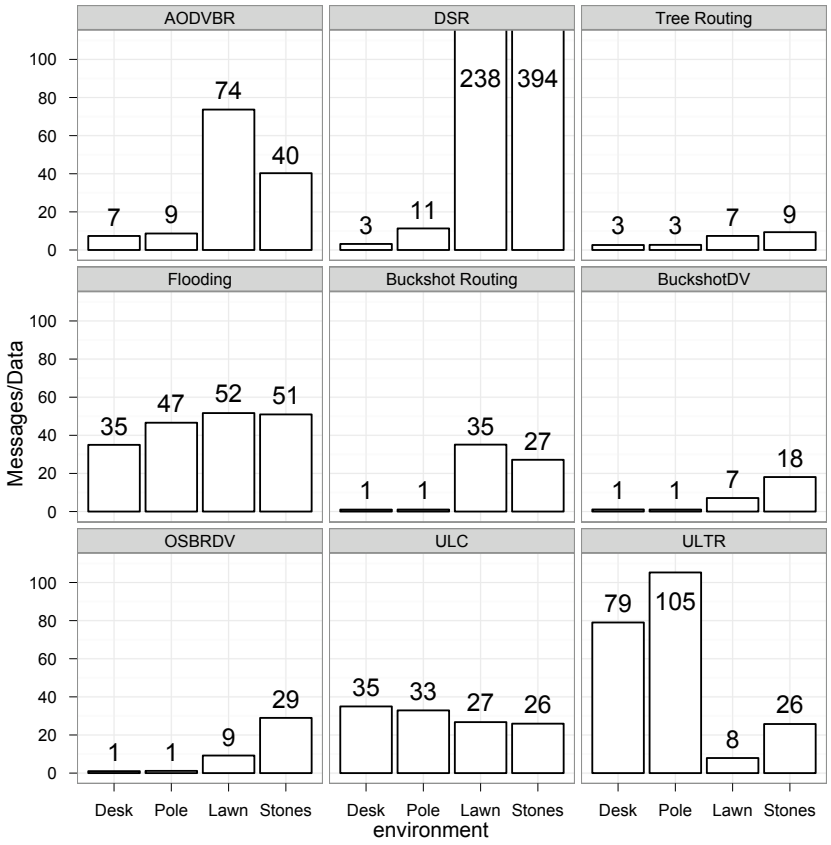


Figure 5.23: Total Number of Messages transmitted by each Protocol divided by the number of delivered data messages, Scenario 1

Another measurement of the cost paid to deliver an application message is shown in figure 5.23. The figure shows the total number of transmitted messages divided by the number of application messages that reached the sink. Unsurprisingly, `Flooding` once more shows a relatively constant performance and DSR and AODV-BR show too high values. Interestingly, `Tree Routing` seems to have performed quite well. If only this figure would be taken into account when choosing a protocol, `Tree Routing` would be preferred. However, the numbers presented here have to be put into perspective. The result of `Tree Routing` is achieved because it uses nearly no protocol messages, and two retransmissions are its only reaction to message loss. No route error messages are generated and no new route discovery is started. Therefore, the cost of a lost application message is much lower than in the other protocols. DSR represents the other end of

the spectrum: When a message loss is detected there, a route error message is created and transmitted to the originator. When the route error is received, the route is deleted and a new route discovery is initiated, which leads to a flooding of the whole network. If conditions are really bad, it may even lead to flooding the network twice. Except for the problems experienced by ULTR in the 1-2 hop locations, the protocols developed in this thesis perform fairly well. But the results also show that the number of nodes used in the real world experiments was actually a little low - as the simulations have shown (see section 5.4.1), the big differences between protocols can be seen better in larger networks. However, using a few hundred nodes in the real world experiments was not possible as there were not that many nodes available.

5.4.3 Comparison between Simulations and Experiments

The real world experiments were conducted with 36 nodes, while the simulations featured either 100, 400, 900 or 1600 nodes. To show that the tendencies seen in the simulations represent those that would be achieved with a large scale sensor network, a network consisting of 36 nodes was also simulated, using the simulation parameters specified in appendix A.

Figure 5.24 shows the median of the delivery ratio of all evaluated protocols for the two multihop experiments (lawn, stones) and the 36 nodes simulation. Naturally, the results of `Flooding` in the simulation are much better than those achieved in the real world experiments, as `Flooding` suffers heavily from the broadcast storm problem in the real experiments. The used CSMA MAC layer simply cannot handle the huge number of messages. Except for ULTR, the simulation results and those of the two experiment settings are quite similar for the protocols developed in this thesis. From the protocols used for comparison, only AODV-BR shows a large difference between simulation and real world results.

When looking at the results those two protocols, AODV-BR and ULTR, achieved in the real world experiments, it can be seen that they have a strong variation in delivery ratio between the lawn and stone experiments. This high variation seems to imply that both protocols are especially vulnerable to one or more properties of the real experiments which do not have so much influence on the other protocols.

Both AODV-BR and ULTR try to use an explicit detour around unidirectional links, using link information detected during route reply transmission (AODV-BR) or during transmission of DATA messages (ULTR). As the connectivity measurements have shown, link changes occurred even more often than expected, making conditions for AODV-BR and ULTR harder in the real world than in the simulations. None of the other protocols

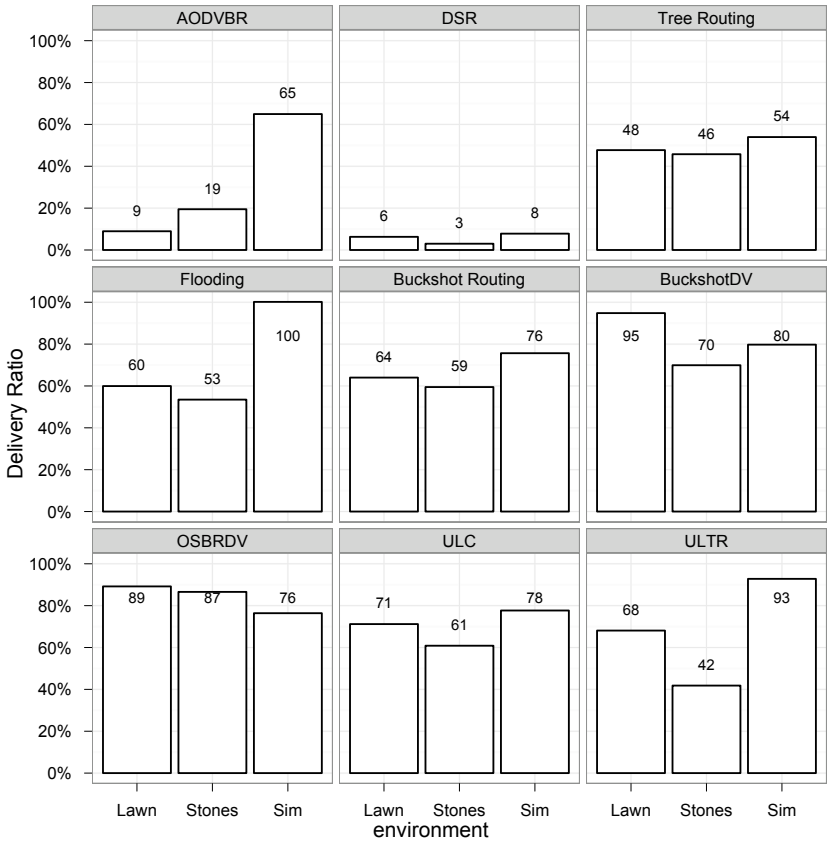


Figure 5.24: Delivery Ratio of each Protocol; Experiments vs. Simulations

suffered as much as these two. For DSR, an additional increase in frequency of changes made no difference, as it could not even tolerate the one simulated. For `Tree Routing`, the small network diameter and the 2 retransmissions on each hop were enough to deliver about 50% of application messages. The link changes would not have influenced `Flooding`, but `Flooding` produced a very high network load which the MAC layer could not handle. ULC worked on incorrect information, but did not suffer as strongly from this due to the fact that only the calculation of the weights might be wrong, but routes could still be found. For Buckshot Routing, BuckshotDV and OSBRDV a mistake in their neighbor table only resulted in unnecessary transmissions, but did not lessen the delivery ratio much. Still, it can also be seen that the deployed sensor network was not large enough for them to show their full potential.

In summary it can be said that the used simulation approach has some limitations, as it does not include the medium access control protocol used in the real experiments. However, the results show that the usage of connectivity matrices and the way they were generated is close to reality, and can be used to evaluate the influence of unidirectional links and frequent link changes on the routing protocols. This is exactly what the simulations were intended for as the used MAC layer and other side effects of the used hardware might (and hopefully will) change for future deployments. When the exact properties of the hardware that will be used in a deployment are known beforehand, these could be included in the simulations, but that was not the case for this thesis. Some of the less favorable communication properties of the eZ430-Chronos (e.g. the inability of the CCA to receive messages during backoff) were only discovered during the connectivity measurements (see section 5.3).

Another advantage of the developed simulation model is the fact that the connectivity data gathered during the connectivity measurement experiments can easily be included. The data that could be gathered this way was not presented in this work, because the number of data sets from the connectivity experiments currently available is too small.

5.5 Application Scenario 2: Single Pairing

In this scenario all settings, including the number of messages a node wants to transmit, are the same as in the sense-and-send scenario. However, instead of a single sink as destination for all messages from all nodes, each node has a randomly chosen partner node it wants to communicate with. This pairing of nodes was generated before the simulations and experiments, and differs only between different network sizes: If e.g. node 15 is the partner of node 21 for the network consisting of 36 nodes, this pairing remains fixed for all protocols as well as for simulations and real world experiments.

This pairing of nodes represents a communication pattern for MANETs and was chosen because two of the protocols used for comparison (AODVBR and DSR) are MANET protocols.

5.5.1 Simulation results

In the simulations for the single pairing scenario, the same connectivity change lists were used that have already been used in the sense-and-send scenario. However, as the destination was not a single fixed one for all nodes, the simulations were not varied according to the destination. Instead, the generated pairings were used as stated above.

Flooding was once again used to measure the upper limit for delivered messages and the delivery ratio was defined as the number of messages delivered by a protocol divided by the number of message delivered by `Flooding`.

Buckshot Routing and Related Work Protocols

The delivery ratio of AODVBR, Buckshot Routing, DSR, `Flooding` and `Tree Routing` is shown in figure 5.25. For all protocols except `Flooding` the delivery ratio declines with increasing number of nodes. It can be seen that AODVBR and `Tree Routing` suffer the most from the increased route length in the larger networks, as the decline of their delivery ratio is steep. For AODVBR, building the initial route is the crucial part. When a route has been successfully established, the fish bone structure can be used to salvage data packets. But since building the initial route requires a bidirectional path and the probability of a complete path being bidirectional decreases with route length, AODVBR only works in small networks. For `Tree Routing`, building the initial route is no problem. However, due to the dynamic nature of links between nodes, the initial path is obsolete soon and the two retransmissions used as reaction to message loss are not sufficient in larger networks.

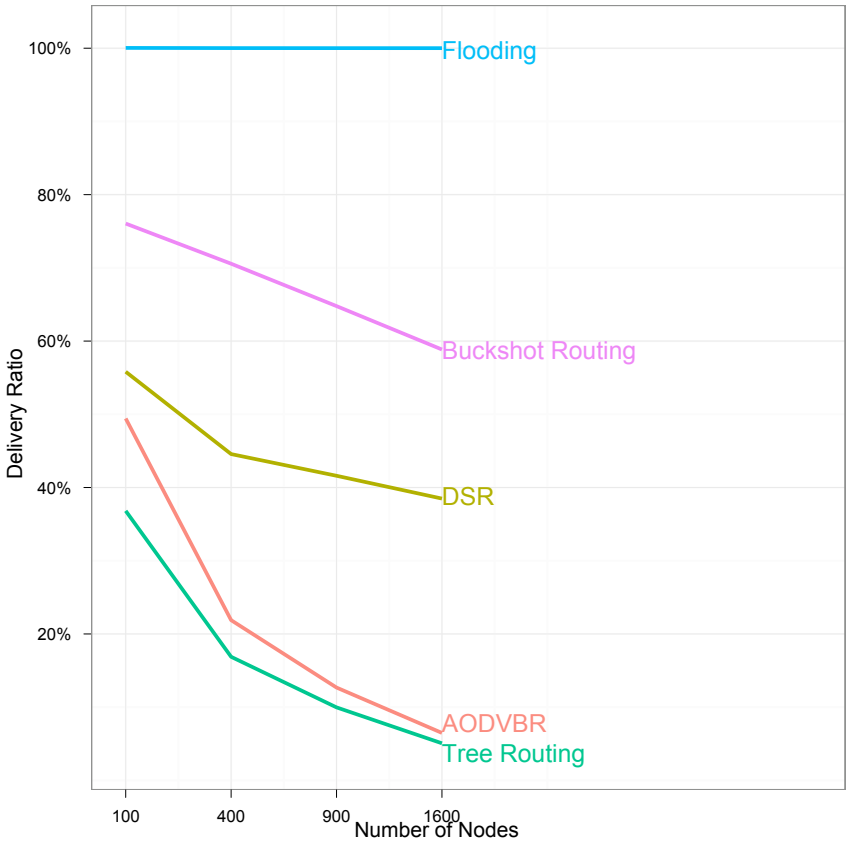


Figure 5.25: Delivery ratio of AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 2

The delivery ratio of DSR and Buckshot Routing also declines due to their source routing nature. However, finding the initial route is not a problem for either of them, as DSR uses one flooding for each direction and Buckshot Routing uses multiple paths implicitly. The main difference between both protocols are their route maintenance mechanisms. When DSR detects a route break it tries to inform the originator of the message that caused the detection of the break. Following this, a new route discovery with all its costs takes place. In Buckshot Routing this route maintenance is done implicitly with each received message, resulting in fewer stale routes and a better delivery ratio. Also, a maximum route length of 40 and caching of overheard routes were used for Buckshot Routing and DSR in this scenario. When the delivery ratio of Buckshot Routing for this scenario is compared to that achieved by the same variant in the sense-

and-send scenario it can be seen that the delivery ratio has increased from less than 20% to nearly 60% in the networks containing 1600 nodes. This is due to the fact that the implicit route maintenance did not work in the sense-and-send scenario as all messages were transmitted from the nodes to the sink. As the nodes never received replies from the sink, they could never use the implicit route maintenance mechanism. In this scenario however, the pairing of nodes results in a constant message exchange between a node and its partner, leading to routes that are up to date most of the time.

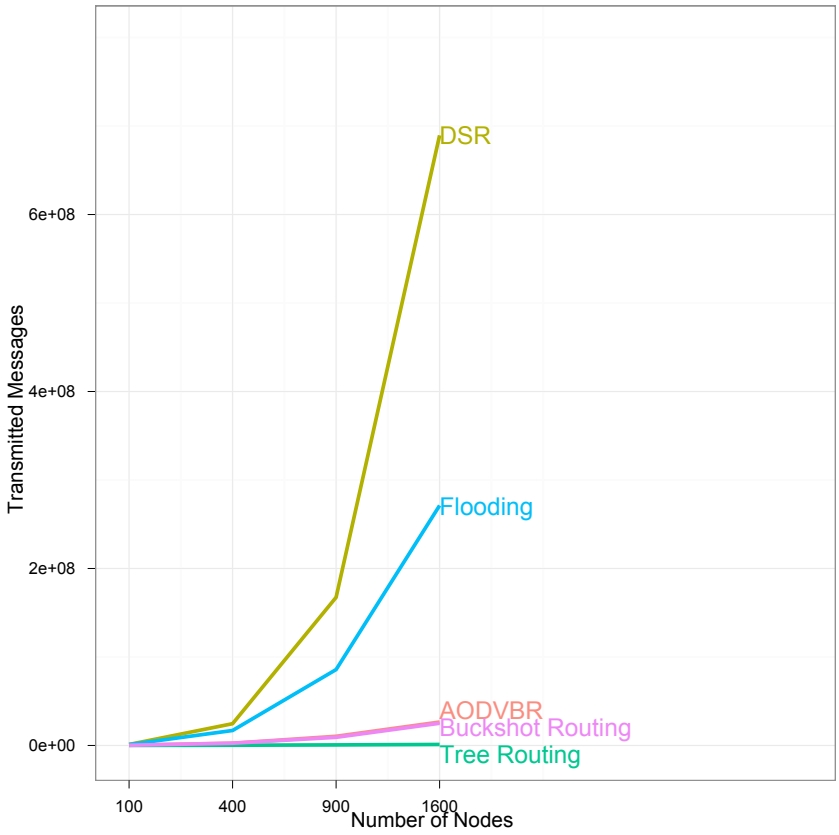


Figure 5.26: Number of transmitted Messages, AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 2

The number of transmitted messages for each protocol is shown in figure 5.26. Here, the impact of the route maintenance mechanism of DSR can be seen: It transmits more than twice as many messages as `Flooding` as it tries to repair broken routes. `Tree Routing` presents the other extreme, it transmits nearly no messages at all, while Buckshot Routing and AODVBR need slightly more messages.

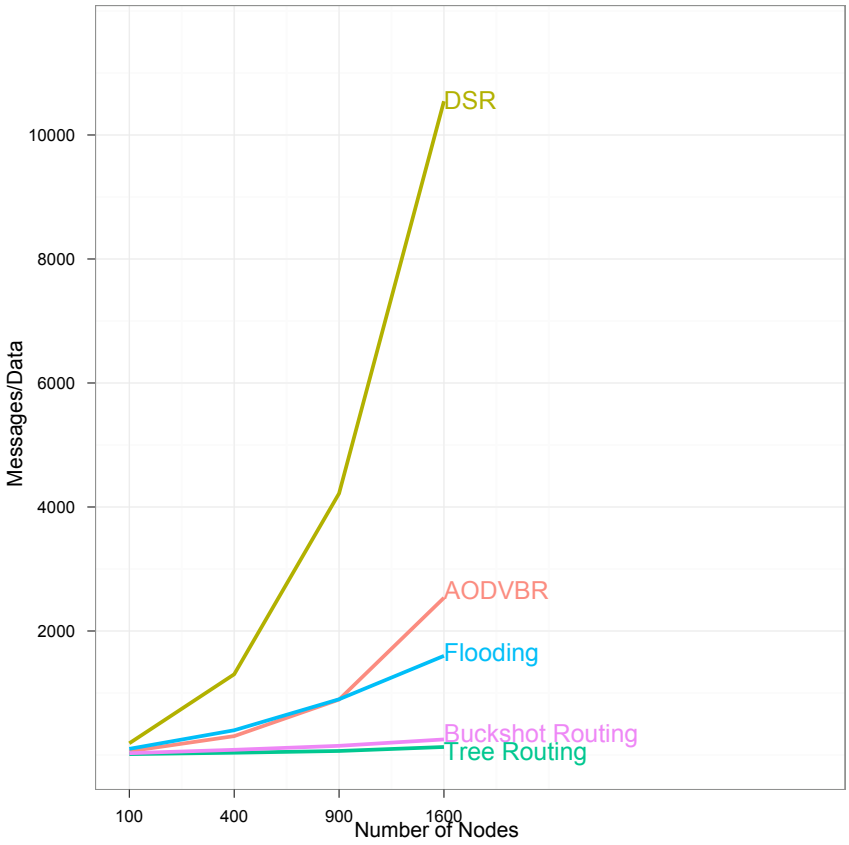


Figure 5.27: Number of Messages transmitted to deliver a single application message,AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 2

When the network load is considered (figure 5.27), the impact of the low number of messages transmitted by `Tree Routing` can be seen even better: The number of messages transmitted to deliver a single application message would suggest that `Tree Routing` is an excellent choice. However, this fact needs to be correlated with the delivery ratio in most cases, and the delivery ratio of `Tree Routing` is the lowest of all protocols. This is once again due to the length of routes. `Tree Routing` delivers a nearly constant number of data messages to the destination (roundabout 8000) for the networks with 400, 900 and 1600 nodes, even though the total number of application messages that is handed to the routing protocol increases proportionally to the number of nodes in the network.

BuckshotDV, OSBRDV, ULC and ULTR

The delivery ratio of BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR is shown in figure 5.28. Note that the scale starts at 80%. Except for ULC, all protocols deliver well above 95% of application messages for networks containing at least 400 nodes. The figure once more confirms that the performance of BuckshotDV increases with network size as the number of available redundant paths increases. OSBRDV does not performs as well as BuckshotDV, which can be attributed to the delaying of messages and the potential loss of redundancy incurred thereby. But with 97% delivery ratio the performance is still much better than that of the related work protocols. The performance of ULTR seems largely independent of the network size with a slight increase from 96% to 97% for the network with 900 nodes.

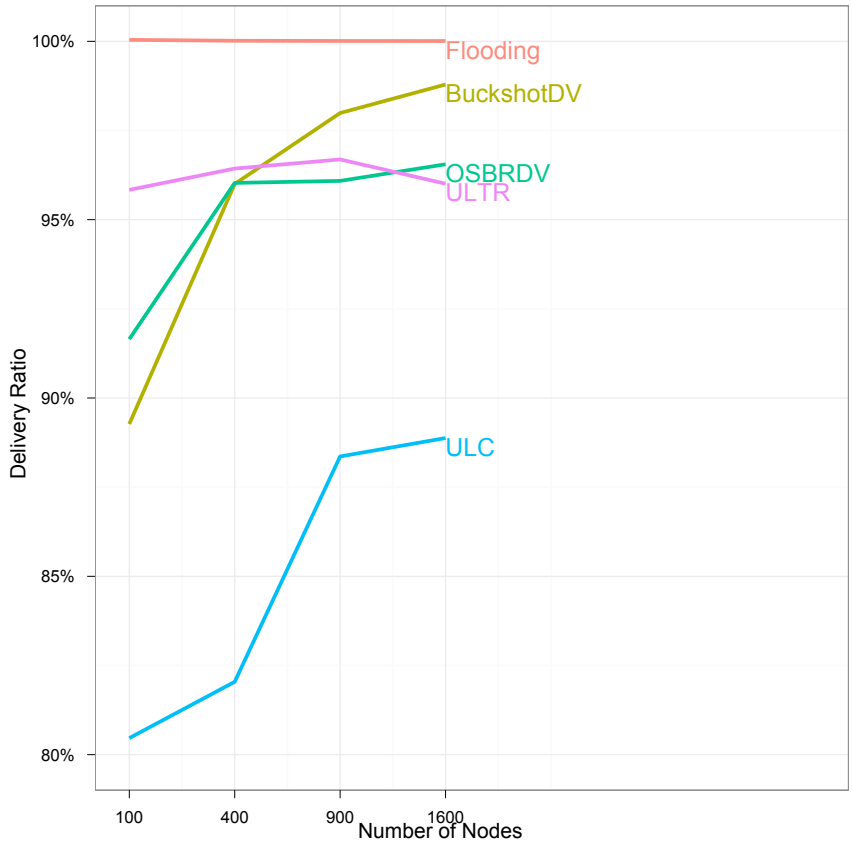


Figure 5.28: Delivery ratio of BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, scale starts at 80%, Scenario 2

Figure 5.29 shows the total number of messages transmitted by BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR. The number of transmitted messages is nearly the same for all protocols, except for `Flooding`. When these results are compared to those of the sense-and-send scenario, it can be seen that the number of messages transmitted by BuckshotDV has increased much more than that of the other protocols. This is the price that BuckshotDV pays for a high delivery ratio: As the routing tables are continuously refreshed by messages from the partner node, the number of nodes that receive a message and also know the next but one hop increases. As all of these forward the messages in BuckshotDV, the number of redundant paths that are used is increased. This leads to an increase of delivery ratio of roughly 5% for all network sizes. But to achieve this raise in delivery ratio the number of transmitted messages is nearly doubled.

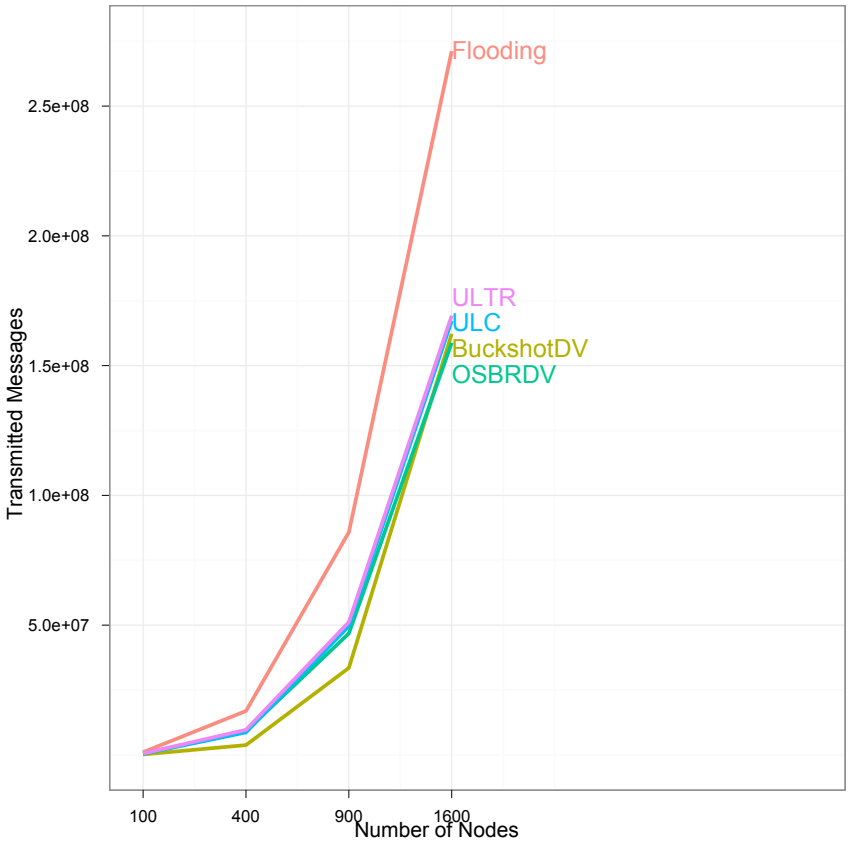


Figure 5.29: Number of transmitted Messages, BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, Scenario 2

The increased number of messages transmitted by BuckshotDV naturally also increases the umber of messages transmitted to deliver a single application message. Figure 5.30 shows the performance of BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR in that regard. BuckshotDV still performs best, but only marginally. When it is compared to the performance of Buckshot Routing in its source routing variant, it can be seen that the source routing variant transmits much fewer messages per delivered data message but only has a delivery ratio of 59% for the network consisting of 1600 nodes, whereas BuckshotDV delivers 99%. This is a good example for a choice to be made by the application programmer: If high network load poses a problem but message delivery might fail every once in while, source routing Buckshot Routing can be used. But if the delivery ratio takes prominence over all else, BuckshotDV is the protocol of choice.

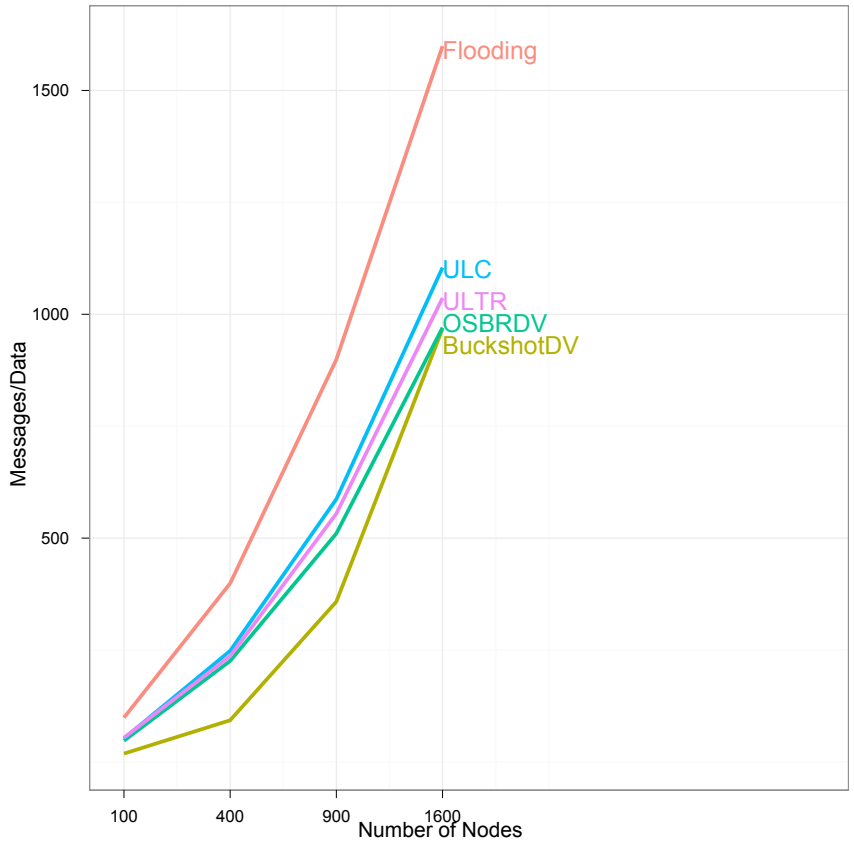


Figure 5.30: Number of Messages transmitted to deliver a single application message, BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, Scenario 2

Comparison between all Protocols

The delivery ratio achieved by each of the simulated protocols in the single pairing scenario is shown in figure 5.31. It can be seen that the protocols developed in this thesis all perform better than those chosen from related work. Moreover, except for Buckshot Routing, the delivery ratio stays the same or increases with network size. For AODVBR, DSR, `Tree Routing` and Buckshot Routing the delivery ratio decreased with network size. But most interestingly, the delivery ratio of DSR improved drastically when compared to the sense-and-send scenario. Also, the decline in delivery ratio with increasing number of nodes is visible, but it is not as steep as for AODVBR and `Tree Routing`.

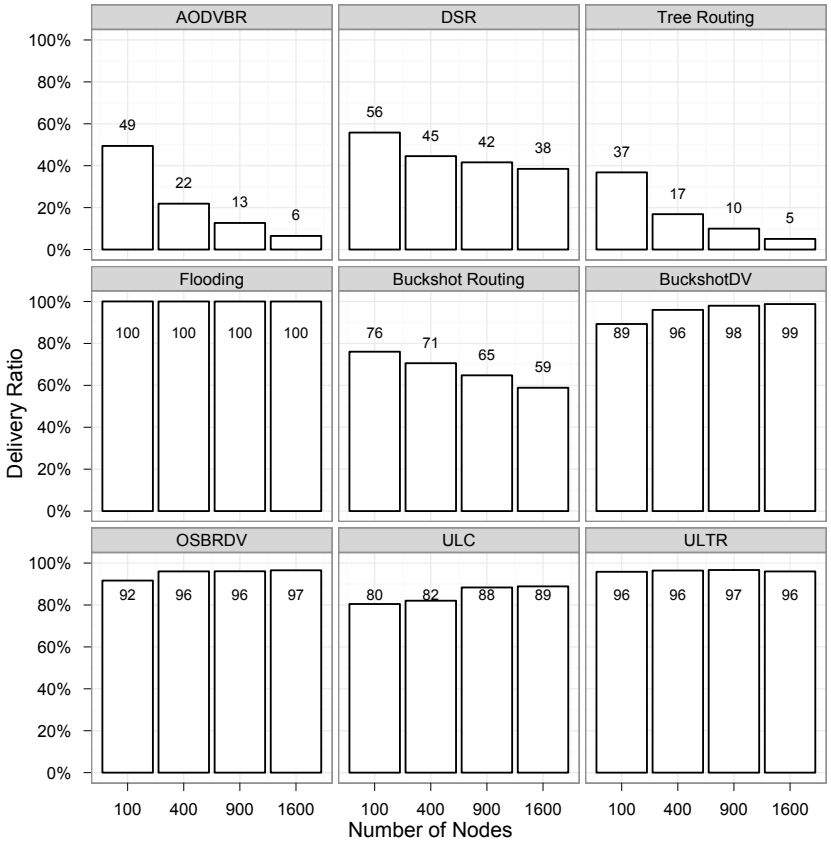


Figure 5.31: Delivery Ratio of all Protocols for different Network Sizes, Scenario 2

Concluding the evaluation of these simulations it can be said that DSR gained most from the change of application scenario. This was expected, as DSR was designed for

MANET scenarios, not for sense-and-send scenarios in wireless sensor networks. However, Buckshot Routing, BuckshotDV and OSBRDV also show an increase in delivery ratio as the implicit route maintenance starts to work for them. Also, the delivery ratio of protocols developed for this thesis is still higher than that of the related work protocols, for all network sizes.

5.5.2 Real World Experiment results

In the experiments for the single paring scenario, only two locations were used: The desk and the stone pavement. No experiments were made on the poles, because of the similarity between pole and desk scenario. On the desk, all nodes can communicate directly while on the poles the logical distance between nodes was only 1-2 hops even in the sense-and-send scenario where the destination was on the corner of the deployed grid. The pairings used in this scenario reduce the average route length and would result in even more single hop routes for the pole scenario, making the experiments redundant. The lawn placement has been neglected due to its similarity with the stone pavement placement.

Figure 5.32 shows the delivery ratios of all protocols that were achieved in the real world experiments on the desk and stone pavement. With the exception of ULTR, all protocols delivered 100% of messages in the desk scenario. This behavior has also been seen in the sense-and-send scenario and can be explained by the absence of up-to-date link information. ULTR normally depends on the MAC layer or the application to deliver neighborhood information. As none was available, neither from MAC nor from the application, the current implementation relies on passive gathering of neighborhood information by overhearing the forwarding of messages. But in a single hop environment not enough forwarded messages are overheard.

In the stone pavement experiments, even `Flooding` did not deliver all messages, which gives an insight into the MAC-layer problematic experienced more or less by all protocols. Apart from `Flooding`, OSBRDV has the best delivery ratio in this scenario. The reason for this lies in the delaying of messages by nodes that are not directly on the path which reduces the MAC layer problems experienced by Buckshot Routing and BuckshotDV which are next in line. `Tree Routing` also has a good delivery ratio in this scenario as it does not produce too much network load and the average path length was fairly small, making its two retransmissions a good reaction to message loss. ULC and ULTR both suffer from inaccurate information in their neighbor tables, and often use their fallback mechanism. DSR is continuously trying to repair routes, and thereby increases the network load very much, which can be seen in the next figure.

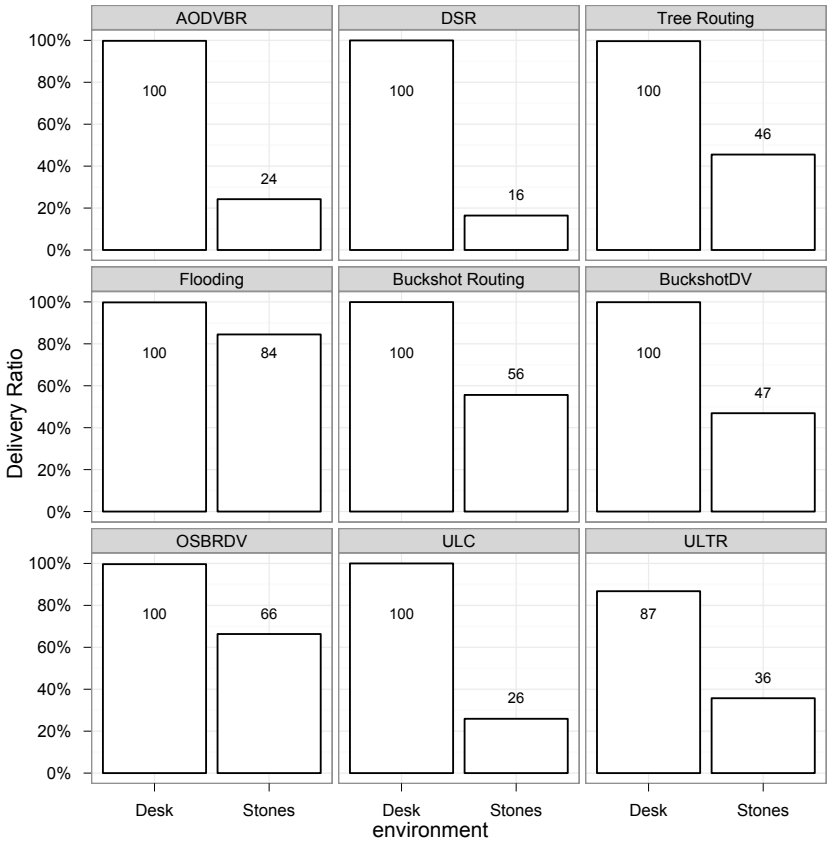


Figure 5.32: Delivery Ratio of each Protocol achieved in the real experiments, Scenario 2

The total number of messages transmitted by each protocol is shown in figure 5.33. For the experiments on the desk it can be noted, that ULTR transmits more messages than `Flooding` which can also be explained by the fallback mechanism in use: When ULTR starts route discovery, the network is flooded with a route request message. The destination receives this message and answers with a route reply but does not know if the link to the previous hop is unidirectional or bidirectional. Therefore, it uses the fallback mechanism, meaning that each node that knows the next hop forwards the message, which results in a second flooding of the network. Now that the route has been built, the data message can be transmitted. This process is repeated every time that the link timeout removes a link to the destination from a nodes neighbor table.

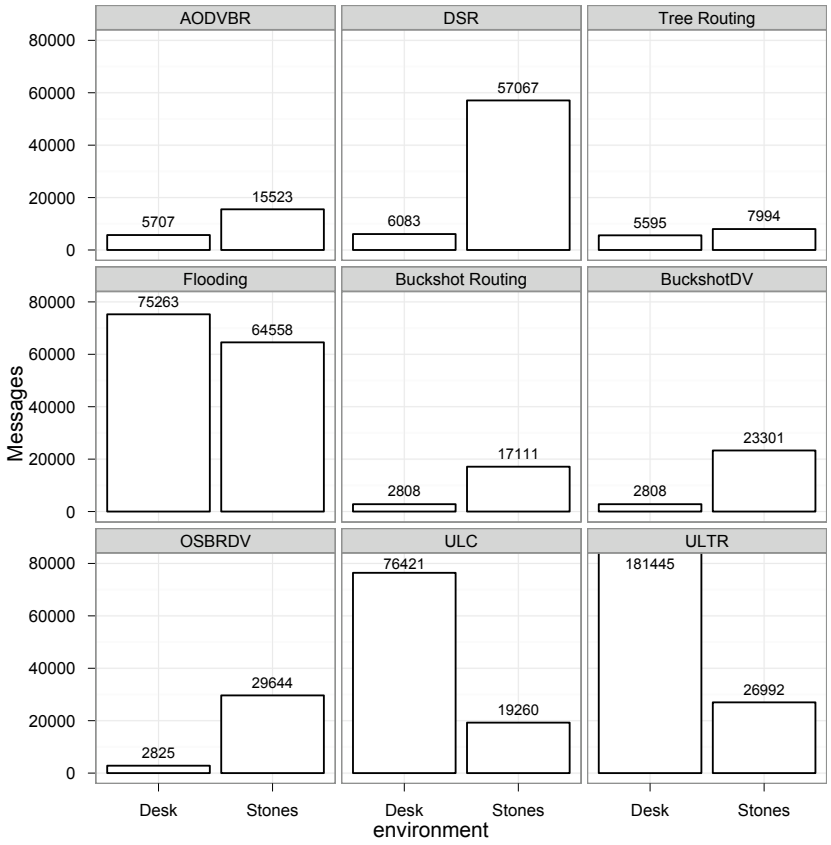


Figure 5.33: Total Number of Messages transmitted by each Protocol, Scenario 2

In the stone pavement placement, the passive neighborhood discovery works much better, leading to fewer messages transmitted by ULC and ULTR. Here, DSR transmits more than 57.000 messages and thus nearly as many as `Flooding`. The protocols developed in this thesis transmit between 17.000 and 30.000 messages while AODVBR and `Tree Routing` transmit about 15.000 and 8.000 messages respectively. These numbers already hint at the fact that `Tree Routing` profits quite a lot from the application setting and the small network diameter.

A more detailed look at the number of messages transmitted by each protocol is given in figure 5.34, where only the protocol packets are counted. Naturally, `Flooding` has the least number of protocol messages as it does not use any, and all transmitted packets are data messages. On the desk, Buckshot, BuckshotDV and OSBRDV transmit nearly the same amount of messages (648-665), while AODVBR, ULC and `Tree Routing` transmit

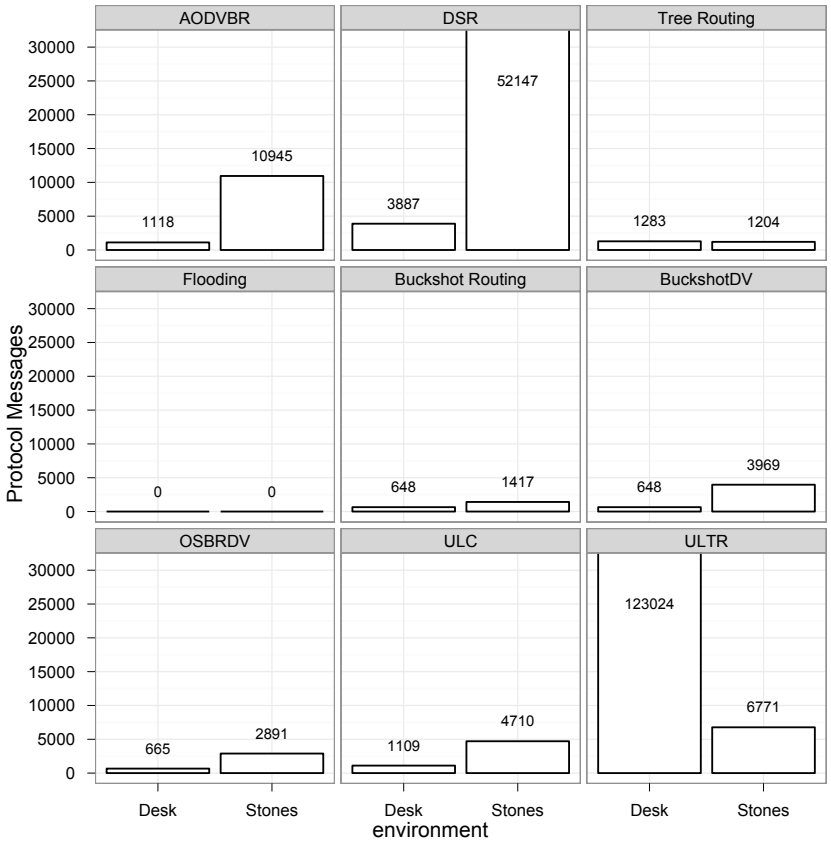


Figure 5.34: Number of Protocol Messages transmitted by each Protocol, Scenario 2

about twice as much (1109-1283). As 36 nodes were present in the network, a flooding of one route request or tree building message by each node would result in 1296 (36×36) transmissions. Therefore, these three protocols transmitted the expected number of messages. Buckshot, BuckshotDV and OSBRDV transmit fewer messages, because they do not need to start a route discovery for each node: When a node overhears a different node transmitting a route reply message, it extracts the information contained therein and enters it into its own routing table. DSR and ULTR flood the network multiple times for each route discovery, resulting in an awfully high number of route request and route reply messages. For DSR this is due to the specification for the operation in the presence of unidirectional links. For ULTR it is once more due to the absence of accurate neighborhood information.

On the stone pavement, the number of protocol messages rises enormously for DSR, as a lot of link breaks lead to the creation of route error messages and subsequent new floodings of the network in order to find a new route. The lowest number of protocol messages (apart from `Flooding`) is transmitted by `Tree Routing` which only transmits its tree building messages at the start of the experiment. When this figure is compared to the previous one, it can be seen that `Tree Routing` transmitted about 6.700 data messages, meaning that most of the time the two retransmissions took place. Buckshot Routing also has a low number of transmitted protocol packets, but the usage of redundant paths leads to a higher number of data messages transmitted.

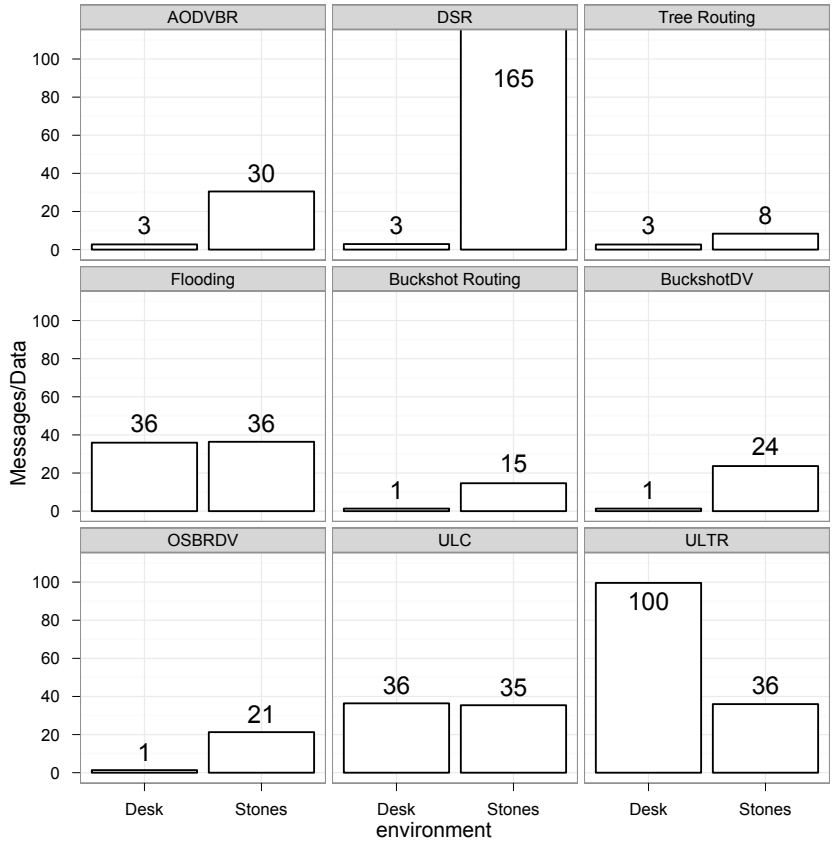


Figure 5.35: Total Number of Messages transmitted by each Protocol divided by the number of delivered data messages, Scenario 2

The number of messages transmitted to deliver a single application message is shown in figure 5.35. As there were 36 nodes in the network, `Flooding` transmitted 36 mes-

sages for each data message delivered to the destination. The overhearing of route reply messages described above leads to a good performance on the desk for Buckshot, BuckshotDV and OSBRDV, with AODVBR, DSR and `Tree Routing` following close. The high number of data messages transmitted by ULC and ULTR due to the inaccurate neighborhood information leads to a performance equal to `Flooding` for ULC and an even worse one for ULTR.

On the stone pavement, `Tree Routing` performed best, with Buckshot, OSBRDV and Buckshot following. When the delivery ratio (figure 5.32) is also taken into account it can be said that for this application scenario, network size an placement, the choice of routing protocol should be made between `Tree Routing`, Buckshot Routing and OSBRDV. OSBRDV has the highest delivery ratio, and should be chosen when network load is not a major concern. `Tree Routing` produced the least network load per application message delivered and should be chosen if some message losses could be tolerated but the network load is the most important factor. Buckshot Routing represents a good choice in between.

5.6 Application Scenario 3: Multiple Pairings

The third application scenario, multiple pairings, once again uses the same settings as the two previous ones, only the application was changed. Instead of all nodes transmitting to a single sink or one communication partner for each node, there are multiple partners now. Each node has one communication partner at the start of the simulations/-experiments and transmits the first five messages to this node. Once five messages have been transmitted, the communication partner is changed. This is repeated every time five messages have been transmitted, until the total number of messages specified (110 for simulations, 60 for experiments) has been reached. The pairings of nodes were once again generated randomly before the start, and the same pairings were used for all protocols.

This represents a MANET scenario where all nodes only want to exchange a few messages with a chosen partner before communicating with a different node. The fact that each pairing is only used for five messages results in a reduction of the importance of route maintenance. It is much more likely that a route is stable for five minutes than for a whole simulation/experiment, resulting in less route errors. Instead, route discovery rises in importance, as it is carried out after every five application messages.

5.6.1 Simulation results

The simulations once again used the connectivity change lists that were generated before the start, to keep network connectivity equal for all protocols. As in the single pairing scenario, the pairings define a different destination for each node, making the additional simulation parameter `destination` used in the sense-and-send scenario unnecessary.

The delivery ratio remains defined as the number of application messages delivered by a protocol divided by the number of messages delivered by `Flooding` in the simulations.

Buckshot Routing and Related Work Protocols

The delivery ratio of Buckshot Routing is compared to that of the related work protocols in figure 5.36. It can be seen that Buckshot Routing still outperforms all related work protocols, even though the application scenario has been switched to one that should be better for the related work protocols. As the importance of route maintenance is reduced, one of the advantages of Buckshot Routing, the implicit route maintenance, has only a small impact.

For AODVBR and `Tree Routing`, the number of nodes and therefore the route length is much more important than the communication pattern of the application: The

changes between single pairing and multiple pairings are marginal. The performance of `Tree Routing` increased by one percent for the largest network while that of AODVBR decreased by two percent. A bigger difference can be seen for the smaller networks, where AODVBR has lost 10% of its performance compared to the single pairing scenario in the network consisting of 100 nodes. This decrease in delivery ratio is due to the fact that building the initial route is one of the weaknesses in AODVBR. When searching for a route, the path has to be bidirectional to enable the route reply to use the same path as the route request. Once this path has been established, the fish bone structure that has been built with the route replies can be used to salvage data messages when links break. In the multiple pairings scenario, each node needs to search routes to 22 different nodes instead of only one.

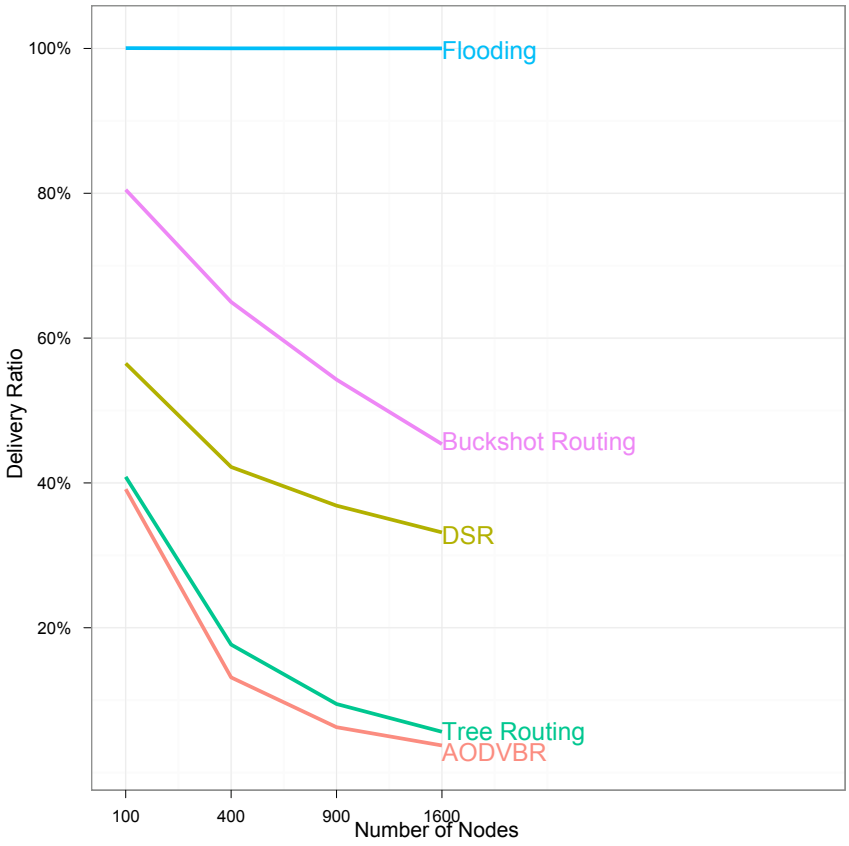


Figure 5.36: Delivery ratio of AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 3

The number of messages transmitted by Buckshot Routing and the related work protocols is shown in figure 5.37. With twice the number of transmitted messages as `Flooding`, DSR once more transmitted the most messages by far. Buckshot Routing, AODVBR and `Tree Routing` transmitted far less messages, with `Tree Routing` producing the least number. When the results are compared to those of the single pairing scenario, only Buckshot Routing shows a significant difference. This is due to the fact that Buckshot Routing now needs 22 times as many floodings of the network, one for each new route discovery and node in the network instead of only one for each node. As Buckshot Routing does not transmit any route maintenance messages, route discovery and data transmission are the two factors that define its performance. Therefore, the increased number of route discoveries has a strong influence on the number of transmitted messages.

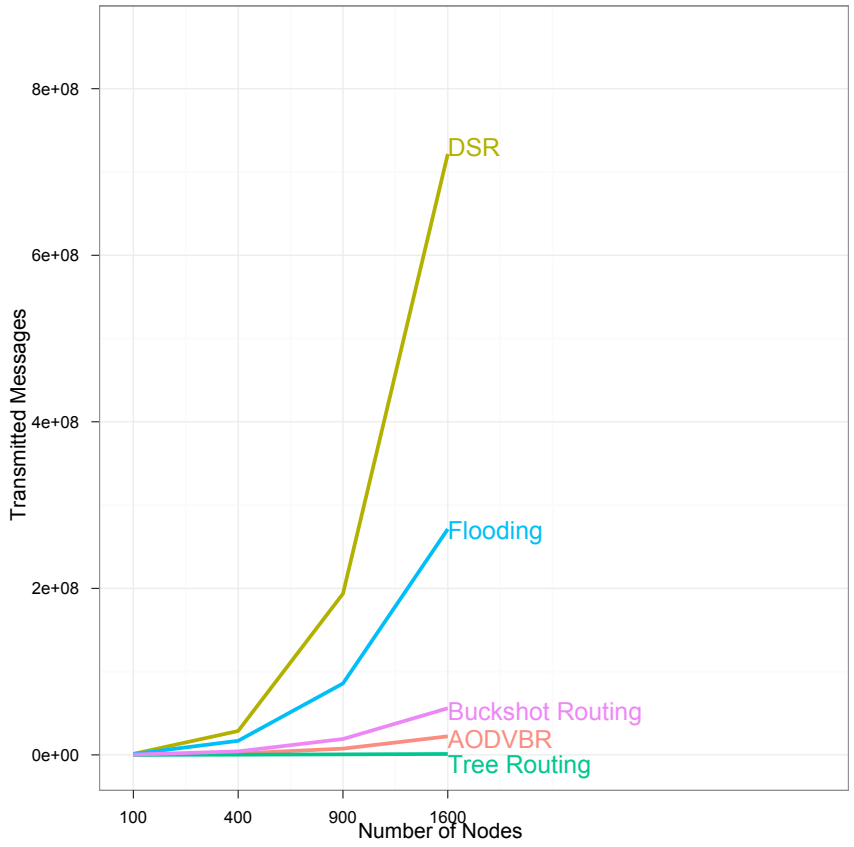


Figure 5.37: Number of transmitted Messages, AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 3

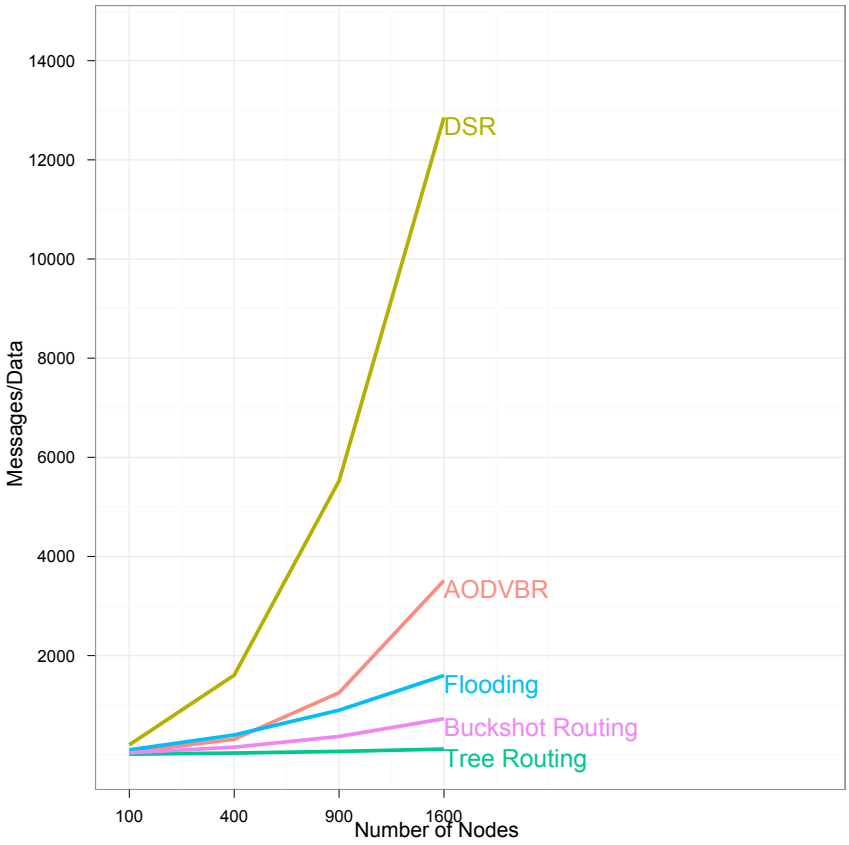


Figure 5.38: Number of Messages transmitted to deliver a single application message, AODV-BR, Buckshot Routing, DSR, `Flooding` and `Tree Routing`, Scenario 3

The cost of delivering a single data message measured in transmitted messages is shown in figure 5.38. Even though Buckshot Routing transmitted more messages than AODVBR, the much higher number of delivered messages results in a fairly good performance. Only `Tree Routing` transmitted less messages per application message delivered. However, this is once more due to the fact that the cost of delivery failure is small in `Tree Routing`. When the delivery ratio is also taken into account, Buckshot Routing emerges as the better protocol. On the downside, the increased number of messages transmitted by Buckshot Routing when compared to the single pairing scenario results in an increased cost of delivered messages.

BuckshotDV, OSBRDV, ULC and ULTR

The delivery ratio achieved by BuckshotDV, OSBRDV, ULC and ULTR is shown in figure 5.39, note that the scale starts at 80%. It can be seen that all protocols deliver more than 95% of application messages, regardless of network size. The only exception is ULC, which starts at 88% for the network containing 100 nodes and rises up to 93% for the largest network, containing 1600 nodes. OSBRDV and ULTR deliver between 95% and 97%, with only a low variation between network sizes. BuckshotDV starts with a delivery ratio of 95% and increases its performance up to 99%.

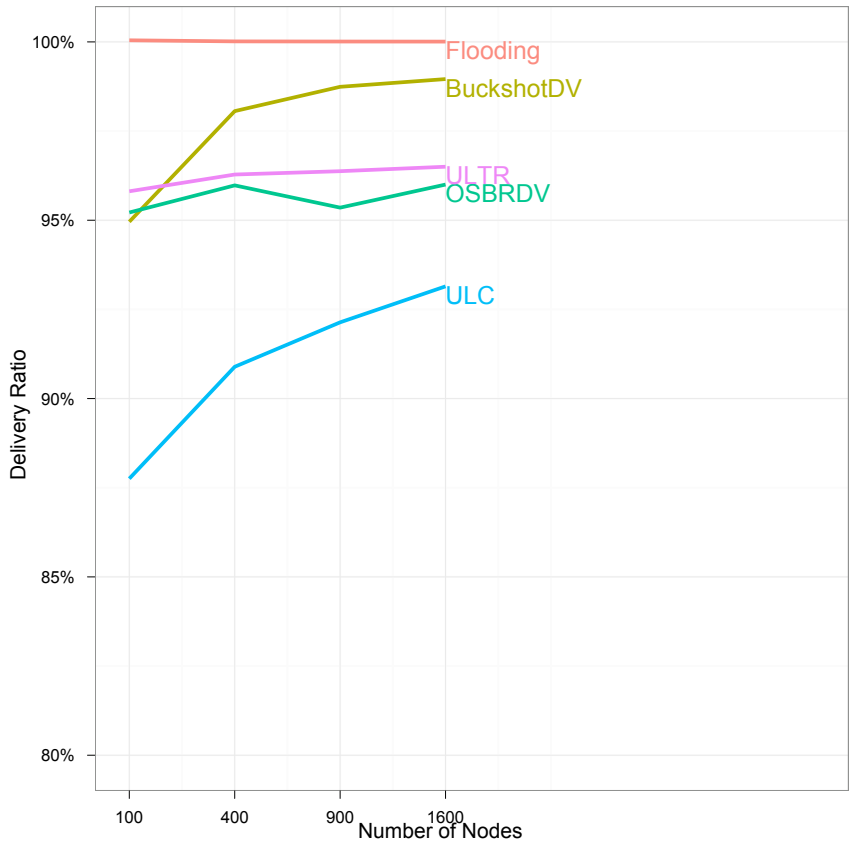


Figure 5.39: Delivery ratio of BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, scale starts at 80%, Scenario 3

The high number of delivered messages comes at the price of an increased number of transmitted messages, as figure 5.40 confirms. Here, it can be seen that the number of messages transmitted by BuckshotDV, OSBRDV, ULC and ULTR has risen when

compared to the single pairing scenario. While the number is still lower than that of `Flooding` for all four protocols, it has gotten close. This is especially true for BuckshotDV, which now transmits the highest number of messages except for `Flooding`. In the single pairing scenario, BuckshotDV transmitted the second least number of messages, only OSBRDV transmitted less. The fact that the number of transmitted messages rises for all protocols developed in this thesis can be explained by the increase in redundancy and the higher number of route searches as the route replies already use multiple redundant paths.

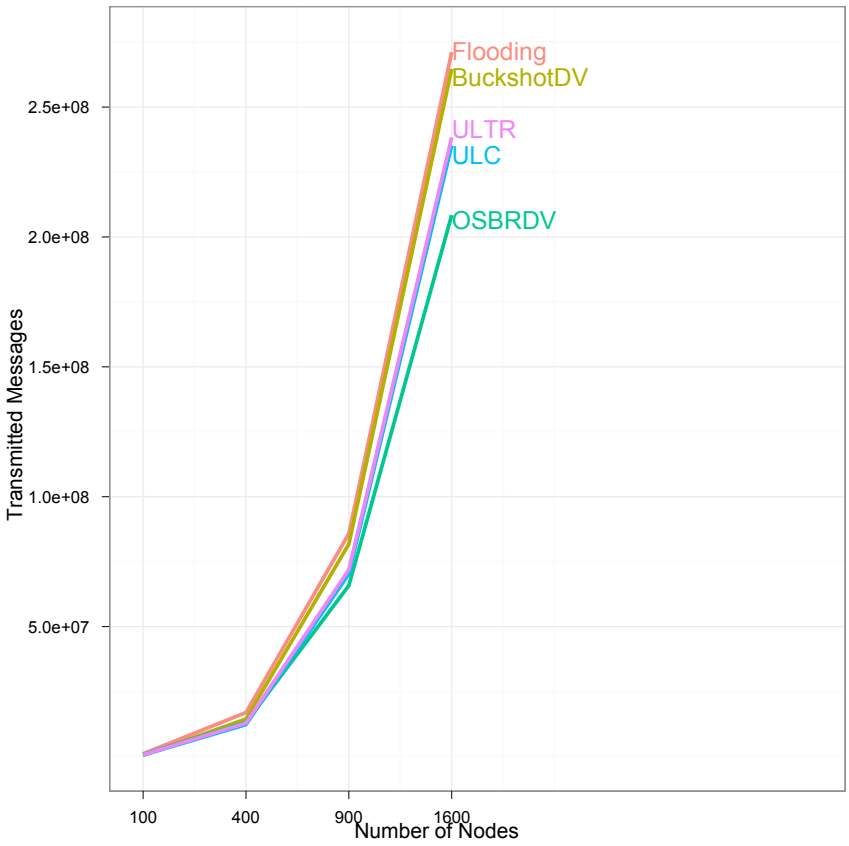


Figure 5.40: Number of transmitted Messages, BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, Scenario 3

This high number of transmitted messages is the reason why the performance of BuckshotDV decreases in the multiple pairings scenario. Figure 5.41 shows the performance of BuckshotDV, OSBRDV, ULC and ULTR measured in messages transmitted

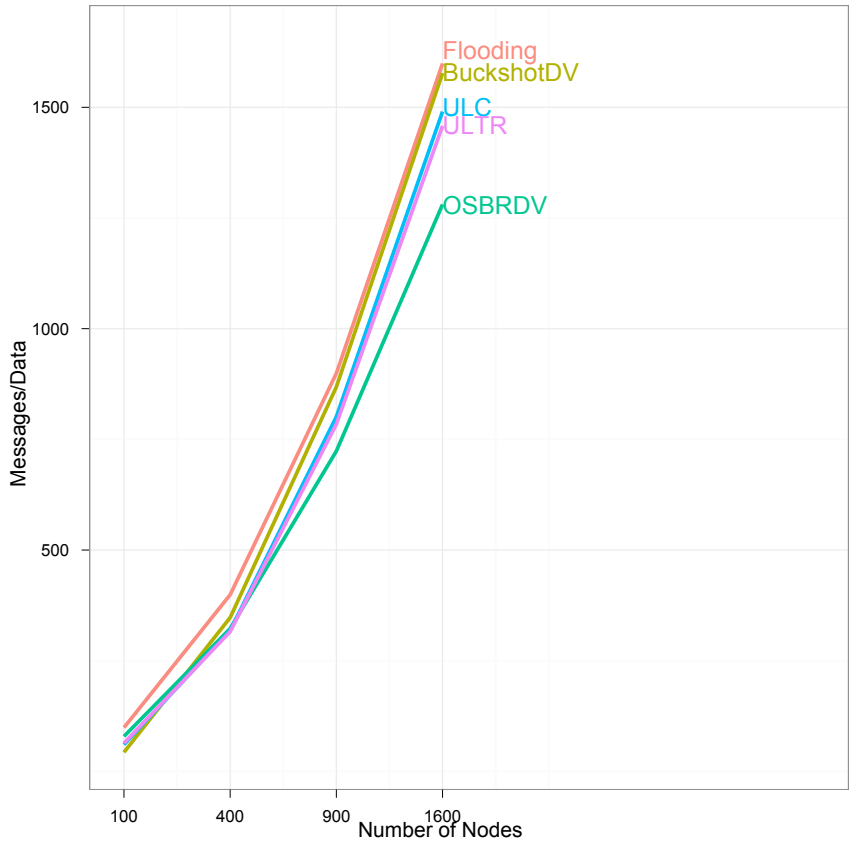


Figure 5.41: Number of Messages transmitted to deliver a single application message, BuckshotDV, `Flooding`, OSBRDV, ULC and ULTR, Scenario 3

per application message delivered. If only this figure were concerned, OSBRDV would be the protocol of choice. When the delivery ratio is also taken into account, it is no longer easy to say which protocol should be preferred. OSBRDV has the best messages/data ratio and delivers 95-96% of messages. In most applications, that will be enough. But if a higher delivery ratio is needed, ULTR, BuckshotDV or even `Flooding` might be considered for this application scenario.

Comparison between all Protocols

The delivery ratio of all protocols is compared in figure 5.42. The related work protocols, AODVBR, DSR and `Tree Routing` all show a steep decline in delivery ratio, with DSR performing best of these three. Interestingly, the decline of delivery ratio is not as steep

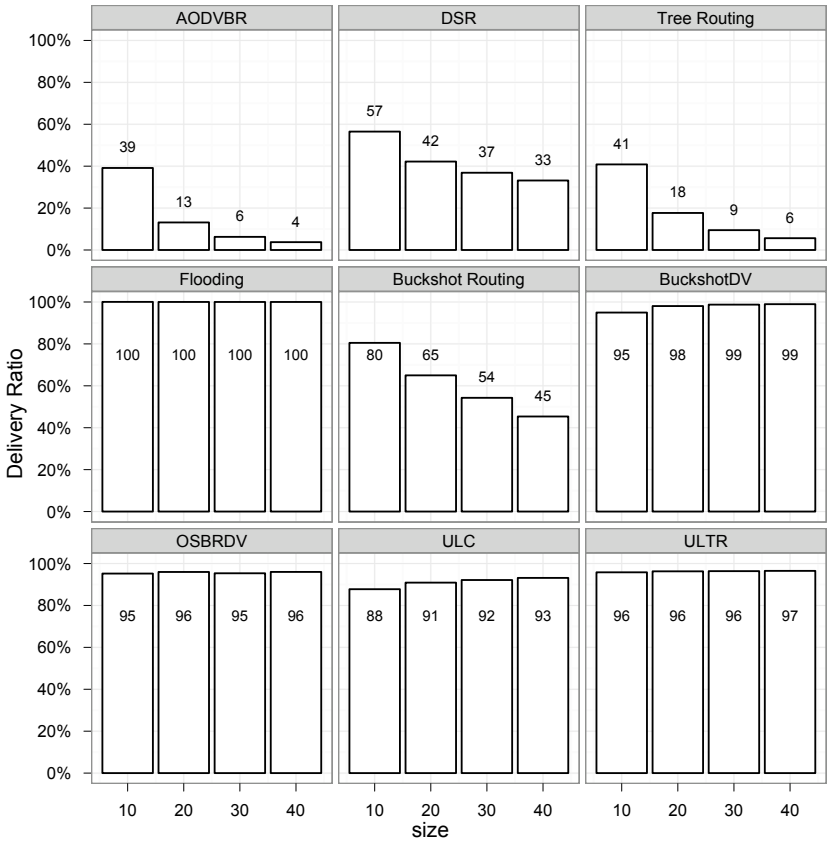


Figure 5.42: Delivery Ratio of all Protocols for different Network Sizes, Scenario 3

for DSR as it is for AODVBR and `Tree Routing`. This is due to the fact that DSR has a better route discovery mechanism. While flooding the whole network twice in order to establish a route produces a lot of network load, it also means that a route will be found in most cases. Only if network separation occurred, no route will be found. How long a route found this way can be used depends on link stability, however. But since it only needs to be used for five messages before a different destination is selected, there is a good chance some of the five messages can be transmitted successfully. This can be seen in the network with 1600 nodes, where DSR was able to deliver one third of application messages, meaning that between one and two messages were delivered to each destination on average.

Buckshot Routing also suffered from increased route length due to its source routing nature while BuckshotDV, ULC and ULTR increase their performance with increased

number of messages. The performance of OSBRDV is mostly independent of the network size.

5.6.2 Real World Experiment results

The experiments for the multiple pairings scenario featured the same settings and locations as the experiments for the single pairing scenario (section 5.5.2): The desk placement was used as single hop, and the stone pavement as multihop environment. The pole placement would have been redundant to the desk placement while the lawn placement would have been similar to the stone pavement environment.

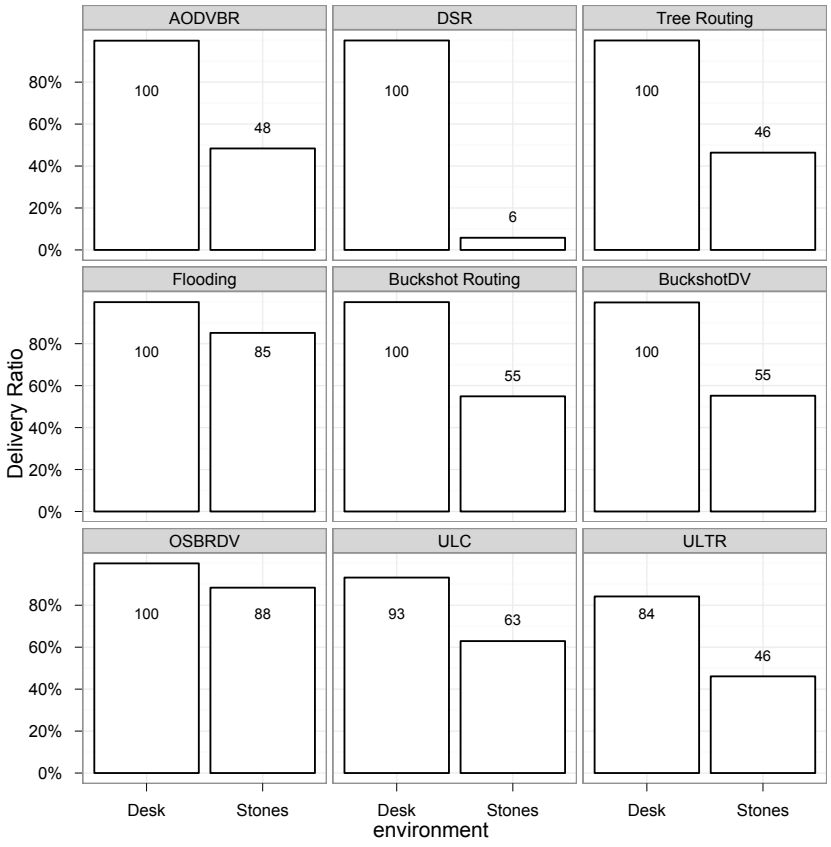


Figure 5.43: Delivery Ratio of each Protocol achieved in the real experiments, Scenario 3

The delivery ratio achieved by all protocols in the multiple pairing scenario is shown in figure 5.43. In the desk experiments, all protocols reached 100 % delivery ratio except

for ULC and ULTR. This is due to the passive neighborhood discovery used in both protocols: Only when the forwarding of a message is overheard by a node that has already forwarded that message and is listed as last hop, the neighborhood discovery assumes bidirectional links. Otherwise, links are assumed to be unidirectional. This leads to a lot of mistakes, as nodes do not need to forward messages in a single hop environment, meaning that all links in the network are assumed to be unidirectional. Therefore, the backup mechanism is always used unnecessarily, resulting in a high network load which in turn leads to more collisions and message loss.

On the stone pavement, OSBRDV has the highest delivery ratio, even higher than `Flooding`. The reason for this can be found in the MAC layer, which has problems with a high network load. In OSBRDV, all nodes that are not on the direct path store the messages for some time, reducing the immediate network load compared to BuckshotDV. However, these messages are transmitted later, increasing the total number of messages.

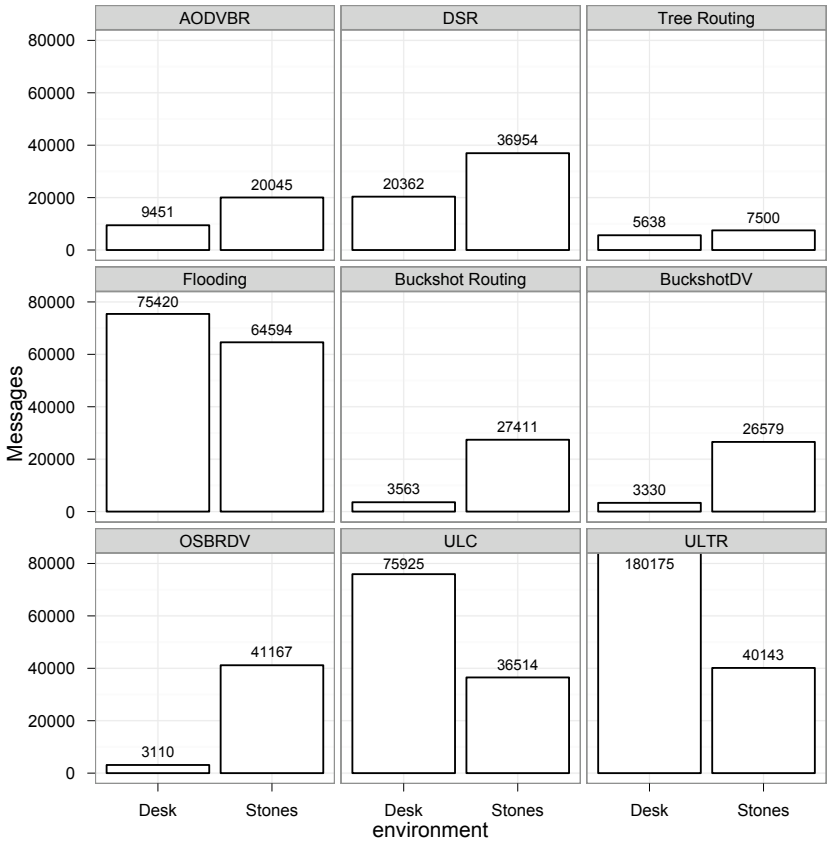


Figure 5.44: Total Number of Messages transmitted by each Protocol, Scenario 3

This total number of transmitted messages is shown for all protocols in figure 5.44. ULTR once more has the highest number of transmitted messages for the single hop environment due to the problems with the neighborhood detection. On the stone pavement, the passive neighborhood detection works better, and the number of transmitted messages is reduced. There, `Flooding` transmits the greatest number of messages while `Tree Routing` transmits the smallest. Still, when considering that only 2160 application messages were generated it can be seen that `Tree Routing` often used its two retransmissions.

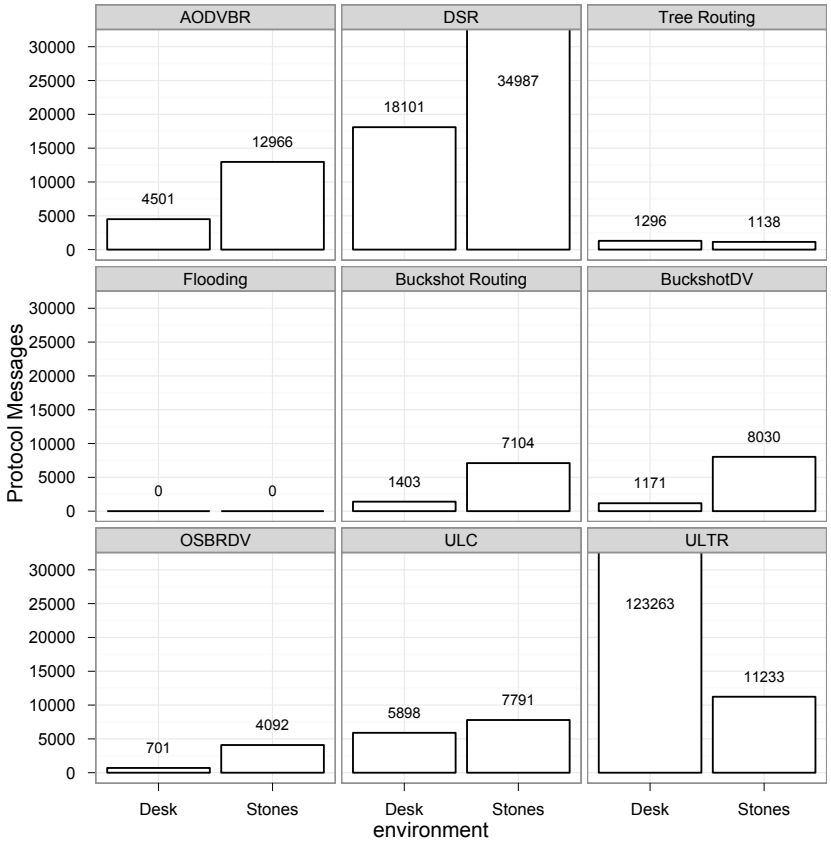


Figure 5.45: Number of Protocol Messages transmitted by each Protocol, Scenario 3

The number of protocol messages transmitted by each protocol can be seen in figure 5.45. `Flooding` naturally did not transmit any protocol messages, followed by OSBRDV which transmitted 701 protocol messages and BuckshotDV with 1171 messages in the desk placement. ULTR transmitted the most protocol messages.

On the stone pavement, `Tree Routing` needed the least number of protocol messages, apart from `Flooding`, followed by OSBRDV. DSR transmitted the most protocol messages, followed by AODVBR and ULTR. Buckshot Routing, ULC and BuckshotDV are placed somewhat in between.

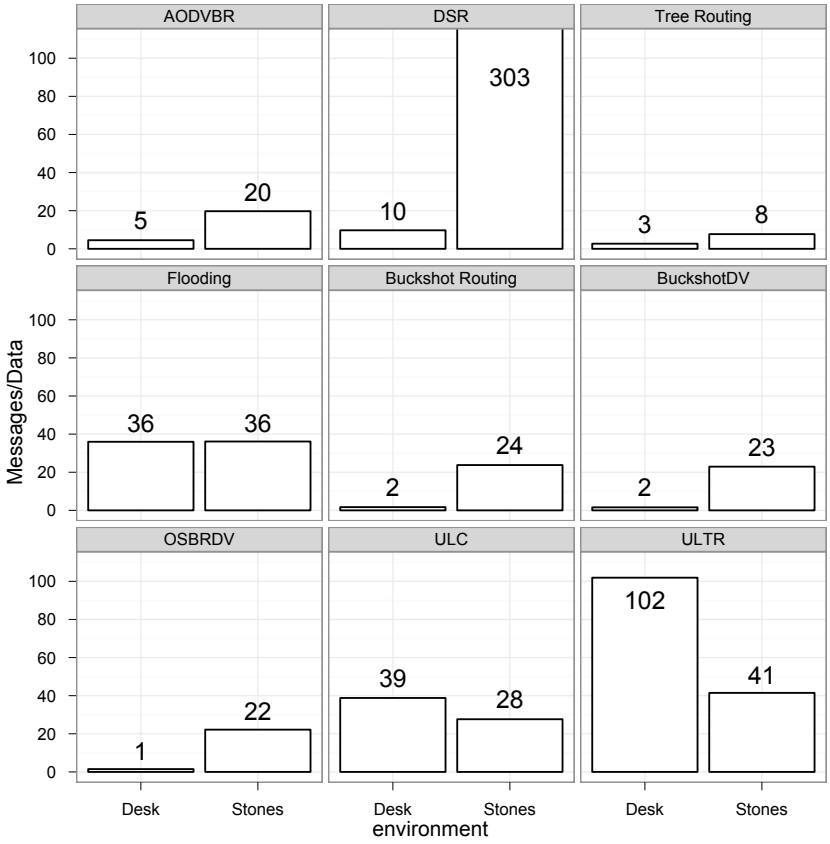


Figure 5.46: Total Number of Messages transmitted by each Protocol divided by the number of delivered data messages, Scenario 3

The number of transmitted messages divided by the number of delivered application messages is used to measure the performance of all protocols in figure 5.46. For the desk placement, OSBRDV shows the best performance, directly followed by Buckshot Routing and BuckshotDV. `Tree Routing` is placed shortly thereafter, with AODVBR following.

When the sensor nodes were placed on the stone pavement, `Tree Routing` needed the least number of transmissions to deliver a single application message, which is once

again due to the low cost of delivery failure. When only the cost of an application message delivery is considered, `Tree Routing` performs best. However, OSBRDV delivered nearly twice as many messages but needs more messages to reach this increase in delivery ratio. If the delivery ratio is most important, OSBRDV would be chosen for such small networks and this application scenario. If the network load is more important, `Tree Routing` should be chosen.

5.7 Interaction between Routing Protocols and Duplicate Suppression

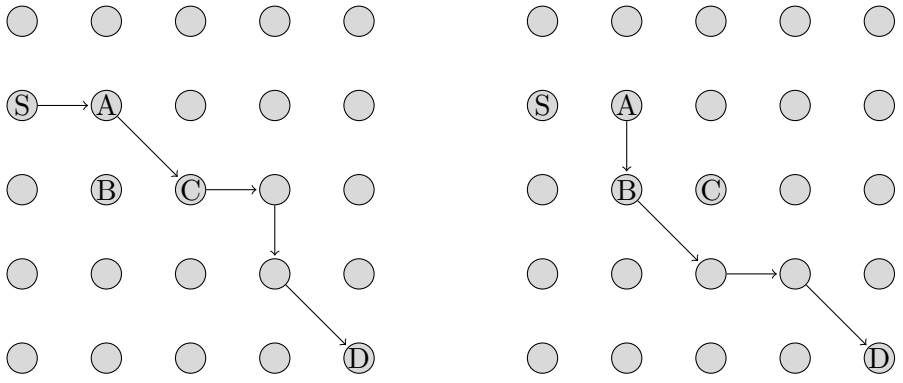
Most papers describing routing protocols do not specify the way duplicate detection should be implemented. Often, it is only stated that a duplicate suppression mechanism should be used. If a way of suppressing duplicates is specified, it is frequently proposed to use the identity of the originator and a sequence number to uniquely identify a message.

When a node receives a message, the first thing it should do in order to avoid unnecessary work is to consult its duplicate suppression. If the message has been received and handled before, there is no need to process it any further and it is silently discarded. Otherwise, the tuple (sender ID, sequence number) is entered into the duplicate list and the message is processed.

During work on the implementation of BuckshotDV it became apparent that this may lead to message loss under certain circumstances which are described in this section.

5.7.1 Route Changes in Intermediate Nodes

The first problem can only arise for distance vector protocols. Figure 5.47(a) shows a path from node S to node D that node S has built with a route discovery. It leads through nodes A and C.



(a) Path from S to D via A

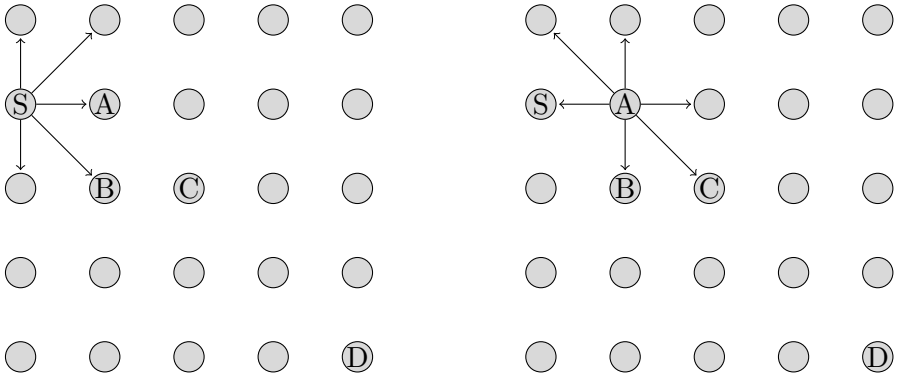
(b) Path from A to D

Figure 5.47: Paths From S to D and from A to D

During the lifetime of that path, node A has learned a different path that leads to node D (figure 5.47(b)). This could be due to timeouts or simply related to the fact

that caching of overheard routes is enabled. As the new path from Node A to node D is not longer but more current than the one stored in node A's routing table, most protocols would use this new path in the future.

The problem arises the next time node S wants to send a message to node D. It looks up the next hop in its routing table and transmits the message to node A. Due to the broadcast characteristics of the wireless medium, not only node A but all nodes within communication range of node S receive this message (figure 5.48(a)). As the first thing they do upon reception of a message is checking for duplicates, each of the receiving nodes, including node B, decide that the message is no duplicate and enter it into their duplicate suppression. During the processing that follows, each node discards the message except for node A, which is listed as next hop.



(a) Path from A to D Step One

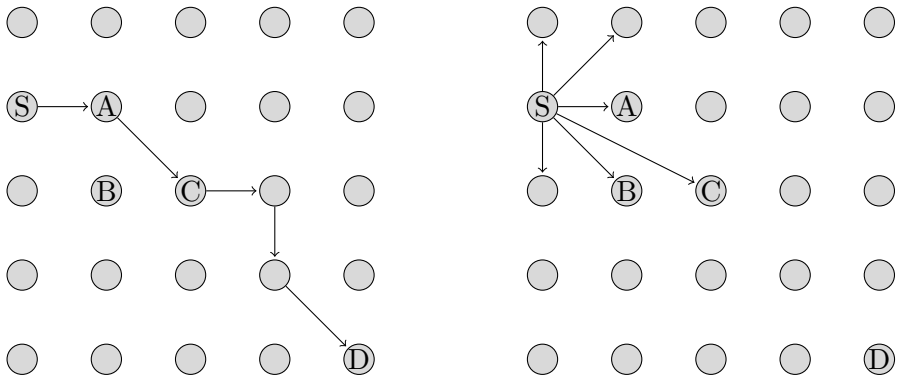
(b) Path from A to D Step Two

Figure 5.48: Hop wise Transmission

In the next step, node A looks up the next hop on the path to node D in its routing table and inserts node B as next hop in the message before retransmitting. Figure 5.48(b) shows which nodes receive the message, among them the intended next hop, node B. Upon reception, node B checks if the received message is a duplicate. As it still has the same originator and sequence number, it is identified as a duplicate and discarded even though node B was the intended forwarder. In source routing protocols this problem should not arise, because intermediate nodes do not change the route.

5.7.2 Changing Local Topologies and Unidirectional Links

This problem is experienced by both distance vector and source routing protocols. It is similar to the problem described above, as a node on the path overhears a message before it is addressed as next hop.



(a) Path from S to D via A (b) Path from S to D Step One

Figure 5.49: Paths From S to D and First Transmission

Figure 5.49(a) shows once again the path from node S to node D, leading from node S over node A and node C. Due to changes in the logical topology, the message that node S transmits is also received by nodes B and C (figure 5.49(b)) in step one. All nodes enter the message into their duplicate suppression, but only node A, being the intended forwarder, retransmits the message. When node C receives the message, it identifies it as a duplicate and discards the message, even though node C is the intended forwarder. Please note that it does not matter if the link between nodes S and C is unidirectional or bidirectional, as the error will occur in either situation. Still, as has been shown in section 2, unidirectional links often have a far greater reach than bidirectional ones, therefore such longer links might more often be unidirectional.

5.7.3 Example Results of BuckshotDV

To quantify the influence of the duplicate suppression on the performance of the routing protocols, BuckshotDV without caching of overheard routes was simulated in two different versions. In BuckshotDV there are three criteria which are checked to see if a DATA message that has been received by an intermediate node has to be discarded or processed:

- Duplicates are discarded
- If the enlisted next but one hop is not a neighbor, the message is discarded
- If no route to the destination is known, the message is discarded

Table 5.2: Delivered Messages for BuckshotDV, Duplicate Suppression first vs. last

Number of Nodes	Version A	Version B	Difference	Percent
100	7691.244	8586.817	895.5733	89.57037
400	32222.42	37224.84	5002.417	86.56162
900	79524.57	88284.47	8759.903	90.07764
1600	148562.3	160569.4	12007.10	92.52217

Only if a message has passed these three checks it is handled and forwarded.

The difference between the two evaluated versions lies only in this handling of a received DATA message. In the first version (A), duplicate suppression is used first. In the second version (B), the duplicate detection is moved to the end of these checks and only used on messages that pass the previous two checks.

Both protocol versions were evaluated with the same settings and the same number of simulations (4950 each) that were used for the sense-and-send scenario (section 5.4). Table 5.2 shows the results for the four different network sizes. Using the duplicate suppression first delivers fewer DATA messages for all network sizes, as expected. However, the amount of messages lost is also much higher than expected. Between 7.5% and 13.5% of DATA messages less were delivered because they were detected as duplicates (which is correct) and not forwarded (which is not correct in this case).

5.7.4 Solutions

There are multiple ways to solve these problems, depending on the protocol for which the problems should be solved:

- Disabling New Routes. The first problem can be solved by disabling the learning of new routes in the intermediate nodes for distance vector routing protocols. This is not recommended as it would obviously lead to stale routes.
- Use a Proactive Approach. If intermediate nodes know which other nodes use them as next hop they could notify those of the route changes, leading to a higher network load.
- Route Shortening. Route shortening enables intermediate nodes (node C in the example) to detect that they should forward the message in the future. Thus, they transmit it upon first reception. This is only possible for source routing protocols.

- Reordering of Processing Order. If a node completely processes a message before checking with the duplicate suppression, nodes that are not the intended next hop would discard the message without entering it into their duplicate suppression list. Obviously, this would also increase the computing load on the nodes.

For all protocols which include the next hop in their message, it seems to be the easiest way is to check upon reception of a message if the current node is the intended next hop and discard the message otherwise. Only after that, duplicate suppression takes place. But even that might not be enough if the forwarding of messages should be used as acknowledgment by the next but one hop. Then, the received message can surely be found in the list of duplicates, because it has been transmitted by this node before. Therefore, the processing order must be similar to the one described above for `OSBRDV` :

1. Check if the message is a passive acknowledgment
2. Establish whether this node is the next hop
3. Use duplicate suppression

5.8 Summary

The connectivity measurements have shown that unidirectional links occur even more often than literature suggests, and that links also change much more often than expected. But even under those conditions, the protocols developed for this thesis show a fine performance as the evaluations of three different applications scenarios have shown. However, the results also show that it is not possible to define a single best protocol.

When application programmers need to decide which routing protocol they are going to use, there are a number of influence factors that should be taken into account. Examples include the network diameter, the frequency of unidirectional links, link stability and the importance of network load versus delivery ratio among others.

Figure 5.50 provides a rough guideline for choosing a protocol if these four parameters are known. In the figure, the network diameter can be very small, small, medium or large. Unidirectional links can be rare, common or make up most of the links in the network ("mostly"). The link stability can be low, medium or high. The fourth choice concerns the importance of data: If the delivery of as many messages as possible takes preference over all, the optimization "delivery" is chosen. Otherwise, if the network load is high due to application requirements, increasing it even further might cause problems for the MAC or lead to energy concerns. Then, the optimization "load" should be chosen.

In the figure, the protocols evaluated in this thesis are shown separately, only ULC and ULTR are both recommended for small networks with a high number of unidirectional links and medium link stability. They are not separated further because they show a similar performance for the presented factors. However, ULC could provide the information it gathered about the status of links to another layer. Whether this is useful or not depends on the application and protocol stack in use. Moreover, all protocols might behave differently when other MAC protocols, other hardware or different placements are used. Therefore, the recommendations given in the figure are only a rough guide as there are a lot of other influence factors in a real world deployment.

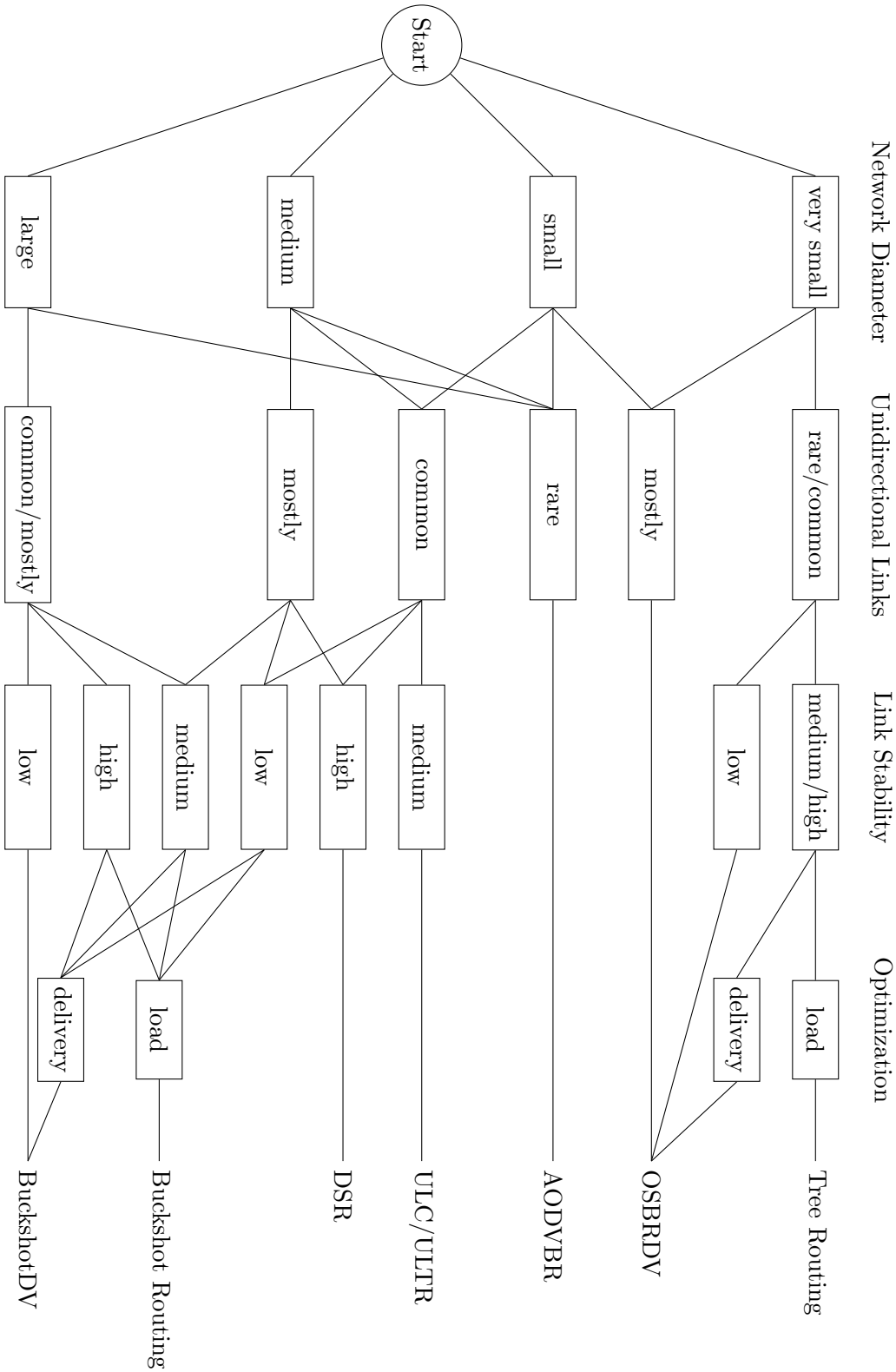


Figure 5.50: Choosing a Protocol under given Network Conditions

5.9 Own publications referenced in this chapter

- Stefan Lohs, Reinhardt Karnapke, and Jörg Nolte. Link stability in a wireless sensor network - an experimental study. In 3rd International Conference on Sensor Systems and Software, 2012. [38]

Chapter 6

Future Work

As has been described in the evaluation section, the routing protocols were only simulated using a perfect behavior medium access control. The decision had been made because the number of variables had to be kept small in order to keep the results interpretable. In the future it might be interesting to investigate the interdependencies between MAC and routing in greater depth, by evaluating the developed routing protocols in fixed scenarios with a predefined percentage of unidirectional links, but varying MAC layers.

As for the routing protocols, all of them are working on neighborhood relations. It might be interesting to investigate a geographic version of Buckshot Routing (section 4.1). If the nodes need to know their location for application purposes, the location might also be used to make forwarding decisions, resulting in a geographically limited directional flooding. This would enable the omission of neighbor table and routing table.

Buckshot Routing might be augmented with a mechanism that reduces redundancy similar to a duplicate suppression algorithm, but used on the MAC layer. If the unique identities of messages that are used for duplicate detection were known to the MAC layer, messages that are still in the transmit queue might be dropped with a certain probability if the transmission of that message by another node is overheard.

Overhearing Supported BuckshotDV (section 4.3) could also be varied using probabilistic means. The size of a message might be decreased further, by having each node forward the message immediately with a certain probability. Those nodes which did not forward the message immediately could use a random delay. After this delay, they would transmit the message anyway, unless they overheard it being transmitted by another node during their waiting period, in which case the message would be dropped finally. While this method has the means to reduce the message size, it will probably have a small impact on delivery ratio. Whether that is tolerable depends on the application.

The presented protocols were developed to deal with the problem of unidirectional links. Some of them have also been designed to reduce network load, and thus to reduce energy consumption implicitly. But a real energy survey of the protocols is still missing. The ranking of the protocols might be different, if node sleep cycles are taken into account. Then, not all nodes would be awake and listening at the same time, which has a strong impact on the way flooding can be realized and on overhearing mechanisms.

As energy consumption is very important in battery powered sensor networks, the forwarding decisions of Buckshot Routing, BuckshotDV and OSBRDV could be modified to include the remaining energy of a node to a certain extent. Nodes could forward messages with a certain probability, based on their remaining energy or, if available, that of their neighbors.

In the real experiments, a strong influence of the network load on the delivery ratio has been seen. In experiments with two versions of BuckshotDV, the version that uses caching of overheard routes delivers many more messages than the version without overheard routes. This is the opposite of the results surveyed in the simulations, where overheard routes led to a decline of delivery ratio. The reason for this difference lies in the properties of the real hardware: The transceivers of the eZ430-Chronos were unable to handle the large number of messages generated. In the future it would be interesting to evaluate all protocols on different hardware. Also, ways should be found to reduce the number of transmitted messages.

Even though Buckshot Routing, BuckshotDV and OSBRDV use a limited directional flooding, the additional messages created thereby are not a big problem. The main concern is finding the initial route. As in most other on-demand routing protocols, this involves a flooding of the whole network with route request messages. Unlike most others, the route reply also uses a limited directional flooding. These two floodings represent a high network load. It would be interesting to replace this route discovery mechanism with a probabilistic or gossiping approach.

In the evaluation, the different routing protocols have shown different strengths and weaknesses. `Tree Routing` has proven to have an excellent cost per delivered data message, because the cost of a transmission failure is very small. In small networks it delivered about 40% of application messages using only a few retransmissions. If this delivery ratio can be tolerated by the application or the data is not time critical and data aggregation techniques for the last three to five samples could be used, `Tree Routing` would be a good choice. It could be possible to combine `Tree Routing` with BuckshotDV: The nodes that are within two or three hops from the sink use `Tree Routing` while those further from the sink need the robustness and resilience against

unidirectional links that BuckshotDV provides. This would result in less network load close to the sink, as `Tree Routing` produces only few messages. This reduction of network load close to the sink could lead to less congestion in high load scenarios and to a longer lifetime of the network: In sense-and-send scenarios, the nodes closest to the sink normally deplete their batteries first because they have to be awake more often in order to forward data messages. Reducing the network load near the sink can reduce this effect. On the nodes farther from the sink, the additional communication cost caused by the redundant transmissions is necessary to realize a certain delivery ratio.

ULC and ULTR suffered a number of problems due to the absence of a neighborhood discovery protocol. Passive neighborhood discovery does not work in single hop scenarios. As a destination never forwards RREQ messages, its neighbors never know that there is a bidirectional connection. Only when they receive the RREP, they realize that there is an unidirectional-incoming link from the sink to them, but they never realize that it is bidirectional. The passive detection of links used in the current implementation works only when the network load is high enough. But a high network load leads to MAC problems which should be avoided if possible. Also, it would lead to a high energy consumption which depletes the batteries of the sensor nodes faster. It would be interesting to evaluate the performance of ULC and ULTR with a neighborhood discovery protocol or a TDMA MAC layer which supplies the two-hop information it needs to calculate time slots and rounds to a cross layer data structure which can be accessed by the routing protocol, too.

The CSMA MAC used in the real world experiments presents a general problem for all evaluated protocols. CSMA MAC layers are known to have a bad delivery ratio when the network load is high. However, network load is especially high when protocols flood messages through the whole network during route discovery, making this phase once again the most vulnerable part of the surveyed routing protocols. In the future it would be interesting to evaluate the influence of different types and categories of MAC-layer. For example D-MAC [39] with its slot allocation scheme that has been optimized for sense-and-send scenarios should be evaluated. The slot allocation scheme tries to minimize latencies from leafs to the sink and therefore reduces the risk of message loss due to congestion. But D-MAC would have to be modified somewhat, as it does not allow message generation on intermediate nodes, only on leaf nodes. Other MAC protocols described in section 3.1 should also be considered.

Even though one of the design goals for the protocols developed for this thesis was simplicity, there are still a number of parameters to configure the protocols. ULC and ULTR have a strong dependence on protocol parameters, and especially the link

timeout. When it is set too low, links are deleted faster than they can be detected, if it is too high, link information if often outdated. This problem is closely related to the network load and the problem of passive detection of links. But different values for the other protocol parameters should be evaluated, too.

While the number of real world routing experiments was high and the way they were conducted was very time consuming, more connectivity measurement experiments should be made. Currently, there are only two sets of data for each scenario described in section 5.3, which could be included in the simulations. While the developed simulation model delivered results close to those measured in the real world routing experiments, it would be good to have more empirically gathered connectivity matrices which could be compared to the generated ones.

To further evaluate the developed simulation model, criteria should be defined which can be used to evaluate the similarity between generated and measured matrices. Also, the simulation could be enhanced with different MAC layers and properties of real hardware. While the current version is good at simulating unidirectional links and often changing communication neighborhoods, MAC layer issues are ignored. Including the problems caused by MAC and other properties of the used hardware would enable better simulations, but also make the simulations hardware dependent. This dependency is the reason why MAC layer issued were not simulated in this thesis. Also, developing realistic models for different hardware and MAC layers is enough work for a thesis on its own.

Chapter 7

Conclusion

In this thesis, unidirectional links in wireless sensor networks were reviewed. A literature study (see chapter 2) as well as own experiments with real sensor network hardware (section 5.3) have shown that these unidirectional links are common, especially in low power, low cost devices that are typically used in wireless sensor nodes. Literature also shows that unidirectional links often have a far greater reach than bidirectional ones, resulting in massive problems for protocols that use shortest paths measured in hop count. Therefore, most protocols try to operate on bidirectional links only, and eliminate the implications of unidirectional links as much as possible.

The impact of unidirectional links on medium access control and routing protocols has been described in theory, before selected state of the art MAC and routing protocols were discussed concerning their ability to handle unidirectional links. In this theoretical discussion it was already apparent that most state of the art protocols are only able to deliver a good performance if unidirectional links are rare. The few protocols that can make explicit use of unidirectional links produce a lot of overhead. Literature suggests that the overhead induced by making unidirectional links usable is too high, and therefore only bidirectional links should be used [45].

This thesis does not deny that the protocols surveyed in [45] induce a lot of overhead, but draws a different conclusion from that fact. When unidirectional links are common but the costs of using them explicitly are too high, the costs must be reduced or the unidirectional links made usable implicitly.

Following this line of reasoning, three new routing protocols, Buckshot Routing, BuckshotDV and OSBRDV were designed which use unidirectional links implicitly. Also, two other routing protocols, ULC and ULTR were designed which make use of unidirectional links explicitly, but should only be used when the information about the links is already available. This could for example be the case when the application

needs a neighborhood discovery protocol, or when a TDMA MAC was used that makes the information about the two hop neighborhood it needs to define slots and frames available to the routing protocol. As no such protocol was available in the evaluation, a passive link detection scheme was devised which can deliver the required information under certain circumstances.

To evaluate the five designed protocols, a simulation model for unidirectional links and link changes was designed and implemented (section 5.1) for the network simulator OMNeT++ [74] with the MiXiM extension [33, 46]. Using this model, the simulation is fed with connectivity matrices represented in change lists, which were generated before the simulation and could therefore be used for each protocol. The used simulation model is based purely on this connectivity input, which replaces the physical layer of MiXiM. It is used solely to analyze the ability of a routing protocol to cope with unidirectional links and link changes, the generated network load is not analyzed. Therefore, a perfect behavior MAC was used.

Four protocols were chosen as competitors for the evaluation: AODV_BR [37], DSR [28], `Tree Routing` and a simple `Flooding`. AODV-BR uses so-called backup routing to salvage data messages when the next hop on the route is unavailable. The reason for this unavailability could, among others, be a link that has turned unidirectional-incoming. The salvaging is realized by forwarding the message that could not be delivered to the intended next hop to neighbors that have overheard the transmission of the route reply that was used to build the currently unavailable route. These neighbors then forward the message to the node from which they received the route reply, enabling the message to take a one hop detour. DSR was one of the first protocols that take unidirectional links into account by flooding the network twice during route discovery, instead of using the inverse route of the route request message for the route reply. These two protocols have been developed for MANETs, but have been chosen for comparison nonetheless. The reason for this is the fact, that to the best of the author's knowledge no routing protocols for wireless sensor networks exist that use unidirectional links. `Tree Routing` remains one of the most commonly used protocols for wireless sensor networks, and was also included in the evaluation therefore. To compensate for link changes, `Tree Routing` uses up to two retransmissions on each hop. In contrast to DSR, which also uses these two retransmissions, `Tree Routing` only discards a message after three unsuccessful transmission attempts, whereas DSR generates a route error message and initiates a new route discovery for the next message. `Flooding` is the most simple of communication patterns, it only needs a duplicate suppression. Also, if a route from source to destination exists at all, `Flooding` uses it as it uses all available

paths. This is the reason why `Flooding` was used in the simulations to determine the maximum possible number of delivered data messages, as the generated matrices could theoretically include network separation.

All nine protocols were implemented in C++, making the implementation independent from the used hardware and operating system. This independence was shown by using the same implementation for the simulations in OMNeT++ and the real world experiments on eZ430-Chronos sensor nodes from Texas Instruments [15].

To show the relevance of using unidirectional links in routing protocols for wireless sensor networks, real world experiments were performed. In the first row of experiments, the connectivity between nodes was evaluated in four different locations and on two different radio channels. The results confirm that unidirectional links are indeed common, even more so than literature suggested. The experiments revealed an average number of more than 100 link changes per round (minute) in a network consisting of only 36 nodes. The number of unidirectional links was always higher than the number of bidirectional ones, with an average ratio between 4 and 5 to 1. There were always more than twice as many unidirectional links than bidirectional ones, and in one case the ratio even reached 91 to 1 as one connectivity graph contained 91 unidirectional links and only a single bidirectional one. Therefore, using unidirectional links implicitly in order to achieve robustness against quick changes in (logical) network topology was the right choice.

The evaluation of the protocols consisted of three different scenarios: A sense-and-send application (section 5.4), a single pairing application where each node transmitted to a random partner node throughout the whole experiment or simulation (section 5.5) and a multiple pairings application in which each node changed its communication partner after each fifth transmitted message (section 5.6).

In the sense-and-send scenario 4950 simulations with grids of nodes consisting of 100 (10x10), 400 (20x20), 900 (30x30) or 1600 (40x40) nodes and different destinations were conducted for each protocol. As four versions of Buckshot Routing and two versions of DSR were simulated, this amounts to 64350 simulations with a runtime between five minutes and more than a day. For the other two application scenarios, the number of simulations was smaller because they already included changing destinations in the application, making the additional variation in the simulations unnecessary.

Simulation results show that all related work protocols suffer heavily from the existence of unidirectional links and from link instability once the network size and thus the length of possible routes increases. However, the protocols designed in this thesis show a very good delivery ratio, which even increases for larger network sizes. This is

due to the fact that more redundant routes become available in larger networks. Up to 97% delivery ratio were reached in scenario one. Buckshot Routing is the only one of the developed protocols whose performance declines for larger networks, which is due to its source routing nature: The path that should be taken by messages is determined at the source, making it vulnerable to link changes. But Buckshot Routing still performs far better than all related work protocols even in scenario one, even though it could not benefit from its greatest advantage of being able to create highly adaptive routes. Highly adaptive routes, as used by Buckshot Routing, BuckshotDV and OSBRDV, change the routing table entries with each received message, making them adaptive to link changes. This is only possible due to limited directional flooding, which uses multiple nearby paths for the same message. However, the sense-and-send scenario used in scenario one implied that messages were only transmitted in one direction, and thus nodes never received updates for their paths to the sink.That the developed protocols achieved such a high delivery ratio even though profiting from their greatest advantage was denied to them shows their robustness. In scenarios two and three, the highly adaptive routes could be used due to the change in application behavior.

After the importance of using unidirectional links was confirmed, all routing protocols were evaluated in four different real-world locations: A single hop environment on a desk, placed onto poles which created an environment with between one and two hops on average and two multihop deployments on the ground, a lawn and a stone pavement.

To compare the results achieved in the real world experiments with those of the simulations, the network consisting of 36 nodes that was used in the real world experiments was also simulated. The results for the stone pavement and lawn scenarios correlate with the simulations for most protocols. Only those that suffered most from timing problems, namely AODV-BR and ULTR, differ significantly. As has been known from literature, `Flooding` suffers from the broadcast storm problem, resulting in too many collisions due to MAC layer problems.

The results of all other protocols achieved in the real world experiments correspond fairly well to the simulation results, leading to the conclusion that the performance of the developed protocols would increase with network size. Seemingly the used network, while being fairly large for an evaluation deployment, was still too small to fully show the advantages of the developed protocols.

But even though the network was not large enough and the communication pattern was not optimal for them, the protocols developed in this thesis delivered a high number of messages.

Appendix A

Implementation Details

This section describes all choices that were made during the implementation of the five developed protocols and the four protocols used as reference. The first choice concerned the operating system that was used as basis, followed by the choice of simulator and hardware for the real experiments. After that, parameter settings and abstractions that were used are described before protocol specific details are discussed.

A.1 Reflex

Reflex [75, 76, 77, 79] is an operating system for deeply embedded systems and sensor nodes that has been developed by the distributed systems/operating systems group at Brandenburg University of Technology Cottbus, Germany. It is based on the event flow principle which removes the need for explicit synchronization within components [78]. Reflex is implemented in C++, which enables the application programmer to use state of the art object oriented programming methods. Also, this fact enables Reflex to be used on a range of different platforms, because all that is needed to deploy it is a C++ compiler and a few lines of assembler code for hardware specific drivers. To enable its use in wireless sensor networks, a power management scheme has been integrated [68] and is continuously being improved.

All protocols are implemented operating system independent (see section A.4), but an operating system has to be used nonetheless. For the reasons listed above, Reflex has been chosen.

A.2 OMNeT++

OMNeT++ [74] is a discrete event simulator that can be used to simulate different kinds of networks. OMNeT supplies a framework of modules which can be combined to form compound modules. Both types of modules contain gates, which can be connected using channels, to allow the modules to communicate with each other. This is done by passing messages from one module to the other. OMNeT is implemented in C++. which made the integration of Reflex into OMNeT possible [35].

To enable simulations of a sensor network, the MiXiM framework [33] has been used. It provides an abstraction for communication layers. Explicit simulation of unidirectional links using a connectivity matrix has been added to MiXiM for this thesis. For more details see section 5.1.1.

A.3 EZ430-Chronos

For all real world experiments, eZ430-Chronos Sensor nodes from Texas Instruments [15] were used. The eZ430-Chronos is an inexpensive evaluation platform for the CC430. It features an MSP430 micro controller with an integrated CC1100 sub-gigahertz (868MHz) communication module [9]. The evaluation board is delivered as a compact sports watch containing several sensors, e.g. a three-axis accelerometer, and five buttons which are connected through general purpose I/O pins. The sports watch casing has been removed in order to use the eZ430s as sensor nodes.

Figure 5.1 in section 5.2.4 shows the used eZ430-Chronos sensor nodes in three different placements which were used in the real world experiments. An external battery pack has been soldered to the nodes, which replaces the internal coin cells. This enables the usage of freshly charged batteries for each protocol.

Apart from the modification for the batteries, the sensor nodes were used as they were delivered, no calibration was made. This should reflect the fact that future users would neither be able nor willing to calibrate a large number of nodes. Instead, they are used ”out of the box”. The transmission power was also left at the preset level of 0 dBm, which lead to a small transmission range. This small transmission range is also due to the absence of a real antenna on the eZ430-Chronos: The metal surrounding the display acts as antenna.

A.4 Common Interfaces and Constants

To ensure that the implementation of the routing protocols is independent of the actual platform and the operating system they are used on, a wrapper class called `RoutingWrapper` has been created which implements the operating system dependent part so that the protocols can be independent.

Listing A.1: Class RoutingWrapper

```
class RoutingWrapper :public LocalBroadcastIF{
public:
 virtual void run();
 virtual void assign(reflex::Buffer* buffer);
 virtual void send(uint8 * message, uint8 length);

private:
 reflex::Sink1<reflex::Buffer* >        *lowerOut;
 nodeIdType id; ///< the unique node id
 RoutingProtocol* routingProtocol;
 RoutingApplication* routingApplication;
 ILogger* log;
};
```

Listing A.1 shows the most important functions and member variables of this wrapper. The `run()`-method is typical for the operating system Reflex, as it is called each time an activity is scheduled. It is used in the routing wrapper to call the `start()`-method of a routing protocol during initialization of the sensor network. After the initialization phase is finished, the `handleTimer()`-method of the used routing protocol is periodically called, which is used in different ways by each protocol (see below). Received messages are handed to the `RoutingWrapper` from a lower layer in the `assign()`-method. There, the `reflex::Buffer`, which is operating system specific, is copied into a simple array and handed to the routing protocol before the buffer is recycled.

The `RoutingWrapper` implements the interface `LocalBroadcastIF` which enables the transmission of an arbitrary number of bytes to all neighboring nodes (local broadcast). This is realized using the `send()`-method with its parameters: a pointer to a character array and the length of the message. Due to restrictions imposed by the operating system (reflex::Buffer cannot hold more than 255 bytes) the size of a message is restricted, and an unsigned character is large enough to describe the length. In the real world experiments the length is even further reduced because the used hardware can only handle messages which are at most 64 bytes in length. In the `send()`-method,

data logging also takes place. The `RoutingWrapper` has a member which implements the `ILogger`-interface, and which is informed of the transmission using the method `logTransmittedPacket(transmittingNode, type, size)`. This logging mechanism is hardware dependent, and differs between simulations where one central component can be used and the real world experiments were each node has its own logger.

In order to exchange the routing protocols encapsulated by the routing wrapper, each routing protocol implements the `RoutingProtocol`-interface (listing A.2).

Listing A.2: Interface RoutingProtocol

```
class RoutingProtocol{
public:
 virtual uint8 send(uint8* msg, uint8 length, nodeIdType dest)=0;
 virtual void receive(uint8 * message, uint8 length)=0;
 virtual void handleTimer()=0;
 virtual void start(){}

protected:
 nodeIdType id; ///< the unique node id
 Receive2IF* upperOut;
 LocalBroadcastIF* lowerOut;
};
```

The methods `send()`, `receive()` and `handleTimer()` are pure virtual functions and must be implemented by each routing protocol, while the `start()`-method has per default an empty implementation. `Tree Routing` is the only protocol that uses this method to build initial routing trees at the beginning of the simulations or experiments. All other implemented protocols (except for `Flooding`) are on-demand routing protocols and establish their routes only when the first messages should be transmitted.

The `send()`-method is used by an upper layer to deliver a pointer to a character array, the length of the message and its destination to the routing protocol. The length of such a message is limited by the protocol parameter `maxLengthRoutingInput`. In this case it is important to remember that the total size of a message, including routing information, may not exceed the maximum size of a `reflex::Buffer` (simulation:255 bytes, real experiments: 64 bytes) and `maxLengthRoutingInput` should be set accordingly. For other operating systems and other sensor node hardware this parameter might differ.

Messages are handed from the `RoutingWrapper` to the routing protocol using the `receive()`-method. There, the protocol decides which type the message has and handles it accordingly.

Table A.1: Handling of a Timeout by the implemented Routing Protocols

AODV-BR	one-hop retransmissions or transmission of route error messages
Buckshot Routing	ignored
BuckshotDV	ignored
DSR	one-hop retransmissions or transmission of route error messages
`Flooding`	ignored
OSBRDV	transmit deferred route reply or data messages
`Tree Routing`	one-hop retransmissions
ULC	remove inactive neighbors from neighbor table
ULTR	remove inactive neighbors from neighbor table

Each protocol reacts differently when its `handleTimer()`-method is called. Some retransmit stored messages, others delete neighborhood information. Table A.1 gives a brief overview.

The member variables `upperOut` and `lowerOut` are used to connect to the upper and lower layers, respectively. In the current implementation the upper layer is the `RoutingApplication` while `RoutingWrapper` represents the lower layer. The `id` is used to decide if received messages should be forwarded, discarded or handed to the upper layer.

The class `RoutingApplication` is used to simulate different applications (listing A.3). At the beginning, the member variable `msgToSend` holds the number of application messages that need to be transmitted. When the `RoutingWrapper` calls the `handleTimeout()`-method of the application, a new message is generated and handed to the `routingProtocol`. As the payload is not really important in this scenario, the message only contains ten times the `id` of this node. Therefore the payload has a size of 10 or 20 bytes, depending on the size of a `nodeIdType`. When the routing protocol hands a message to the `RoutingApplication` using the `receive()`-method, the `log` is used to store information about the received data message. The `log` is of type `ILogger`, and depends on the environment. In simulations, a global logging component was used which counts all received messages and is also called by the `RoutingWrapper` each time a message is transmitted to record the number of transmitted bytes (see section 5.1). In experiments with real hardware, no central component exists and each node has to store this information locally. It was retrieved after the end of each experiment.

Listing A.3: The Routing Application

```
class RoutingApplication: public Receive2IF
{
public:
 virtual void receive(uint8 * message, uint8 length);
 void handleTimeout();

private:
 RoutingProtocol* routingProtocol;
 nodeIdType id;
 int msgToSend;
 ILogger* log;
};
```

All implemented routing protocols use the same duplicate suppression. It is based on the identity of the originator of the message and a sequence number. However, the implementation of the duplicate suppression differs between simulations and real experiments, as the sensor node hardware does not offer very much storage space.

Due to the types of routes that are stored, some protocols can use the same type of routing table while others cannot. For example, Buckshot Routing and DSR are both source routing protocols and both use the same routing table. The same can be said about neighbor tables - some protocols need to store the same information and share an implementation while others do not. ULC needs to store information about the status of the links to its neighbors to make forwarding decisions, while Buckshot Routing only needs to know if a certain node is a neighbor at all. Table A.2 shows which protocol uses which implementation.

To keep the results comparable, only a single configuration file was used for all protocols: the `routingconf`. Table A.3 shows the parameters, their setting for the simulations and which protocol uses them. Those parameters that were changed for the real experiments are shown in table A.4 below.

MaxRouteLength describes the maximum allowed length of a route in hops. It is necessary for the routing tables and to guarantee that no messages larger than the maximum size imposed by the operating system (Reflex : 255 Bytes) and hardware (eZ430-Chronos nodes: 64 Bytes) are created. As only the source routing protocols use messages of variable size and need to store the whole route in their routing tables, this parameter is used only by `Buckshot Routing` and `DSR`. These two protocols were evaluated with two different settings: 15 hops and 40 hops limit.

Table A.2: Routing - and Neighbor Tables used by the implemented Routing Protocols

	routing table	neighbor table
AODV-BR	two instances of `RoutingTable_AODVBR`	none
Buckshot Routing	`RoutingTable_BR`	`Neighbortable`
BuckshotDV	`RoutingTable_BRDV`	`Neighbortable`
DSR	`RoutingTable_BR`	none
`Flooding`	none	none
OSBRDV	`RoutingTable_OSBRDV`	`Neighbortable`
`Tree Routing`	`RoutingTable_TreeRouting`	none
ULC	`RoutingTable_ULC`	`Neighbortable_ULC`
ULTR	`RoutingTable_ULTR`	`Neighbortable_ULTR`

Table A.3: Configuration Parameters used by the implemented Routing Protocols

Parameter	Setting	Used By
MaxRouteLength	15 or 40	Buckshot, DSR
NumNodes	100/400/900/1600	all (duplicate suppression)
duplicateEntries	100	all (duplicate suppression)
illegalID	NumNodes+1	BRDV,DSR,OSBRDV,Tree,ULC,ULTR
NumRTEntries	NumNodes	all except `Flooding` (routing tables)
MaxNeighbors	NumNodes	Buckshot,BRDV,OSBRDV,ULC,ULTR
maxLengthRoutingInput	80	all
nodeIdType	uint8 / uint16	all
maxRREQ	10	all except Tree and `Flooding`
maxDeferred	20	OSBRDV
deferredTime	25	OSBRDV
linkTimeout	5	ULC, ULTR
weightUnidirectional	2	ULC
weigthBidirectional	1	ULC
maxStored	NumNodes	AODV-BR, DSR, Tree
oneHopTimeout	25	AODV-BR, DSR, Tree
numberOfRetransmissions	2	DSR, Tree

NumNodes The number of nodes in the network.

duplicateEntries defines the number of sequence numbers stored for each node by the duplicate suppression. In combination with `NumNodes` it defines the RAM consumption of the duplicate detection and is needed by every protocol.

illegalID is used in routing tables as well as in some types of messages to mark entries as invalid.

NumRTEntries The number of entries in a routing table. As `Flooding` does not need a routing table, it is the only one that does not use `NumRTEntries`.

MaxNeighbors The number of entries in a neighbor table, it is used only by the protocols developed in this thesis.

maxLengthRoutingInput The maximum size of application data that can be handed to the routing protocol. Used by each protocol to calculate the maximum size of a DATA message.

nodeIdType The type of ID used for each node, either uint8 or uint16, depending on network size. It is also needed by all protocols. Together with `MaxRouteLength` and `maxLengthRoutingInput` it defines the largest size a message can grow up to.

maxRREQ is used by all protocols except `Tree Routing` and `Flooding`. It defines the highest number of route discoveries that might be carried out at the same time, as each application message must be stored until route discovery is completed. Once a route reply message was received and the application message transmitted, the storage space is available again.

maxDeferred is a parameter that is only used for `OSBRDV`. When a message is overheard by a node that is not on the direct path included in the message, it is deferred for a certain time. If the forwarding of that message is overheard during this time, there is no need to forward it anymore and it is discarded.

deferredTime The time in which the forwarding of a deferred message must be overheard in `OSBRDV` before it is transmitted. Measured in number of times the method `handleTimer()` must be called.

linkTimeout `ULC` and `ULTR` use the `linkTimeout` to remove incoming links from their neighbor lists.

weightUnidirectional used in `ULC` to determine whether routes that contain unidirectional links are preferred over those that do not.

weigthBidirectional used in `ULC` to determine whether routes that contain bidirectional links are preferred over those that contain unidirectional ones.

maxStored The number of DATA messages that can be stored for one hop retransmission. Used by `AODV-BR`, `DSR` and `Tree Routing`.

oneHopTimeout The number times `handleTimer()` must be called before a stored message is retransmitted or, if the maximum number of retransmissions was reached, a route error message is generated. Used by `AODV-BR`, `DSR` and `Tree Routing`.

numberOfRetransmissions The number of times a stored message gets retransmitted before it is discarded. As `AODV-BR` uses exactly one retransmission, this variable is only used by `DSR` and `Tree Routing`.

A.5 Parameter Differences between Simulations and Outdoor Experiments

Table A.4 shows the four parameters that were changed for the real experiments. All other parameters remained the same as in the simulations and described in table A.3.

Table A.4: Parameter differences between Simulations and real Experiments

Parameter	Setting	Used By
nodeIdType	uint8	all
duplicateEntries	5	all (duplicate suppression)
maxLengthRoutingInput	10	all
maxStored	15	AODV-BR, DSR, Tree

The nodeIdType was changed because of the network size: For a network consisting of 36 nodes a uint8 is sufficient, as it can hold up to 256 values. The number of duplicate entries has also been reduced, because of the network size. In a smaller network, it is highly improbable that more than one message from the same node is still being transmitted by some nodes. Therefore, it is more than sufficient to remember the last five sequence numbers seen from each node. Reducing the number of messages that can be stored as well as the length of the routing input leads to a large reduction of

memory consumption, as these two are multiplied to determine the space needed to store messages for retransmission. This reduction of memory consumption was necessary in order to fit DSR onto the EZ430-Chronos.

A.6 Protocol Specific Implementation Details

Even though the implementation of some protocols was straightforward, there are some aspects that emerged during the implementation of others. Here, these details are discussed.

A.6.1 AODV-BR

Even during implementation and the short simulations that were used only to validate the implementation, it became apparent that `AODV-BR` is not as robust as it seemed at first. The "fish bone" structure it uses to recover lost data packets works quite well once a bidirectional path between source and destination has been established. But as it uses the inverted path of the route request for the route reply message, and route reply messages are not salvaged by the fish bone structure, finding a working bidirectional path at the beginning proves to be the biggest challenge for `AODV-BR`.

A.6.2 DSR

In its original version, DSR has been specified with IPv4 addresses in mind. For this thesis, it has been altered to use `nodeIdType` instead. According to the specification, when unidirectional links are used, explicit layer 3 (routing) acknowledgment messages should be used for each hop. Because of these acknowledgment messages, the worst case scenario for `DSR` is an uninitialized network, in which a single message needs to be transmitted from one end of the network to the other. As no routes are known, and caching of overheard routes should not be used (according to the specification), each node on the route needs to start a route discovery for the last hop. As each route discovery includes two floodings of the whole sensor network, a route of length n would result in 2n+2 floodings.

Memory usage is problematic, too: all messages need to be stored for one hop retransmission until an acknowledgment message is received or the number of retransmissions has been exceeded.

As `DSR` creates a lot of messages, nodes soon ran out of network buffers in some preliminary simulations. Moreover, the order of operations is important, especially if network buffers are scarce. When a message has been received, its buffer is recycled.

Therefore, at least a single buffer must be available. If it is used to transmit an acknowledgment to the last hop, then the actual message might be lost because there is no buffer for it to be transmitted. This means that the received message is forwarded first, the acknowledgment is sent thereafter. If there once again is no available buffer, this leads to a retransmission of the message by the last hop which is undesirable, but at least the message reaches its destination. Even if enough network buffers are available, this reversed order means that finding a fitting value for `oneHopTimeout` is getting much more complicated. It can no longer simply be set to the approximate round trip time, because one or more other messages might get transmitted before the acknowledgment message.

In the simulations, all of this could be fixed by increasing the number of network buffers to 200 per node, but for a real deployment this seems hardly possible. On the other hand, the necessity to handle so many messages in such a short time should not arise in a real deployment.

A.6.3 OSBRDV

One of the most critical elements of `OSBRDV` is the timer. Starting a separate timer for each message that gets deferred would introduce much overhead, therefore a periodic timer is used. When a message is deferred, it is stored in an array, and its timeout set to `deferredTime` (25). The periodic timer is activated every 100 ms, meaning that a message is stored for 2400 - 2500 ms, depending on the moment it was stored. In the simulations, this is ample time for the intended next hop to process and transmit the message.

As OSBRDV depends on overheard messages to discard deferred ones, it must process each incoming message. Only after the list of deferred messages has been checked, duplicate suppression may take place.

The number of messages that can be stored is also critical, as the memory to store them must be allocated static in Reflex. This can lead to a huge amount of wasted memory, which is a problem for resource constrained sensor nodes. On the other hand, choosing the number too low would lead to a bad delivery ratio as messages that could not be stored are discarded.

According to the algorithm, nodes that are neighbors of the destination of a RREP or DATA message would wait for it to retransmit the message. As a destination does not normally forward the message, all its neighbors would retransmit the message even though it has already been received. To avoid this, received messages are retransmitted so that the neighboring nodes may discard their deferred messages.

There is a strict order in which the operations must be performed when receiving a RREP or DATA message in OSBRDV:

1. Check deferred messages
2. Search next but one hop in neighbor table
3. Discard duplicates
4. Check if route to destination is known
5. Check if message has to be deferred
6. Alter message and retransmit

A.6.4 `Tree Routing`

`Tree Routing` needs an initialization phase in which each node transmits a tree building message. The tree will be used throughout the lifetime of the network. Nodes store each DATA message they forward for a certain time. If they do not overhear the next hop transmitting the message, they retransmit it up to `numberOfRetransmissions` times in the hope that the link to the next hop has become available again. The timeout and the number of retransmissions are configuration parameters (see table A.3).

A.6.5 ULTR

Finding the right setting for the link timeout is absolutely critical for ULTR. If it is chosen too small, the implemented version of ULTR degenerates to a complicated version of Buckshot Routing. If it is set to high, link breaks are not detected and wrong routing decisions are made. No suggestion of a good value can be made here, because a good value depends on the network load and thus on the application. As the link status is checked implicitly with every transmission, an application that transmits often can set a lower timeout value than one that transmits only seldom. Still, even if the application transmits only rarely, the timeout should not be set too large, because link changes may still occur due to environmental causes.

When ULTR switches to Buckshot mode, it checks only if there is an incoming link from the next hop. If there had been an outgoing link, the message could have been transmitted in normal mode. This is due to the fact that, in the absence of a neighborhood detection protocol, only incoming and bidirectional links can be detected implicitly. Pure outgoing links are listed as unknown in the neighborhood table. The same is true for ULC.

A.6.6 ULC

Like ULTR, the implemented version of ULC works without a neighborhood discovery protocol. This leads to a suboptimal performance right after the start of the application. When the first message needs to be delivered, route discovery is started by transmitting the first RREQ message. All nodes that receive this message consider the link over which they have received it to be unidirectional-incoming, because no outgoing links have been passively detected yet.

As multiple RREQ messages will be received by a destination one after the other, multiple RREP messages are transmitted, if the weight of the path stored in the messages received later is preferable to that of the earlier received ones. The route in the routing table is changed accordingly.

ULC suffers from the same timeout problem as ULTR. If the timeout is set too high, messages are discarded ineffectively. If it is set too low, ULC will become a weighted version of Buckshot Routing. The problem of message loss in normal mode could be solved by storing the message in intermediate nodes. If the link to the next hop is assumed to be bidirectional, it is safe to assume that the forwarding of the stored message will also be overheard. If it is not overheard, the message could be transmitted in Buckshot mode. However, this is not implemented as it would lead to a higher memory consumption.

A.7 Own publications referenced in this chapter

- Karsten Walther, Reinhardt Karnapke, and Jörg Nolte. An existing complete house control system based on the reflex operating system: Implementation and experiences over a period of 4 years. In Proceedings of 13th IEEE Conference on Emerging Technologies and Factory Automation, 2008. [76]
- Karsten Walther, Reinhardt Karnapke, André Sieber, and Jörg Nolte. Using preemption in event driven systems with a single stack. In The Second International Conference on Sensor Technologies and Applications, Cap Esterel, France, 2008. [77]

Glossary

AI-LMAC The MAC protocol AI-LMAC. 35

AMAC A MAC protocol that uses tunneling to make unidirectional links usable on the MAC layer. 32, 33

AODV Ad hoc On-Demand Distance Vector Routing. 51–53

AODV-BR Backup Routing in Ad hoc Networks. 53

Buckshot Routing A source routing protocol that was developed for this thesis. It uses unidirectional links implicitly by creating multiple paths. 86–88, 90–94

BuckshotDV A distance vector version of Buckshot Routing that was developed for this thesis. 95–99

BW_RES A modified RTS/CTS reservation scheme. 31

D-MAC A combination of MAC and tree routing protocol. 42

Directed Diffusion A diffusion based routing protocol for wireless sensor networks. 67–69

DSDV Highly Dynamic Destination-Sequenced Distance-Vector Routing. 49, 50

DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. 60, 61, 63

DYMO Dynamic MANET On-demand Routing. 58–60

ECTS-MAC Extended Clear To Send MAC. 31, 32

GEAR Geographical and Energy Aware Routing. 79

GeoTORA A geographic version of TORA. 75–77

GPSR Greedy Perimeter Stateless Routing. 77–79

LBSR The Loop Based Source Routing protocol. 66

LMAC The MAC protocol LMAC. 34, 35

MLMAC An enhancement of LMAC for mobile sensor nodes. 35–39

MLMAC-UL An enhancement of MLMAC that utilizes unidirectional links. 39, 40

MMP The Multicast MAC Protocol. 29, 30

NMAC The Neighbor MAC. 30

NST-AODV Not So Tiny - AODV. 55

OSBRDV An overhearing supported version of BuckshotDV that was developed for this thesis. 100–103

PANAMA Pair wise Link Activation and Node Activation Multiple Access. 33

Rumor Routing An agent based routing protocol for wireless sensor networks. 71, 73

Solar Aware Routing A diffusion based routing protocol that prefers nodes that scavenge energy. 69, 70

SPEED A geographical routing protocol for real-time applications. 80, 81

TORA Temporally-Ordered Routing Algorithm. 73–75

ULC Unidirectional Link Counter (ULC) was developed for this thesis. It assigns weights to paths based on the number of unidirectional and bidirectional links contained therein. 110–112, 114

ULTR Unidirectional Link Triangle Routing (ULC) was developed for this thesis. It uses a detour of one hop to bypass a unidirectional incoming link, building a triangle of links. 105–107, 109

Bibliography

[1] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Elsevier Ad Hoc Network Journal, Vol. 3/3*, pages 325–349, 2005.

[2] J.N. Al-Karaki and A.E Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE Volume: 11, Issue: 6*, pages 6– 28, Dec 2004.

[3] Tomonori Asano, Hiroyuki Unoki, and Hiroaki Higaki. Lbsr: Routing protocol for manets with unidirectional links. In *18th International Conference on Advanced Information Networking and Applications (AINA'04) Volume 1*, page 219, 2004.

[4] Rendong Bai and Mukesh Singhal. Salvaging route reply for on-demand routing protocols in mobile ad-hoc networks. In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–62, New York, NY, USA, 2005. ACM Press.

[5] Lichun Bao and J. J. Garcia-Luna-Aceves. Channel access scheduling in ad hoc networks with unidirectional links. In *DIALM '01: Proceedings of the 5th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 9–18, New York, NY, USA, 2001. ACM.

[6] S. Bradner. Key words for use in rfcs to indicate requirement levels, http://tools.ietf.org/html/rfc2119.

[7] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM Press.

[8] Costas Busch, Srikanth Surapaneni, and Srikanta Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *SPAA '03: Proceedings*

of the fifteenth annual ACM symposium on Parallel algorithms and architectures, pages 210–219, New York, NY, USA, 2003. ACM Press.

[9] Texas instruments cc430f6137, `http://focus.ti.com/docs/prod/folders/print/cc430f6137.html`.

[10] I. Chakeres and C. Perkins. Dynamic manet on-demand (dymo) routing, http://tools.ietf.org/html/draft-ietf-manet-dymo-18, feb 2010.

[11] circuit design inc. low power radio solutions. www.lprs.co.uk/main/product.info.php?productid=154.

[12] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.

[13] Tuan Le Dinh, Wen Hu, Pavan Sikka, Peter Corke, Leslie Overs, and Stephen Brosnan. Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In *LCN '07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 799–806, Washington, DC, USA, 2007. IEEE Computer Society.

[14] E. Duros, W. Dabbous, H. Izumiyama, N. Fujii, and Y. Zhang. A link-layer tunneling mechanism for unidirectional links, http://www.faqs.org/rfcs/rfc3077.html, Mar 2001.

[15] Texas instruments ez430-chronos, `http://focus.ti.com/docs/toolsw/folders/print/ez430-chronos.html?DCMP=Chronos&HQS=Other+OT+chronos`.

[16] Internet Engeneering Task Force. http://www.ietf.org/.

[17] C. Gomez, P. Salvatella, O. Alonso, and J. Paradells. Adapting aodv for ieee 802.15.4 mesh sensor networks: Theoretical discussion and performance evaluation in a real environment. In *WOWMOM '06: Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, pages 159–170, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Hrishikesh Gossain, Nagesh Nandiraju, Kumar Anand, and Dharma P. Agrawal. Supporting mac layer multicast in ieee 802.11 based manets: Issues and solutions.

In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 172–179, Washington, DC, USA, 2004. IEEE Computer Society.

[19] Global system for mobile communication, http://gsmworld.com.

[20] T. He, J. Stankovic, C. Lu, and T. Abdelzaher. Speed: A real-time routing protocol for sensor networks. Technical report, University of Virginia Tech. Report CS-2002-09, 2002.

[21] Tian He, Sudha Krishnamurthy, Liqian Luo, Ting Yan, Lin Gu, Radu Stoleru, Gang Zhou, Qing Cao, Pascal Vicaire, John A. Stankovic, Tarek F. Abdelzaher, Jonathan Hui, and Bruce Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[22] C. Hedrick. Routing information protocol, http://tools.ietf.org/html/rfc1058.

[23] C. Hedrick. Texas instruments cc2420, http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[24] John Heidemann, Fabio Silva, and Deborah Estrin. Matching data dissemination algorithms to application requirements. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 218–229, New York, NY, USA, 2003. ACM Press.

[25] Guido R. Hiertz, Sebastian Max, Yunpeng Zang, Thomas Junge, and Dee Denteneer. Ieee 802.11s mac fundamentals. *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, 0:1–8, 2007.

[26] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM.

[27] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[28] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[29] D. Johnson, H. Yu, and D. Maltz. The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4, https://tools.ietf.org/html/rfc4728.

[30] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[31] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM.

[32] Young Ko and Nitin H Vaidya. Geotora: A protocol for geocasting in mobile ad hoc networks. Technical report, College Station, TX, USA, 2000.

[33] A. Koepke, M. Swigulski, K. Wessel, D. Willkomm, P.T. Klein Haneveld, T.E.V. Parker, O.W. Visser, H.S. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++: The MiXiM vision. In *1st Int. Workshop on OMNeT++*, mar 2008.

[34] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 64–75, New York, NY, USA, 2005. ACM.

[35] Andreas Lagemann and Jörg Nolte. Integration of event-driven embedded operating systems into omnet++ – a case study with reflex. In *2nd International Workshop on OMNeT++*, Rome, Italy, March 2009.

[36] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proc. 14th Intl. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2006.

[37] Sung-Ju Lee and Mario Gerla. AODV-BR: Backup routing in ad hoc networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2000)*, Chicago, IL, Sepember 2000.

[38] Stefan Lohs, Reinhardt Karnapke, and Jörg Nolte. Link stability in a wireless sensor network - an experimental study. In *3rd International Conference on Sensor Systems and Software*, 2012.

[39] Gang Lu, Bhaskar Krishnamachari, and Cauligi S. Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in sensor networks. In *In Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN*, 2004.

[40] Gang Lu, Bhaskar Krishnamachari, and Cauligi S. Raghavendra. An adaptive energy-efficient and low-latency mac for tree-based data gathering in sensor networks: Research articles. *Wirel. Commun. Mob. Comput.*, 7(7):863–875, 2007.

[41] Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. An adaptive tdma based mac protocol for mobile wireless sensor networks, best paper award. In *International Conference on Sensor Technologies and Applications*, 2007.

[42] Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. Mac protocols for wireless sensor networks: Tackling the problem of unidirectional links. In *International Journal on Advances in Networks and Services, vol 2 no 4*, pages 218 – 229, 2009.

[43] Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. Mlmac - an adaptive tdma mac protocol for mobile wireless sensor networks. In *Ad-Hoc & Sensor Wireless Networks: An International Journal, Vol.8 Nr.1-2*, 2009.

[44] Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. Mlmac-ul and ects-mac - two mac protocols for wireless sensor networks with unidirectional links. In *Third International Conference on Sensor Technologies and Applications*, Athens, Greece, 2009.

[45] Mahesh K. Marina and Samir R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '02, pages 12–23, New York, NY, USA, 2002. ACM.

[46] Rex Min, Manish Bhardwaj, Seong-Hwan Cho, Amit Sinha, Eugene Shih, Alice Wang, and Anantha Chandrakasan. An Architecture for a Power-Aware Distributed Microsenseo Node. In *In Proceedings the IEEE Workshop on signal processing systems (SIPS'00), October 2000.*, 2000.

[47] Rex Min and Anantha Chandrakasan. A framework for energy-scalable communication in high-density wireless networks. In *ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design*, pages 36–41, New York, NY, USA, 2002. ACM.

[48] Rex Min and Anantha Chandrakasan. Mobicom poster: top five myths about the energy consumption of wireless communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(1):65–67, 2003.

[49] Erik Nordström, Per Gunningberg, Christian Rohner, and Oskar Wibling. Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. In *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 29–34, New York, NY, USA, 2007. ACM.

[50] Stefan Nürnberger, Reinhardt Karnapke, and Jörg Nolte. Sensorium - an active monitoring system for neighborhood relations in wireless sensor networks. In *Second International Conference on Ad Hoc Networks*, 2010.

[51] Unidirectional Link Routing Group of the IETF. http://www.udcast.com/udlr/.

[52] Jorge Ortiz and David Culler. Multichannel reliability assessment in real world wsns. In *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 162–173, New York, NY, USA, 2010. ACM.

[53] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 1405, Washington, DC, USA, 1997. IEEE Computer Society.

[54] Stefan Penz. A dsr extension for connection stability assessment in mobile adhoc networks. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 509–513, Washington, DC, USA, 2007. IEEE Computer Society.

[55] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, http://www.ietf.org/rfc/rfc3561.txt, 2003.

[56] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, New York, NY, USA, 1994. ACM Press.

[57] Charles E. Perkins and Elizabeth M. Royer. "ad hoc on-demand distance vector routing.". In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA*, pages 90–100, FEB 1999.

[58] David Peters, Reinhardt Karnapke, and Jörg Nolte. Buckshot routing - a robust source routing protocol for dense ad-hoc networks. In *Ad Hoc Networks Conference 2009*, Niagara Falls, Canada, 2009.

[59] Neeraj Poojary, Srikanth V. Krishnamurthy, and Son Dao. Medium access control in a network of ad hoc mobile nodes with heterogeneous power capabilities. In *in: IEEE International Conference on Communications (ICC), 2001*, pages 872–877, 2001.

[60] Ravi Prakash. A routing algorithm for wireless ad hoc networks with unidirectional links. *Wirel. Netw.*, 7(6):617–625, 2001.

[61] V. Ramasubramanian, R. Chandra, and D. Mosse. Providing a bidirectional abstraction for unidirectional ad hoc networks, 2002.

[62] Bor rong Chen, Kiran-Kumar Muniswamy-Reddy, and Matt Welsh. Lessons learned from implementing ad-hoc multicast routing in sensor networks. Technical report, November 2005.

[63] Bor rong Chen, Kiran-Kumar Muniswamy-Reddy, and Matt Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 87–94, New York, NY, USA, 2006. ACM.

[64] P.J.M. Havinga S. Chatterjea, L.F.W. van Hoesel. Ai-lmac: An adaptive, information-centric and lightweight mac protocol for wireless sensor networks.

[65] Lifeng Sang, Anish Arora, and Hongwei Zhang. On exploiting asymmetric wireless links via one-way estimation. In *MobiHoc '07: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 11–21, New York, NY, USA, 2007. ACM Press.

[66] Jochen Schiller. *Mobile Communications*. Addison Wesley, 2003.

[67] Jochen Schiller, Achim Liers, Hartmut Ritter, Rolf Winter, and Thiemo Voigt. Scatterweb - low power sensor nodes and energy aware routing. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.

[68] André Sieber, Karsten Walther, Stefan Nürnberger, and Jörg Nolte˙ Implicit sleep mode determination in power management of event-driven deeply embedded systems. In *7th International Conference on Wired / Wireless Internet Communications*, University of Twente, The Netherlands, 2009.

[69] I. Stojmenovic. Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions [topics in ad hoc and sensor networks]. *IEEE Communications Magazine*, 46(12):102–107, 2008.

[70] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2003.

[71] Turau, Renner, and Venzke. The heathland experiment: Results and experiences. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks.*, Jun 2005.

[72] Volker Turau. The heathland experiment: Results and experiences, presentation. www.tm.uka.de/forschung/spp1140/events/kolloquium/2005-11/turau.pdf.

[73] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *INSS, Japan*, Jun 2004.

[74] András Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.

[75] Karsten Walther, Reinhard Hemmerling, and Jörg Nolte. Generic trigger variables and event flow wrappers in reflex. In *ECOOP - Workshop on Programming Languages and Operating Systems*, Jun 2004.

[76] Karsten Walther, Reinhardt Karnapke, and Jörg Nolte. An existing complete house control system based on the reflex operating system: Implementation and experiences over a period of 4 years. In *Proceedings of 13th IEEE Conference on Emerging Technologies and Factory Automation*, 2008.

[77] Karsten Walther, Reinhardt Karnapke, André Sieber, and Jörg Nolte. Using preemption in event driven systems with a single stack. In *The Second International Conference on Sensor Technologies and Applications*, Cap Esterel, France, 2008.

[78] Karsten Walther and Jörg Nolte. Event-flow and synchronization in single threaded systems. In *Proceedings of First GI/ITG Workshop on Non-Functional Properties of Embedded Systems (NFPES)*, Mar 2006.

[79] Karsten Walther and Jörg Nolte. A flexible scheduling framework for deeply embedded systems. In *In Proc. of 4th IEEE International Symposium on Embedded Computing*, 2007.

[80] Guoqiang Wang, Damla Turgut, Ladislau Bölöni, Yongchang Ji, and Dan C. Marinescu. A simulation study of a mac layer protocol for wireless networks with asymmetric links. In *IWCMC '06: Proceeding of the 2006 international conference on Communications and mobile computing*, pages 929–936, New York, NY, USA, 2006. ACM Press.

[81] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[82] Suyoung Yoon, Rudra Dutta, and Mihail L. Sichitiu. Power aware routing algorithms for wireless sensor networks. In *ICWMC '07: Proceedings of the Third International Conference on Wireless and Mobile Communications*, page 15, Washington, DC, USA, 2007. IEEE Computer Society.

[83] Xin Yu. Distributed cache updating for the dynamic source routing protocol. *IEEE Transactions on Mobile Computing*, 05(6):609–626, 2006.

[84] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical report, 2001.

[85] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 1–13, New York, NY, USA, 2003. ACM Press.