BACKBONE CONSTRUCTION BASED ON SYNC-MESSAGES AND ENERGY SAVINGS IN WIRELESS SENSOR NETWORKS

Master Thesis
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Reto Zurbuchen
2009

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Abstract

In the past few years sensor networks came increasingly into scope of interest. One of the main fields of research of these kinds of networks is to save energy while running to extend the lifetime of the battery-operated sensors. Several approaches to increase the lifetime of the batteries of each mobile host (also called node) have been proposed. One way to save energy is to turn off the radio periodically. This can be done on the *Medium Access Control (MAC)* layer. Currently well known MAC protocols such as *IEEE 802.11, GMS, UMTS* are not energy-efficient but optimised to achieve maximum throughput. That means that there is need for new energy-optimised protocols in the MAC- and network layer. In this master thesis two versions of a SYNC-message-based backbone construction mechanism for efficient MAC are presented for static networks. Both approaches establish a virtual backbone tree with a *Connected Dominating Set (CDS)*. This CDS is realised without any additional packet exchange. It only adds some information to existing packets of a well-known sensor network protocol called T-MAC. Besides that, a new approach of a global synchronisation of all nodes of the network is described and evaluated. The focus is on static networks.

Contents

Chapter 1

Introduction

A sensor network is a special kind of a wireless network. Sensor nodes implement a radio, several sensors and a small processing unit. The entire node is battery operated. A high error-rate and malfunctions of some nodes are expected. Therefore, a sensor network requires high redundancy. To achieve high redundancy a high density of the network is required. Another effect of a large number of deployed nodes is the need for dynamic topology control and routing. At the moment a node is deployed it should start sensing and try to establish routing tables at each node. Otherwise, it could take too much time until a node is able to route a packet the first time. The lifetime of a network is limited by the battery lifetime of its nodes. Therefore, energy-saving approaches such as the ones proposed in this thesis are required. The main application of such networks is environmental monitoring of a specific area.

1.1 Problem Statement

One of the main energy consumers of a sensor node is its radio. At the moment the radio is off, the total energy usage of the node is decreasing by a factor of around 100 [1]. This factor varies depending on the type of hardware of the node. The challenge is to find a way to temporally turn off the radio of some nodes while keeping the entire network running. A good technique will keep the maximum bandwidth high and the latency low as long as there are many data packets waiting for transmission. At the moment there are only a few data packets waiting for transmission, the maximum bandwidth can be lowered. All these improvements have to be done at the MAC and network layers of the communication protocol stack.

In this thesis the focus is on flat, unstructured and unsupervised networks. There are several challenges for a new communication protocol for such a wireless sensor network:

- The hardware of the node has to be cheap. Thus, the hardware is often of lower quality and the accuracy of the clock is accordingly less exact, which leads to some clock drifts. To detect that a sender node and its particular receiver node are awake at the same time, they need bo be synchronized.

- Like in any other wireless network, the well-known hidden node problem has its impact on the protocol [2].

- The entire network needs to organise the routing and an optional time synchronization itself. There is no master node (apart from the base station) and there is no hierarchy except of a possible MAC address for each node. Setting up static routing tables for all nodes is not feasible because of the large number of deployed nodes. Additionally, the deployment of the nodes might be unsupervised.
- Like for other wireless protocols there are some variable background noises on the radio. There is no guarantee that a packet arrives at the destination even if there is no concurrent communication.

1.2 Contributions

The main contributions of our work are:

- Medium access has been optimized by exploiting the information distributed through SYNC messages. The information has been used to achieve local synchronization and to implement routing on the MAC layer.
- A simple synchronization method for SYNC-based MAC protocols has been developed. The method prevents the existence of multiple schedules without introducing additional control traffic.
- The common schedule is realized by a gravitation-like principle, where temporary present clusters attract each other until they fuse. Due to the usage of SYNC messages, the fusion process is very robust. No temporary storage of multiple schedules is necessary.
- Virtual routing backbones have been established on the MAC layer. This enables the routing over this virtual backbone without additional routing control traffic and memory allocation.
- The backbones support routing and allow the temporary shut-down of the radios of non-backbone nodes to preserve additional energy.

1.3 Thesis Outline

This thesis is organized as follows: Chapter 2 describes existing protocols for wireless sensor networks. After that Chapter 3 explains the idea of a *Connecting Dominating Set (CDS)*, which is an approach to setup a virtual backbone of a network. A virtual backbone is helpful to optimise routing and to save energy. Chapter 4 introduces existing techniques to synchronize the clocks of sensor nodes. The simulations of the presented protocols and a reference protocol use the simulation environment called OMNeT++ [3]. A brief overview of this tool is given in Chapter 5. Based on the ideas of two existing MAC protocols and the theory of CDS, two new versions of a SYNC-based MAC protocol are described in Chapter 6. For synchronization purpose, both protocols apply a new synchronisation concept called *Local Adaptive Clock Assimilation Scheme (LACAS)* which is presented in Chapter 7. The functionality, implementation details and the results of the simulations in comparison to an existing MAC protocol are described in Chapter 8. Finally, Chapter 9 contains the conclusions and Chapter 10 contains a list of possible future work.

Chapter 2

Medium Access Control Protocols (MAC) for Wireless Sensor Networks

Transmitting information over a network of any kind is a challenging task. Therefore, there is a de facto standard to handle this task with several layers. Each layer is responsible for specific operations. The lowest layer is called the physical layer. It is responsible to translate the stream of bits, which has to be transmitted, into a sequence of signals which are transmitted over the physical connection to the destined destination. At the destination the receiver retranslates the sequence of signals back into a stream of bits and finally sends it to the upper layer called MAC layer.

The MAC layer coordinates the access to the communication medium. The main goal of classic MAC layer protocols is to maximise the data throughput and minimise the latency. In addition, the MAC layer of a wireless sensor network needs to minimise the energy usage of all nodes. As mentioned in the introduction, one of the most efficient ways to save energy is to turn off the radio periodically. On the other hand, if the radio of a specific node is off, there is no way to communicate with this node. Therefore, it has to be ensured that sender and receiver are awake at the same time, i.e., they need to be synchronized.

There are five major sources of energy waste in wireless sensor networks [4]. These are idle listening, overhearing, control packet overhead, collisions and overemitting.

- Idle listening
  Idle listening is the situation when a node is listening to the radio but there is no communication running. Typical nodes such as the Berkeley motes [5] or the ESB (Embedded Sensor Board) from Scatterweb [1] use in idle mode 50 - 100% of the amount of energy that is used in the receiving mode. All idle listening is huge waste of energy.

- Overhearing
  A node is listening to the communication on the channel but the node is not involved in this communication. There is no need to listen to this communication.

- Control packet overhead
  Each transmission of data consumes energy. On the other hand, all MAC protocols need control packets. Usually, an energy and maximum-bandwidth optimised MAC protocol tries to reduce the number of control packets as much as possible.

- Packet collisions
  Packet collisions are waste of energy because the senders of the packets need to be able to detect them and have to resend the packets again.

- Overemitting
  Overemitting occurs if a node sends data to another node and the receiver node is not ready to listen. The sender has to resend the data again.

In wireless networks all nodes share the medium. Therefore, the MAC layer has to control the medium access. In general, there are two different categories of MAC protocols to control the medium access and also to reduce the according sources of energy waste.

- Contention-based approaches
  In contention-based protocols, each node can send its data at arbitrary time. If more than one node wants to send data, they have to compete for the medium. Each node with data must wait a random time (according to the contention window) until it starts the transmission. The winner of this contention sends its data and all others have to wait. During this competition there is high risk of packet collisions because everybody is trying to obtain the channel. As soon as the contention is over, the risk of a packet collision is low because all listening nodes have overheard the winning node and delay their transmission therefore. The most widespread protocol for a wireless network of this type is *IEEE 802.11* [6]. Beside IEEE 802.11 there are numerous MAC protocols which belong to this category. In this thesis we focused on *S-MAC* [7] and *T-MAC* [8]. Both protocols are described in the next two sections.

- Scheduled approaches
  A slot-based approach divides the time into frames. Each frame has a defined amount of time and a common general structure. The common general structure defines which slots of the frame are signalling or data slots. All *Time Division Medium Access protocols (TDMA)* try to minimise packet collision by splitting up the frame into different slots. Each node gets then a slot assigned for communication. Obviously, this is a waste of maximum throughput, but most sensor network applications do not need to transmit a huge amount of data. Moreover, it is possible to assign multiple slots to a node to guarantee certain throughput requirements. A challenging task for TDMA protocols is the slot management. TDMA requires an assignment of slots to each node. Therefore, before the nodes can send data in their slot, they have to compete for such a slot, or a cluster head has to assign the slots to the nodes in its cluster. A cluster in a network is a group of nodes. In general, all nodes in such a group have a direct connection to the cluster head. The already mentioned clock drift has a higher impact on slot-based approaches than on contention-based protocols, because the size of a slot is much smaller that the common listen period in contention-based protocols. Examples for this type of protocols are: *TRAMA* [9] and *LMAC* [10].

The upper layer of the MAC layer is the routing layer. If the source of a message and its destination do not have a direct link, the routing layer provides the functionality to route messages over intermediate nodes. The routing layer also takes care that the MAC layer knows to which node it has to send a message.

2.1 S-MAC

2.1.1 Overview

S-MAC [7] splits an entire frame into two periods. One is the working slot (listen period) where the radio is on. In the other period the node goes to sleep. From time to time a node remains awake for an entire frame (see also Figure 2.1). This occurs periodically for synchronization reasons. Each node tries to broadcast a synchronisation (SYNC) packet at the beginning of its listen period. The *SYNC* packet contains a relative timestamp, indicating the senders next listen period. These SYNC messages are used to adjust the listen periods of the nodes.

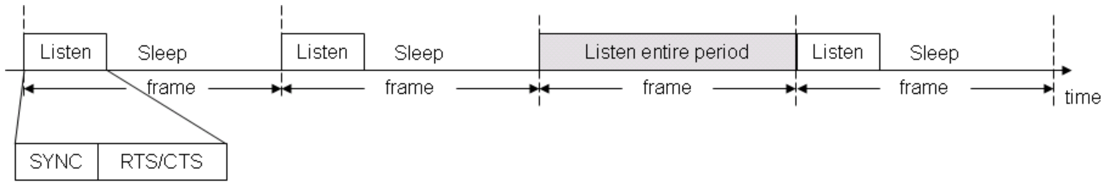


Figure 2.1: Sleep and listen periods of S-MAC: The node periodically (grey) listens the entire frame. A listen to sleep ratio of 0.1 is normal.

Some ideas of S-MAC have been taken from IEEE 802.11. Like IEEE 802.11, S-MAC is working with RTS/CTS (Request To Send/Confirm To Send) signalling packets, because IEEE 802.11 does a good job on collision avoidance with such RTS/CTS packets. Each packet in S-MAC has a duration field. So all overhearing nodes know how long they have to be silent until they can try to send their own packet. According to this rule, theoretically, a node can use the channel all the time. Therefore, the system is intrinsically unfair. In most cases, this is not an issue because all nodes in a wireless sensor network work for the same application and the application itself takes care that all nodes can send their data.

If the listen period is chosen longer, the power usage increases but the bandwidth becomes higher too. To increase throughput while keeping communication costs low, S-MAC introduces a new technique called adaptive listening (see Section 2.1.4).

2.1.2 Synchronization and Virtual Clustering

As already mentioned synchronisation is a big issue. S-MAC uses a simple, but well working algorithm. A new node listens to the channel if there is another node which has already a schedule. In other words, a new node is listening for a SYNC packet. These packets contain the ID of the sender and the remaining time until it starts sending its next SYNC packet. Before sending its packet each node introduces a random delay according to the current contention window to prevent collisions. If a SYNC packet is received within a given time period, the node adapts this schedule. Otherwise, it defines its own schedule and announces this schedule by broadcasting a SYNC packet.

All nodes sharing the same schedule build a virtual cluster. All nodes regularly stay awake for an entire frame. Thus they have the possibility to detect other virtual clusters. If there are two or more schedules from two or more nodes, the overhearing node adapts all schedules, but sends its SYNC packet only in one listen period. It still belongs to its original virtual cluster but it awakes also in the other listen periods. This means that the node has a significantly higher power usage. Otherwise it would not be possible to send data from one virtual cluster to an other.

The clock drift of each node does not affect S-MAC much, because the listen period is significantly longer than the clock drift rates. Moreover, each SYNC packet updates and adjusts the schedules of all receivers. Therefore, nodes overhear the SYNC packets from their neighbourhood nodes. No SYNC packet are received in case of packet collision, if background noises are too high, or if a node sends its SYNC packet outside the listen period due a high clock drift caused by a high packet loss rate.

2.1.3 Overhearing Avoidance

In general, overhearing is a waste of energy. A node does not need to listen to the communication between two nodes if it does not take part in the communication of them. Accordingly, at the moment a node is overhearing a RTS or CTS packet, which is not destined for the node, it goes to sleep. All packets have a *Network Allocation Vector (NAV)* field. This fields contains the duration of the entire transmission and hence all nodes know how long they can turn off their radio (see also Figure 2.2).

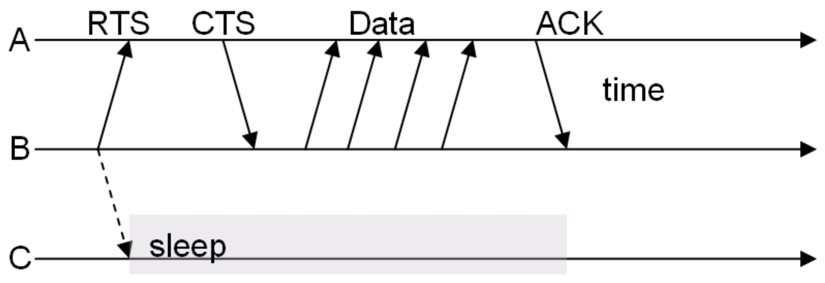


Figure 2.2: Overhearing Avoidance: Node B sends a packet to node A. Node C overhears the RTS packet and goes to sleep thereafter. Because of the NAV field, node C knows the duration of the transmission and therefore wakes up just at the end of the transmission.

2.1.4 Adaptive Listening

In a later publication [7] S-MAC was enhanced with adaptive listening and message passing. The basic idea of adaptive listening is to let any node x which overhears a transmission of a neighbour node y wake up for a short period of time at the end of the transmission. Thus, neighbour node y is able to immediately pass the data to node x. Otherwise, y would have to wait for the next scheduled listen time. If node x does not receive anything during adaptive listening, it goes back to sleep until its next scheduled listen time.

2.1.5 Message Passing

The transmission of a long message in a single packet needs less energy than to divide the message into several packets. The amount of payload remains the same but the number of control packets is reduced and there are fewer headers needed. However, the risk that the receiver does not receive a message correctly is increased with the length of the packet. The approach of S-MAC is to fragment long messages into multiple small fragments and transmit them in a burst (see Figure 2.3). The receiver confirms each data packet with an ACK packet. The RTS/CTS procedure is only required to setup the connection. For the subsequent exchange of fragments it is not needed.

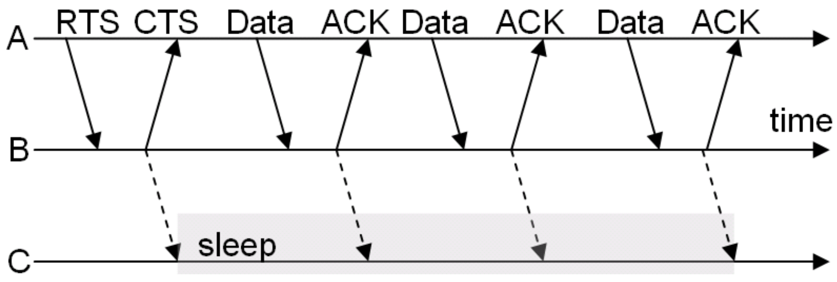


Figure 2.3: Message Passing: Node A sends 3 packets to node B. Node C overhears the RTS and goes to sleep according to the NAV value in the RTS message. Node A can send 3 packets after only one RTS/CTS. Message passing saves the transmission of 2 RTS and 2 CTS packet in this example.

Every packet (even the ACK packet) has a duration flag. If another node, which is not involved in the communication, wakes up during a transmission, it will immediately go to sleep again as soon as it overhears the NAV value in a packet. It does not matter whether this node overhears the sender or the receiver.

2.1.6 Discussion

The advantage of S-MAC compared to IEEE 802.11 is obvious. If there is not much traffic, the energy consumption is much lower than in IEEE 802.11. The delay can be improved by implementing adaptive listening. S-MAC is not designed for applications with high data rate requirements or very tight transmission delay requirements. A system with adaptive listening has a significantly higher data throughput than a system without this feature.

2.2 T-MAC

T-MAC (Timeout-MAC) [8] is an adaptive energy-efficient MAC protocol for wireless sensor networks. It reduces the idle listening. T-MAC is an improvement of S-MAC. Under homogeneous load both protocols achieve similar results. In a scenario with variable load, such as an event detection application, T-MAC outperforms S-MAC, according to the authors, by a factor of up to 5. T-MAC uses the same clustering and synchronization algorithm as S-MAC (virtual clustering) [7][8]. The main difference between T-MAC and S-MAC is the idea of a variable length of the listen period. If there is no data to be sent, the listen period can be shortened. Each node can switch to sleep mode at the moment it determines that there is currently no traffic for it. On the other hand, a node in T-MAC never goes to sleep as long as any communication is overheard. It always remains awake until the end of the transmission. There is consequently no overhearing avoidance like in S-MAC.

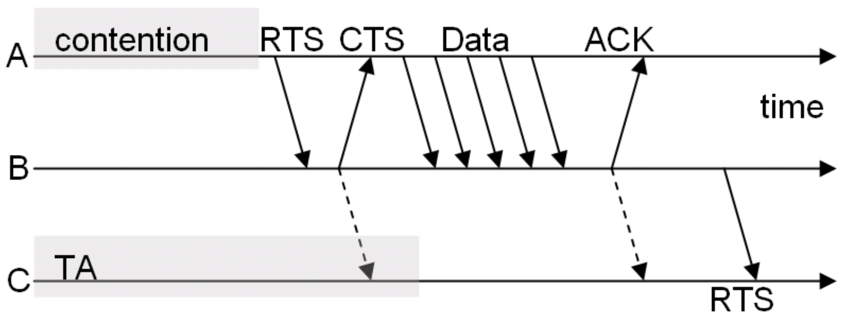


Figure 2.4: T-MAC with contention window.

In contrast to S-MAC, nodes in T-MAC turn off their radio after a short period if there is no communication of data pending. Before going to sleep, any T-MAC node has to make sure that no data is scheduled for it. This means it has to check that no communication is pending in its neighbourhood. Therefore, T-MAC introduces a new parameter called TA (Time Amount). TA defines the minimum awake time of a node after sending its SYNC packet. As indicated above, this timeout is needed to determine pending data transmissions in the neighbourhood. According to Figure 2.4 Node C has to listen to the radio for at least the time of the following actions: The maximum possible length of the contention window, the transmission of a RTS packet and a short amount of time so it could overhear the start of a CTS packet. Because Node C overhears the communication between A and B it remains awake in this example. After the complete data transmission between A and B, it is possible that B routes the data further to C. Therefore, Node C has to wait for the end of the entire transmission. Experiments in [8] have shown, multiplying the TA with a value of 1.5 yields better results while increasing the power usage only little.

TA determines the minimal amount of idle listening per frame. The buffer capacity determines an upper bound on the maximum frame time, because all messages between two listen periods must be buffered. In a real-world implementation the maximum frame time must be chosen even shorter because there is no guarantee (for example packet collisions) that all nodes can send all the packets in their buffers in the next listen period.

T-MAC follows the principle that it is better to maximise the throughput for a short amount of time than to save energy by avoiding overhearing. Avoiding overhearing has also a bad side effect. The collision overhead increases with the overhearing avoidance of S-MAC, because a node which is not involved in a communication could disturb some communication by sending a RTS packet after awaking. This cannot be prevented because of the communication model of S-MAC.

2.2.1 Early Sleeping Problem

A node can go to sleep at the moment there are no more packets to be transmitted to it. However, it is difficult for a node to determine whether there is a transmission waiting for it or not. A node does not always know if another node wants to send data to it and therefore it might turn off its radio too early. This early sleeping problem forces T-MAC to have a different data exchange mechanism than S-MAC. The problem is discussed in the following.

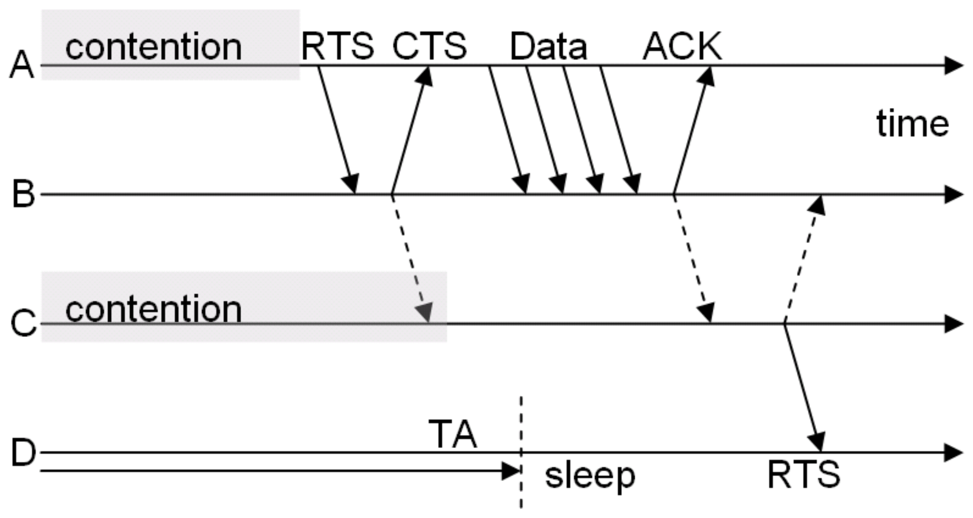


Figure 2.5: Early sleeping problem of T-MAC: A sends a packet to B; C would like to send a packet to D. However, D goes to sleep after the expiration of its TA.

The problem is shown in Figure 2.5. Node A wants to send a packet to node B. Node C has a message for D at the same time. A has the shorter delay and it starts to send a RTS to B. B receives this packet and it replies with a CTS afterwards. C has now to be quiet until the communication between A and B is finished. In the meantime D goes to sleep because it is outside of communication range and C cannot inform D that another communication is waiting.

T-MAC introduces a simple solution to fix this. It introduces a ”Future request-to-send” packet (FRTS) and Data Send packet (DS). Consider the situation in Figure 2.6. This time, node A sends a dummy data packet with the same size of a FRTS packet after receiving the CTS from B. This gives C the possibility to send its FRTS packet to D meanwhile. Thus node C is able to inform D about its pending data for D. Node D will receive a RTS packet after another running communication. C knows the duration of the transmission because it overhears the CTS packet with the NAV of Node B. FRTS increases the throughput while decreasing the latency. To prevent that a different node can take the channel while C is sending its FRTS, node A broadcasts

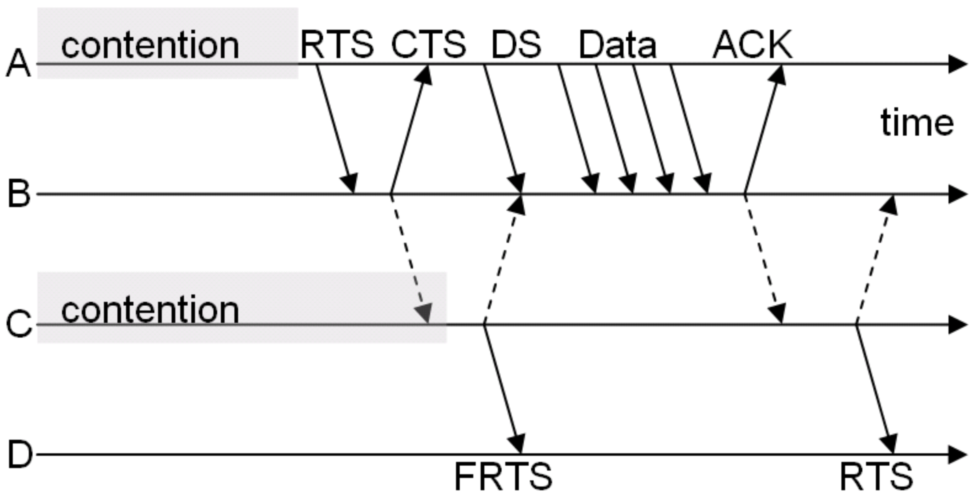


Figure 2.6: Solution to the early sleeping problem: Same Situation as in Figure 2.5. Node C can send a FRTS packet to node D before node A sends its data to B. Concurrently, node A sends a DS Packet to keep the channel occupied.

a DS packet at the same time. The content of this packet is regardless, though. Its only purpose is to keep the channel occupied.

2.2.2 Taking Priority on Full Buffers

A sensor network differs from an IEEE 802.11 network in some ways. The packet buffer at each sensor node is much smaller and allows the buffering of only few messages. Therefore, the risk of loosing packets by a buffer overflow is much higher. On the other hand, network fairness is not so important because, in general, all nodes work for the same application. Therefore, the application itself takes care of network fairness. These two facts make it possible that a node can refuse accepting any packets if its buffer is (almost) full. If another node wants to transfer data and sends a RTS, it will not reply and send its own RTS.

2.2.3 Discussion

The authors of [8] compared T-MAC with S-MAC and CSMA. They have shown that CSMA without sleeping periods is very inappropriate for a sensor network. S-MAC and T-MAC achieve energy consumption of up to 98% compared to CSMA without sleeping. There are big differences in the results depending on the communication pattern. S-MAC and T-MAC achieve almost the same results if the payload is homogeneous, which is typical for polling applications such as glacier monitoring. In a sample scenario with variable load, e.g., a tracking application, T-MAC outperforms S-MAC by a factor of 5 in terms of energy consumption. The designers of T-MAC took the ideas of S-MAC and improved them to gain better results for event-based applications in wireless sensor networks.

Chapter 3

Connected Dominating Sets (CDS)

3.1 Introduction

There are already some papers and works about establishing and running a Connected Dominating Set (CDS), but only few of them focus on wireless sensor networks [11]. Moreover, most of the published papers use the CDS for routing purpose [11],[12],[13]. Only few introduce the CDS to save energy. One could say that a CDS is a kind of a virtual backbone in a wireless network. A CDS is defined as follows (taken from [12]): ”In general, a dominating set (DS) of a graph G = (V,E) is a subset V’ $\subset$ V such that each node in V - V’ is adjacent to some node in V’, and a connected dominating set (CDS) is a dominating set which also induces a connected subgraph of G. A (connected) dominating set of a wireless ad hoc network is a (connected) dominating set of the corresponding unit-disk graph. To simplify the connectivity management, it is desirable to find a minimum connected dominating set (MCDS) of a given set of nodes.”

Finding a MCDS is NP-complete, though. Therefore, heuristics are used to approximate a MCDS. In this chapter three relevant approaches are introduced. Our own approach is then presented in Chapter 6.

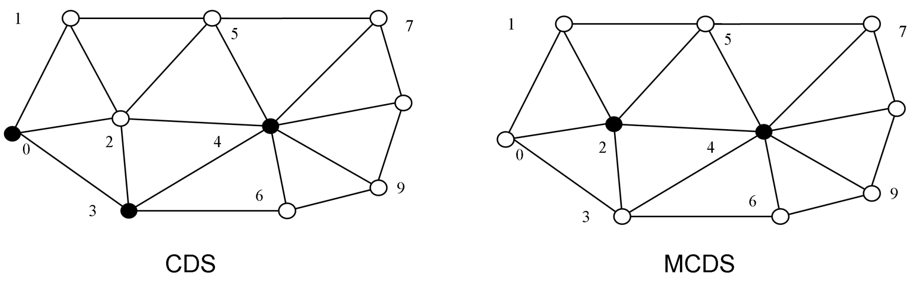


Figure 3.1: CDS and MCDS

Figure 3.1 shows examples of a CDS and a MCDS. Nodes in the respective dominating set are coloured black. Every white node has a direct link to a black node. In both examples the black nodes build a connected network. The minimum number of nodes to form a CDS in this example is two. Therefore, the CDS on the right is a MCDS.

3.2 CDS with Pruning Rules

One of the first approaches for building a CDS for mobile ad-hoc networks was published in [11]. In addition to the usual packets for data transmission such as RTS and CTS, the authors of [11] introduce an additional control message packet. Within this packet each node broadcasts its 1-hop neighbourhood information. This allows all receivers to know their 2-hop neighbourhood. If two neighbours of a node do not have a direct link between each other, they are ”unconnected neighbours”. Based on its 2-hop neighbourhood knowledge each node can determine whether it has unconnected 1-hop neighbours or not. Each node marks itself if it has unconnected neighbours in its 1-hop neighbourhood. These nodes form the basic CDS. According to this rule, some nodes are marked in Figure 3.2. Node 3 has for example nodes 4 and 1 as unconnected neighbours and therefore joins the backbone. On the other hand, having received the control messages from nodes 2 and 3, Node 1 ”knows” that there is a link between 2 and 3. Hence, Node 1 does not have any two unconnected neighbours, which means that Node 1 does not have to join the backbone.

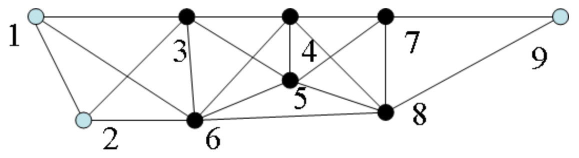


Figure 3.2: Nodes 1, 2 and 9 are border nodes. For example Node 3 is a marked node because neighbouring nodes 4 and 1 are not connected.

On the initial CDS two pruning rules are applied to reduce the size of the CDS by minimizing the number of marked nodes. The first rule means: If the 1-hop neighbourhood of a marked node (v) is also covered by another marked node (u) of the 1-hop neighbourhood of node v, and if the ID of v is smaller than the one of u, then v can leave the CDS.

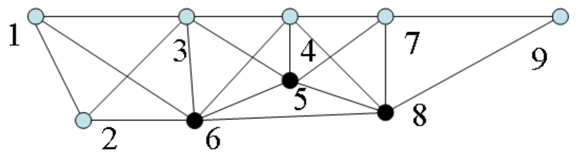


Figure 3.3: Result after the first pruning rule. Nodes 3, 4 and 7 are not marked anymore. The neighbours of Node 3 are for example covered by Node 6. The same applies to nodes 4 and 7.

An example is depicted in Figure 3.3. Node 3 leaves the CDS because Node 6 covers all neighbours of Node 3 and its ID is smaller. Therefore, Node 6 is able to handle all routing tasks of Node 3. The second pruning rule is similar: If all nodes in the 1-hop neighbourhood of a node v are covered by two or more marked and connected nodes of the 1-hop neighbourhood of node v and v has the smallest ID, then v can leave the CDS.

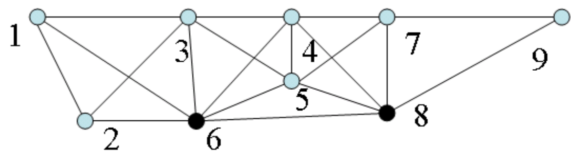


Figure 3.4: Result after the second pruning rule. Node 5 is not marked anymore. Nodes 6 and 8 cover the neighbours of Node 5.

In Figure 3.4 Node 5 leaves the CDS. Nodes 6 and 8 cover together all neighbours of Node 5. Additionally, Node 5 has a lower ID than nodes 6 or 8. Therefore, nodes 6 and 8 are able to handle all routing tasks of Node 5.

Discussion of CDS with Pruning Rules

This CDS is primarily developed for routing purpose. The design goal is to minimize the number of nodes in the backbone, which means to search the minimum number of nodes which is needed to guarantee network connectivity. Most nodes in the centre of the network are a part of the CDS because they have at least two unconnected nodes and many of them will also endure the subsequent pruning. A backbone makes routing simpler, but not all nodes in the centre of the network have additional benefits such as saving energy or increasing maximum throughput. An unmodified implementation of such a protocol would not make much sense in a wireless sensor network, because many CDS nodes (mainly in the centre of the network) would run out of energy soon. Moreover, the complete 2-hop neighbourhood knowledge is required. The collection of this two-hop neighbourhood information might be too expensive in terms of communication and storage for wireless sensor networks, though.

3.3 Maximal Independent Set (MIS) CDS

[12] introduced an approach in which the algorithm needs first a leader election algorithm and afterwards a level calculation phase. [12] did not invent a new algorithm for the leader election and the level calculation, instead they overtook ideas from [14]. In the level calculation phase each node ”learns” along with its own rank also the rank of all 1-hop neighbours. The rank is a combination of the number of hops to the leader and its ID. This pre-work is required to build up a maximal independent set (MIS). This is a dominating set in which any pair of nodes are non-adjacent. That means that there are no nodes in the dominating set which have a direct link to each other. The two black nodes in Figure 3.5 build a MIS. Except for the leader of the network all nodes are coloured white in the beginning. The leader is marked black. It broadcasts a BLACK message afterwards (see also Figure 3.5). Like the later used GREY message this is a simple packet with a packet type flag and the ID of the sender. All white receivers of a BLACK packet mark themselves grey. All these grey nodes broadcast a GREY message afterwards. All receivers of the GREY messages check whether there is no other white node with a lower rank in their 1-hop neighbourhood. If this is the case, this specific node becomes a BLACK node as

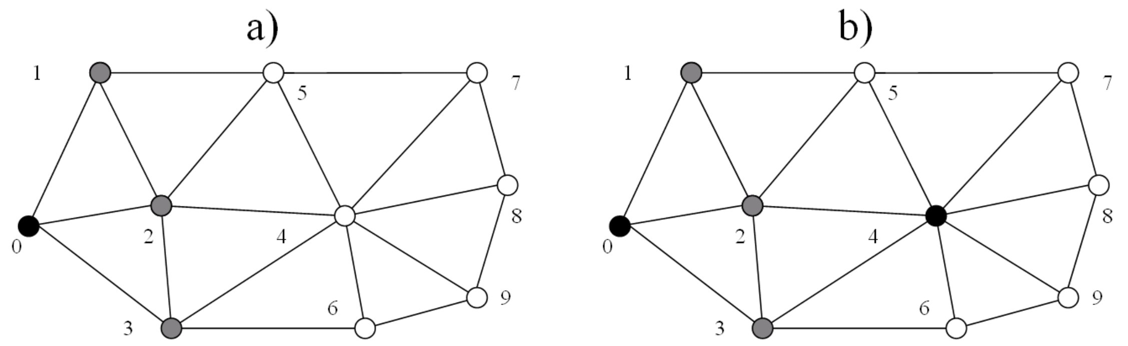


Figure 3.5: a) Node 0 broadcasts a BLACK message and all receivers mark themselves grey. b) Node 1, 2 and 3 broadcast a GREY message. Node 4 marks itself black because it has the lowest rank.

well, and broadcasts a BLACK message. The others remains coloured grey. Finally, all black nodes form the MIS.

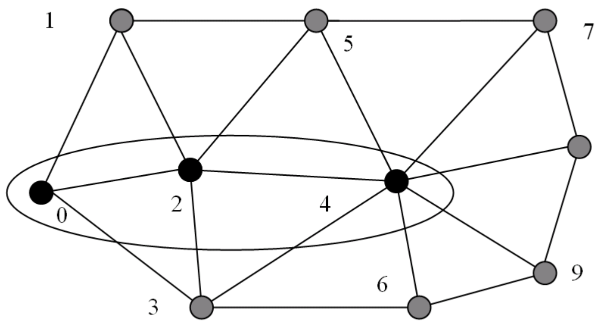


Figure 3.6: In this example Node 2 is elected to connect the MIS established in the previous step.

In order to generate a CDS from the MIS, the nodes in the MIS need to be connected. Therefore, in a second step a smart algorithm selecting promising connecting nodes has been proposed (see Figure 3.6).

Discussion of MIS-based CDS

The entire development process takes some time. Besides, it produces a lot of traffic. Before the MIS selection algorithm is able to be started, another algorithm has to define the leader of the network if there is no base station. After that, each node has to know its ranking position based on the number of hops to the leader in a virtual tree and its ID. Each node must only store the information of its 1-hop neighbourhood. In comparison to the CDS with pruning rules [11], the MIS-based CDS achieves a better CDS and it builds up the network faster. There are some drawbacks of using MIS-based CDS in a wireless sensor network. The MIS-based CDS is not flexible in terms of local updates. If a node has a malfunction, it is possible that the entire process needs to be restarted. Last but not least, the algorithm with its different algorithmic steps might be too time consuming to be implemented on a wireless sensor node.

3.4 Timer-based CDS

The MAC-Layer Timer-based Connected Dominating Set Construction Protocol (MTCDS) has been introduced in [15]. This protocol aims at saving energy and enabling routing in general. The protocol is divided into two parts. The first one is called the initiator election. During this phase, it defines the initiating node or initiator of the second phase. If a base station is the initiator, this phase can be skipped. MTCDS uses the node ID and determines the node with the smallest ID as initiator. In the second phase, the initiator starts to construct the CDS. For this, the initiator becomes a member of the CDS (inDS) and announces this in the beacon frame extension of a normal IEEE 802.11 beacon message. A node periodically broadcasts these beacons, which are similar to the SYNC packets in S-MAC or T-MAC. The beacon contains the ID of the sender. From beacon messages, each node is able to learn its 1-hop neighbourhood. Its extension has a size of 58 bits. This is only about 10% of the default beacon packet size of IEEE 802.11, which is about 550 bits. The beacon extension contains the status of the node, which is either uncovered, covered or in DS. The initial state is uncovered. All receivers of the beacon sent by the initiator become covered nodes, which means that they have a 1-hop connection to the CDS. All receivers start a timer immediately. The value of this timer is defined for all nodes with one or more uncovered neighbours by the following function:

$$\Delta T = T_{max} \cdot \frac{1}{(number\ of\ uncovered\ neighbours)^{\alpha}} \tag{3.1}$$

When the timer expires, the node becomes a member of the CDS and announces this in the next beacon frame extension. If the number of uncovered neighbours is 0, the specific node is not part of the CDS.

The process is also illustrated in Figure 3.7. Node 0 is the initiator and becomes a member of the CDS. It announces its status by its next beacon message. Nodes 1, 2 and 3 receive this message and start a timer (Figure 3.7 (a)). The timers of Node 2 and 3 expire first because Node 1 has only one uncovered neighbour. Therefore, nodes 2 and 3 join the CDS (Figure 3.7 (b)). Node 2 and 3 announce their status in their next beacon messages and nodes 4, 5 and 6 start their timer at the moment they receive the messages (Figure 3.7 (c)). Node 4 has the highest number of uncovered neighbours and therefore it joins the CDS (Figure 3.7 (d)). The process goes on until all nodes are either in the CDS or they are completely covered and become dominated.

Discussion of Timer-based CDS

[15] introduced a solution for building up a CDS without any additional packets. The additional traffic is low as well. This approach could be useful for a wireless sensor network if Equation 3.1 was extended with the battery level. If not, a well placed node always has to enter the CDS. For a wireless sensor network with battery operated nodes, this is not acceptable. Even if the battery level would be low, the specific node would still be in the CDS and still use more energy

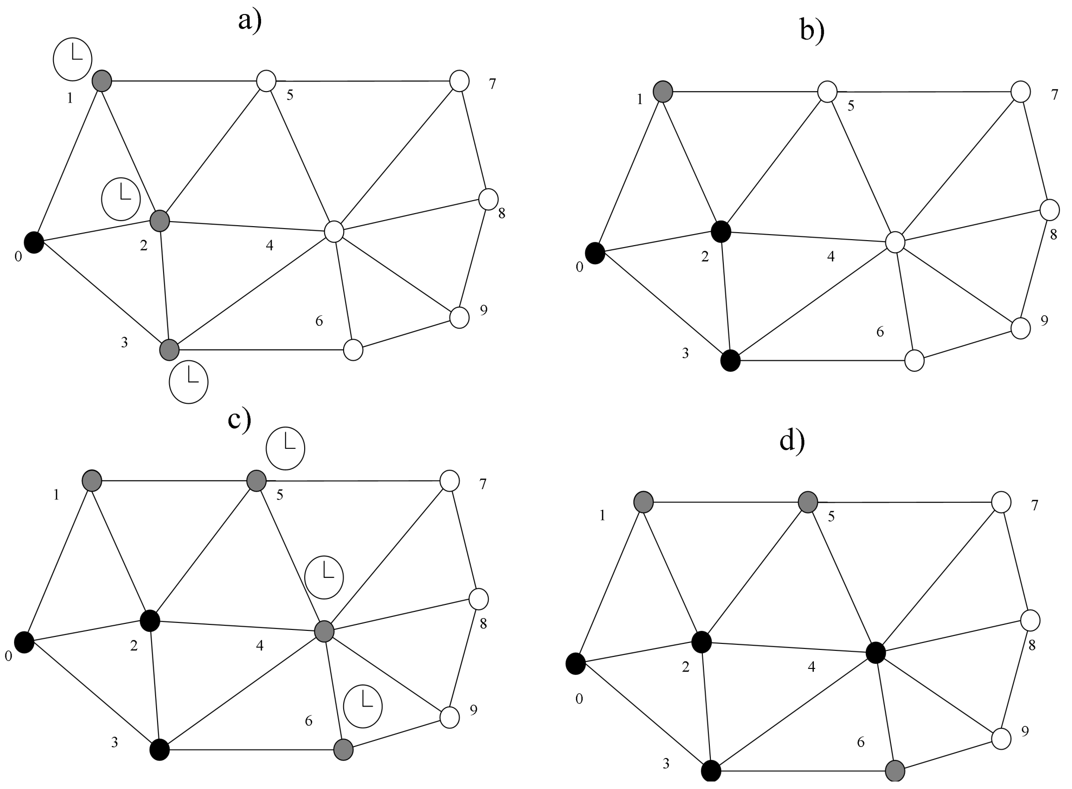


Figure 3.7: CDS building process of MTCDS.

than others. A wireless sensor network should to distribute the energy consumption over all nodes. Finally, the approximation factor of the MCDS by MTCDS could be improved. In our own approach we incoporate a timer-baded approach such as MTCDS with negotion and 2-hop meighbourhood information.

Chapter 4

Clock Synchronization

4.1 Introduction

Each node has an internal clock. The nodes use this clock mainly for internal alerts but the application itself could need the clock as well. For example, an application that detects sensor events and sends them to the base station needs a clock. Otherwise, the base station will not know how old this information is. On the other hand, nodes for wireless sensor networks have to be cheap. Cheap devices are normally not as exact as expensive ones. Consequently, due to fabrication reasons each node has an individual clock drift which is not negligible. For many MAC protocols there is the requirement that nodes are synchronized, at least with the nodes in their neighbourhood. In other words, many MAC protocols need a solution to synchronize the nodes. In this chapter an overview over relevant related work is given. Our own solution is then proposed in Chapter 7.

4.2 Network Time Protocol NTP

NTP [16] is a well-known protocol used mainly in the Internet. The first definition was made in 1992. NTP works with reference clocks. These are atomic (caesium, rubidium) clocks, GPS clocks or other radio clocks. These reference clocks are the roots of a hierarchical system of ”clock strata”. All reference clocks have, therefore, the stratum level 0. All hosts which are able to get the time from a stratum n device will become stratum n + 1. Accordingly, a client of a NTP server acts as a server to down-link devices.

If a NTP client wants to update its clock, it sends a request packet to a NTP server. This packet contains the timestamp of the client. The server adds its own timestamp and a transmit timestamp to this packet before it sends the packet back. The second timestamp helps the client to determine the travelling time of the packet. Thus, the client can estimate the actual time of the server. It is only an approximation because there are several delays. There are some variable processing times at the client and at the server and there is a variable transmission delay over the network. The shorter and more symmetric the round-trip time is, the more accurate the estimate of the server time will be. This procedure is performed several times before it passes some sanity checks.

Discussion of NTP and Wireless Sensor Networks

NTP works well but it is not optimised for energy and does not exploit a broadcast medium. Additionally, there is a lot of preconfiguration required until it works in a new, independent network.

4.3 Time Synchronisation with GPS

GPS (Global Positioning System)[17] computes the position of a GPS device all over the world. Exact clock synchronization with the GPS satellites is required to achieve an accurate position estimate. More than 24 satellites are currently placed in the orbit. Each of them knows its exact position and the current time. Each satellite has more than one atomic clock on board.

The basic principle of GPS is simple. All satellites periodically broadcast their position and the time of sending this information. With the time difference between sending the data and receiving it at a receiver it is possible to estimate the distance between sender and receiver. Moreover, because the propagation speed of the signal approximates light speed, the clocks at the receiver are adjusted very exact. In general there are 3 unknown variables. This are x and y for the position and t for time. At the moment there are at least four satellites visible, it is possible to solve the according system of equations. With five visible satellites it is possible to determine also the altitude.

Discussion of Time Synchronisation with GPS

GPS is an exact approach to compute the current time. There are several disadvantages for using it in a wireless sensor network, though. First, the GPS module takes a lot of energy. Modern GPS receiver chips like the SIRF Star III [18] still need about 50 mW. GPS device needs line of sight to contact the satellites. Without a relay GPS is not available indoors. And last but not least, a GPS chip still costs several dollars.

4.4 IEEE 802.11 Synchronization

IEEE 802.11 defines a Timing Synchronization Function (TSF) for the ad-hoc mode in the MAC layer of a wireless network. Basically it uses a similar concept as S-MAC and T-MAC, i.e., a synchronization packet is periodically broadcast by each node. IEEE 802.11 defines a frame-length. The length of this frame depends on the various IEEE 802.11 standards and bit-rates. At the beginning of each frame there is a beacon generation window. Each node defines at the beginning of the first window a random number which is uniformly distributed between 0 and w where w is 15 or 31. This value represents the delay until it tries to send its beacon. Each beacon contains a timestamp among other parameters. At the moment a node overhears a beacon from another node it will cancel its own pending beacon transmission and adjust its own clock to the timestamp of the received beacon. TSF will never adjust a clock backwards. If the received timestamp is older than the current of the receiving node, it is discarded. Thus, TSF adjusts the clocks to the fastest one in the entire network.

Discussion of IEEE 802.11 Synchronization

TSF is easy to implement. It never achieves such good accuracy as NTP or GPS. On the other hand, there is no additional hardware required and the additional traffic is low. If the nodes are listening to the radio only during the beacon generation window, it is possible that a node gets disconnected from the network. Huang and Lai [19] show that this can happen if the number of nodes is large.

4.5 Reference Broadcast Synchronization (RBS)

Reference Broadcast Synchronization (RBS) [20] is a simple but powerful solution for wireless networks. Similar to T-MAC and S-MAC each node sends periodically a synchronization packet. RBS calls this packet *reference packet*. In opposite to T-MAC and S-MAC, nodes running with RBS will never adjust their clocks. Every node only adjusts the clock skew. RBS tries to eliminate the fix drift between two or more nodes.

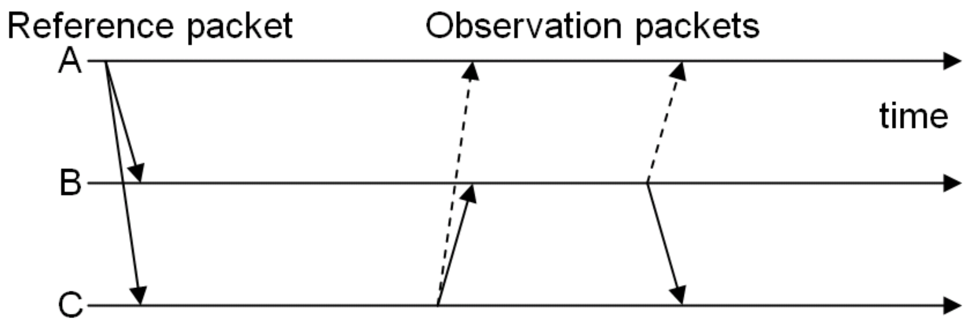


Figure 4.1: Reference Broadcast Synchronization.

At the moment two receivers (B and C) get the reference packet (from A) they will broadcast their observations. This is illustrated in Figure 4.1. Both receivers broadcast in the observation packets the local time of the moment they have received the reference packet. At the moment a receiver (B or C) has received both the reference packet and the observation packet, it can determine the time difference between the clock from the sender of the observation packet and its own clock.

Discussion of RBS

RBS eliminates several sources of errors. In particular, the impact of the processing time at the sender of the reference packet and the delay in the network interface card are avoided. The reasons for synchronization inaccuracy are the different propagation speeds and the fluctuations in packet processing time at the receiver. RBS has been implemented in a wireless sensor network. The reference packet could be a SYNC packet with a special flag. The observation packet would be an additional packet.

4.6 Timing-sync Protocol for Sensor Networks (TPSN)

The Timing-sync Protocol for Sensor Networks (TPSN) [21] was designed for wireless sensor networks. TPSN has two phases: The first is the level discovery phase and the second is the synchronization phase. The algorithm defines a root node before the first phase starts. A possible election algorithm is [22]. This root node is the top of a hierarchical structure and therefore gets level 0 assigned. It broadcasts a level_discovery packet. The level_discovery packet contains the identity and the level of the sender, i.e., 0 if the sender is the root node. All receivers of these packets become level i+1 and rebroadcast their level_discovery packet. If there are no packets lost due to collision, all nodes get a level assigned. The second phase is then quite similar to NTP. The root node is stratum 0 and all nodes with level 1 synchronize their clock with the root node. In general all nodes from level i synchronize their clock with a node from level i-1.

In a wireless sensor network there might be many nodes in the 1-hop neighbourhood. Thus collisions might occur quite often. It is possible that a node will never receive a level_discovery packet. Therefore, TPSN introduces a local level discovery phase with an additional level_request packet. All receivers have to answer this request by rebroadcasting their level_discovery packet. The requesting node collects all level information from the level_discovery packets and chooses the smallest level. Finally, it assigns itself a hierarchical level of this value plus one.

The lifetime of a sensor node is limited. Hence, a node will stop working at the moment it runs out of energy. TPSN does not need to care about that if this happens during the level discovery phase, and before the node got a level assigned. However, the virtual tree could be broken if this happens after that time. The disconnected nodes in the 1-hop neighbourhood still try to get synchronized, but their requests will never be answered if there is no other node in the vicinity with the same level as the dead one. In this case all disconnected nodes send again a level_request packet following the procedure of the local level discovery phase. After some time the virtual tree will be repaired.

Discussion of TPSN

Simulation results show that TPSN achieves two times better results than RBS. Even over a 5-hop distance the average error is only about 22.66 microseconds. [21] implements the RBS protocol and TPSN in an IEEE 802.11 network. The hierarchical structure of TPSN can also be used for routing purpose if there is only node-to-sink communication required and the sink is the top of the tree. A drawback of TPSN is that it requires a lot of control packets and that it takes a rather long time until the nodes are synchronized.

Chapter 5

Simulation Environment

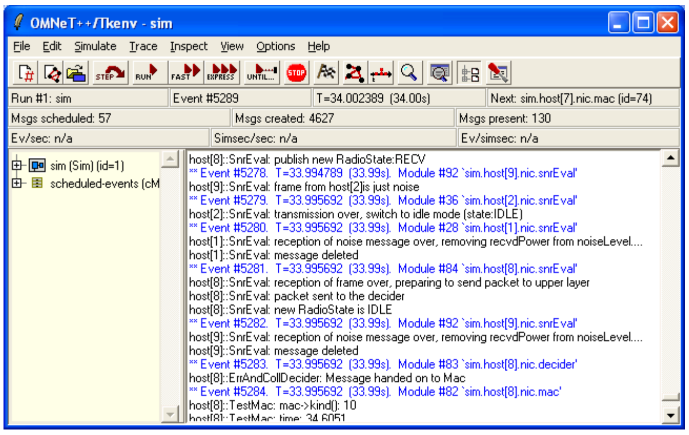
All protocols have been implemented in Omnet++ 3.1 [3]. The simulation model called mobility framework version 1.0a5 has been used. All tests were executed on various PCs with Windows XP or Windows 2003.

5.1 OMNeT++ 3.1 Overview

OMNeT++ is an open-source, component-based, modular and open-architecture simulation environment. The short description of OMNeT++, according to the developer, is [3]: ”Its primary application area is the simulation of communication networks and because of its generic and flexible architecture, it has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business processes as well. OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. Several open source simulation models have been published, in the field of internet simulations (IP, IPv6, MPLS, etc), mobility and ad-hoc simulations and other areas.” There is also a commercial version of OMNeT++. It is called Omnest [23].

OMNeT++ is a discrete event simulator written in C++. An event can be everything. It can be the start of a packet transmission, the detection of a movement of a sensor, a timeout and so on. All events are stored in a global event scheduler in strict time order. The simulation terminates as soon as there are no more events scheduled or the simulation reaches the user defined time limit. The component-based character of OMNeT++ provides the user with various features. Each layer of the network stack is a component. Therefore, it is simple to exchange individual layers to alter scenarios. In this work only the MAC layer was altered in the different experiments. Because it is an open-source application, it is possible to change the core of the application, too. By changing a single value in the makefile, OMNeT++ creates, instead of an application with a graphical user interface (GUI), a simple program without any GUI.

Figure 5.1 shows the main GUI of OMNeT++. The main space in the window shows the log of the executed events. Additionally, the user can view the current state of each module and the scheduled events on the left side of the GUI. The top of the GUI provides various functions to interact with the simulation such as to define the speed of the simulation. Also, there is an item in the menu to open a new window to view the network topology (Figure 5.2). This window is,


OMNeT++/Tkenv - sim
File Edit Simulate Trace Inspect View Options Help
STEP RUN FAST EXPRESS UNTIL... STOP

```
Run #1: sim    Event #5289    T=34.002389  (34.00s)    Next: sim.host[7].nic.mac (id=74)
Msgs scheduled: 57    Msgs created: 4627    Msgs present: 130
Ev/sec: n/a    Simsec/sec: n/a    Ev/simsec: n/a

sim (Sim) (id=1)
scheduled-events (cM

host[8]::SnrEval: publish new RadioState:RECV
** Event #5278.  T=33.994789  (33.99s).  Module #92 `sim.host[9].nic.snrEval'
host[9]::SnrEval: frame from host[2]is just noise
** Event #5279.  T=33.995692  (33.99s).  Module #36 `sim.host[2].nic.snrEval'
host[2]::SnrEval: transmission over, switch to idle mode (state:IDLE)
** Event #5280.  T=33.995692  (33.99s).  Module #28 `sim.host[1].nic.snrEval'
host[1]::SnrEval: reception of noise message over, removing recvdPower from noiseLevel....
host[1]::SnrEval: message deleted
** Event #5281.  T=33.995692  (33.99s).  Module #84 `sim.host[8].nic.snrEval'
host[8]::SnrEval: reception of frame over, preparing to send packet to upper layer
host[8]::SnrEval: packet sent to the decider
host[8]::SnrEval: new RadioState is IDLE
** Event #5282.  T=33.995692  (33.99s).  Module #92 `sim.host[9].nic.snrEval'
host[9]::SnrEval: reception of noise message over, removing recvdPower from noiseLevel....
host[9]::SnrEval: message deleted
** Event #5283.  T=33.995692  (33.99s).  Module #83 `sim.host[8].nic.decider'
host[8]::ErrAndCollDecider: Message handed on to Mac
** Event #5284.  T=33.995692  (33.99s).  Module #82 `sim.host[8].nic.mac'
host[8]::TestMac: mac->kind(): 10
host[8]::TestMac: time: 34.6051
```

Figure 5.1: OMNeT++ main window.

together with the event log of the main window, very helpful to find bugs. The user has also the possibility to check the state of each module and its submodules. This allows the user to verify if the network setup was properly configured.

5.2 Configuration Files

In this section the configuration files of the OMNeT++ simulator are introduced. The individual components of the OMNeT++ simulator are configured in specific ned-files. Global parameters are set in an omnet.ini file.

5.2.1 Ned-files

OMNeT++ is a component-based simulation environment. Each component provides interfaces to connect other components. These are called gates. Each component can have some parameters such as the number of nodes of the network, battery size and many more. OMNeT++ introduces a configuration file type called ned-files and a simple language to define the parameters and gates. Each module has its own file. In the following an example of such a ned-file is given:

```
simple TestMacLayer
    parameters:
        debug: bool, // debug switch
        headerLength: numeric const, // length of the MAC packet header (in bits)
        queueLength: numeric const,
        seed: numeric const,
        bitrate: numeric const,
        maxVariableDrift: numeric const,
        maxFixDrift: numeric const,
        listenEntireFramelengthPeriod: numeric const,
```

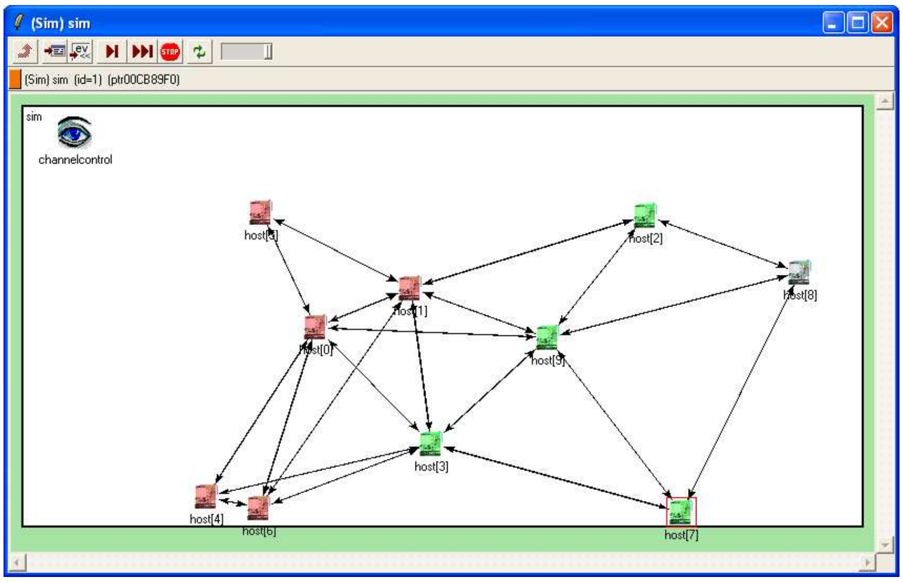


Figure 5.2: OMNeT++ network topology view.

```
        numHosts: numeric const,
        batterySize: numeric const;
    gates:
        in: uppergateIn;
        out: uppergateOut;
        in: lowergateIn;
        out: lowergateOut;
endsimple
```

The ned-file contains several sections. The first section describes the parameters of the module. In the example the simple module called ”TestMacLayer” has several parameters such as debug, headerLength, queueLength. Each parameter needs a type declaration such as bool or numeric const. The values for these parameters are defined in the omnetpp.ini file which is presented in the next subsection. The second section contains the gates. Here, the possible connections to others modules are listed. The example provides four gates. A module can also have some submodules. This helps to split up the code into multiple files. OMNeT++ furthermore supports the exchange of submodules to achieve different scenarios. The submodules would also be defined in the ned-file. OMNeT++ translates the ned-files into C++ code before it starts compiling the entire project. The syntax of the ned-files is much simpler than the syntax of the translated C++ code.

5.2.2 Omnetpp.ini

Instead of setting the parameters directly in the source code, OMNeT++ stores the parameters to configure the simulation in a file called omnetpp.ini. This file stores all parameters which are valid for all modules. In the following a part of an example of such a file is shown:

```
(...)
sim.host[*].nic.mac.debug = 1

sim.host[*].nic.mac.queueLength=25
sim.host[*].nic.mac.headerLength=16

sim.host[*].nic.mac.seed=-1

sim.host[*].nic.mac.batterySize = 40

sim.host[*].nic.mac.maxFixDrift= 0.001
sim.host[*].nic.mac.maxVariableDrift= 0.001

sim.host[*].nic.mac.listenEntireFramelengthPeriod= 35
sim.host[*].nic.mac.bitrate=115.2E+3; in bits/second

sim.host[9].mobility.x = 100
sim.host[9].mobility.y = 100
sim.host[*].mobility.x=-1
sim.host[*].mobility.y=-1
sim-time-limit = 100h
**.numHosts = 50
sim.playgroundSizeX = 1000
sim.playgroundSizeY = 700
(...)
```

The first parameter called ”debug” indicates whether debug statements are executed in the MAC layer or not. Debug information is provided if the value is 1. The debug parameter is defined in the according ned-file (see Section 5.2.1). Another interesting value is ‘sim-time-limit‘. It defines the maximum duration of the simulation. The last three lines define the total number of nodes and the dimension of the network. As there can be more than one host it is possible to define different parameters for each host. In this example sim.host[9].mobility.x and sim.host[9].mobility.y have different values than for all other hosts. This means that host 9 is set on position (100/100) in the simulation area, but all other nodes will be set randomly (indicated by -1). All parameters defined in the ned-file must be set in omnet.ini. Otherwise the simulator asks the user to provide these values during simulations start via a GUI.

5.3 Mobility Framework Overview

A cable connection between two devices is a one-to-one connection. There is always exactly one sender and one receiver of messages. OMNeT++ provides only such one-to-one interfaces. A gate of a module can connect to only one gate of another module. There is no wireless network connection type with one sender and many receivers. This is however required in a wireless medium. Accordingly, such broadcast communications need to be developed. With the mobility framework each host establishes connections to every other node inside its radio coverage. Moreover, the mobility framework provides functionality to model mutual interferences. This is required as in a shared wireless medium transmissions can disturb each other. Finally, the connections are not fix. At the moment the nodes move around connections change accordingly. This is again handled by the mobility framework. The radio coverage of the nodes is modelled by two path-loss models called Free Space [24] and Two Ray [25]. In our simulations we use the Two Ray model. The calculation of the interference distance D_i and the maximum receiving distance D_R of the Two Ray model is based on the transmission power P_T, the signal attenuation threshold α, the minimal receiving power P_R to decode data and the antenna height h_t. The formulas for the interference and the maximum receiving distance look as follows:

$$D_i = \left(\frac{P_T}{10^{\frac{\alpha}{10}}} \cdot h_t^4\right)^{\frac{1}{4}} \tag{5.1}$$

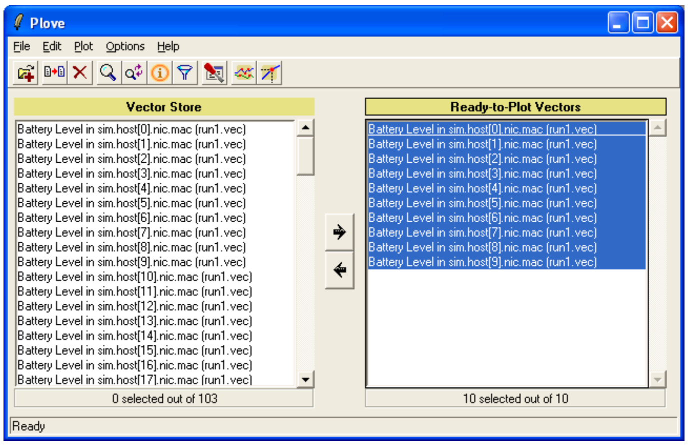
$$D_R = \left(\frac{P_T}{10^{\frac{P_R}{10}}} \cdot h_t^4\right)^{\frac{1}{4}} \tag{5.2}$$

An important parameter in both equitations is the antenna height. A long antenna increases the interference distance and the receiving distance because of the power of 4.

5.4 Collecting Simulation Results in OMNeT++

OMNeT++ collects two kinds of simulation results. These are the ‘vectors‘, which contain all values of a specific parameter measured over time, and the ‘scalar‘ values, which are single global values without a specific simulation time. A typical example of a ‘vector‘ is the transport delay of packets over time. An example of a ‘scalar‘ is the average transport delay of all data packets. Both results are stored in according files called omnet.sca and omnet.vec respectively. Both are simple text files and contain the value descriptions and the values. For each of these files there is an analysis tool provided by OMNeT++.

Figure 5.3 shows the GUI of the vector analysis tool. All stored vectors from the simulation are listed in the left side of the GUI. The user can select a subset of them and move them to the right side. By pushing the button called plove it plots graphs such as in Figure 5.4. OMNeT++ provides some setting options to configure these graphs and to save them finally as ps-files. The scalar analysis tool 5.5 is simpler than the vector analysis tool. It lists directly the entire collected scalar values from the file. Afterwards the user has the possibility to filter them and to plot them as a simple bar chart. Figure 5.6 shows, for example, the amount of energy used per node after 100 hours of a specific simulation. Finally, it is also possible to collect simulation



Vector Store

Battery Level in sim.host[0].nic.mac (run1.vec)
Battery Level in sim.host[1].nic.mac (run1.vec)
Battery Level in sim.host[2].nic.mac (run1.vec)
Battery Level in sim.host[3].nic.mac (run1.vec)
Battery Level in sim.host[4].nic.mac (run1.vec)
Battery Level in sim.host[5].nic.mac (run1.vec)
Battery Level in sim.host[6].nic.mac (run1.vec)
Battery Level in sim.host[7].nic.mac (run1.vec)
Battery Level in sim.host[8].nic.mac (run1.vec)
Battery Level in sim.host[9].nic.mac (run1.vec)
Battery Level in sim.host[10].nic.mac (run1.vec)
Battery Level in sim.host[11].nic.mac (run1.vec)
Battery Level in sim.host[12].nic.mac (run1.vec)
Battery Level in sim.host[13].nic.mac (run1.vec)
Battery Level in sim.host[14].nic.mac (run1.vec)
Battery Level in sim.host[15].nic.mac (run1.vec)
Battery Level in sim.host[16].nic.mac (run1.vec)
Battery Level in sim.host[17].nic.mac (run1.vec)

0 selected out of 103

Ready-to-Plot Vectors

Battery Level in sim.host[0].nic.mac (run1.vec)
Battery Level in sim.host[1].nic.mac (run1.vec)
Battery Level in sim.host[2].nic.mac (run1.vec)
Battery Level in sim.host[3].nic.mac (run1.vec)
Battery Level in sim.host[4].nic.mac (run1.vec)
Battery Level in sim.host[5].nic.mac (run1.vec)
Battery Level in sim.host[6].nic.mac (run1.vec)
Battery Level in sim.host[7].nic.mac (run1.vec)
Battery Level in sim.host[8].nic.mac (run1.vec)
Battery Level in sim.host[9].nic.mac (run1.vec)

10 selected out of 10

Figure 5.3: OMNeT++ analysis tools: The desired vectors are chose (in the right) and can then be plotted.

results without these functions because the entire application is written in C++ and, therefore, everybody can make their own log files.

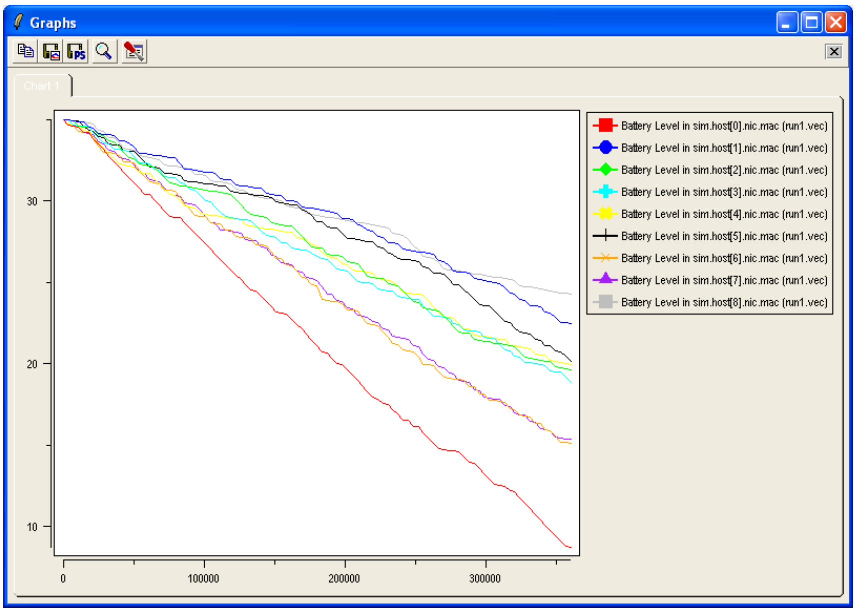
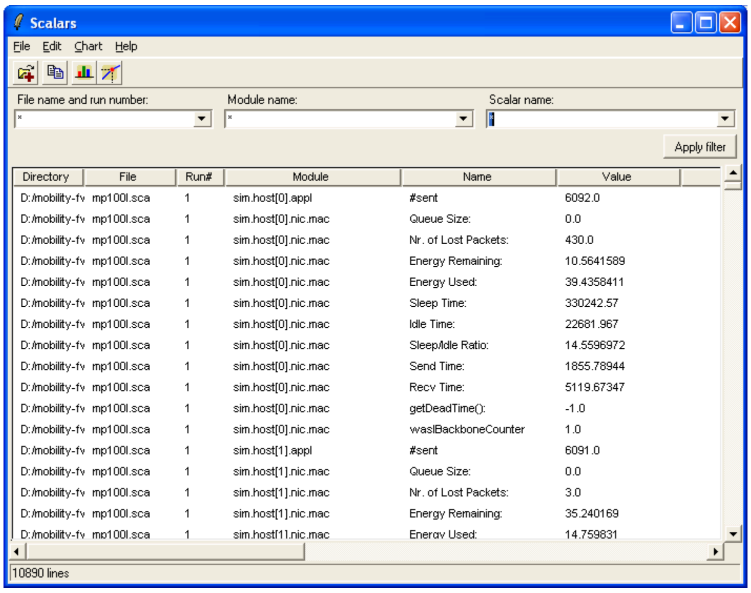


Figure 5.4: OMNeT++ analysis tools: Plot of a set of selected vector variables. The graph shows the decreasing amount of energy per node over time.



Directory	File	Run#	Module	Name	Value
D:/mobility-fv	mp100l.sca	1	sim.host[0].appl	#sent	6092.0
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Queue Size:	0.0
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Nr. of Lost Packets:	430.0
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Energy Remaining:	10.5641589
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Energy Used:	39.4358411
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Sleep Time:	330242.57
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Idle Time:	22681.967
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Sleep/Idle Ratio:	14.5596972
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Send Time:	1855.78944
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	Recv Time:	5119.67347
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	getDeadTime():	-1.0
D:/mobility-fv	mp100l.sca	1	sim.host[0].nic.mac	wasIBackboneCounter	1.0
D:/mobility-fv	mp100l.sca	1	sim.host[1].appl	#sent	6091.0
D:/mobility-fv	mp100l.sca	1	sim.host[1].nic.mac	Queue Size:	0.0
D:/mobility-fv	mp100l.sca	1	sim.host[1].nic.mac	Nr. of Lost Packets:	3.0
D:/mobility-fv	mp100l.sca	1	sim.host[1].nic.mac	Energy Remaining:	35.240169
D:/mobility-fv	mp100l.sca	1	sim.host[1].nic.mac	Energy Used:	14.759831

Figure 5.5: OMNeT++ analysis tools: List of scalar values.

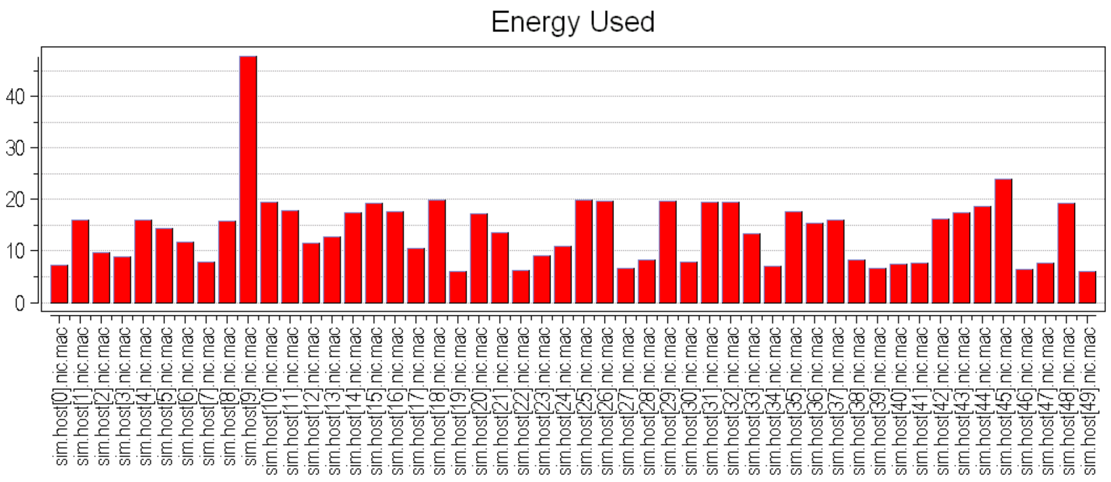


Figure 5.6: OMNeT++ analysis tools: Simple bar chart. The bars show the amount of energy used per node after 100 hours. Node 9 is the base station.

Chapter 6

Synchronization-based CDS for Wireless Sensor Networks

T-MAC and S-MAC allow communication between all nodes. Each node can create and send packets to all other nodes. Many possible applications for wireless sensor networks do not need to transmit data between sensor nodes, but rather have to report their data to a base station. In other words: Many applications need only node-to-sink communication, whereby the sink is the base station. In this work this property is used to extend the ideas of T-MAC and S-MAC to achieve better results for such kind of applications. In detail an algorithm that establishes a routing tree by using the SYNC messages of these contention-based protocols is provided.

6.1 Introduction

In T-MAC and S-MAC all nodes wake up periodically even if they have nothing to be sent. There are two reasons for that: First, they have to synchronise and second, they have to check whether there is another node waiting to send data to them. Under the assumption that there is mainly node-to-sink communication required, there is no need that all nodes wake up periodically as long as routing is guaranteed. In this work we guarantee routing by establishing a connected dominated set (CDS) which operates as backbone of the network. The entire traffic runs over this backbone. A node outside the backbone sends its data directly to a backbone node. For such non-backbone nodes there is no need to remain awake as long as there is no data from its sensors to be sent to the base station. Accordingly, these nodes can enter a long sleep period. Concerning the CDS setup, the following issues are considered:

- The base station is the sink of the CDS. The destination of all data is the base station.
- The number of nodes in the CDS should be as small as possible. Each node must have the possibility to periodically disconnect from the backbone, as long as the algorithm is able to maintain the connectivity of all nodes.
- Packets are routed over the backbone.
- The number of control traffic should be as small as possible.

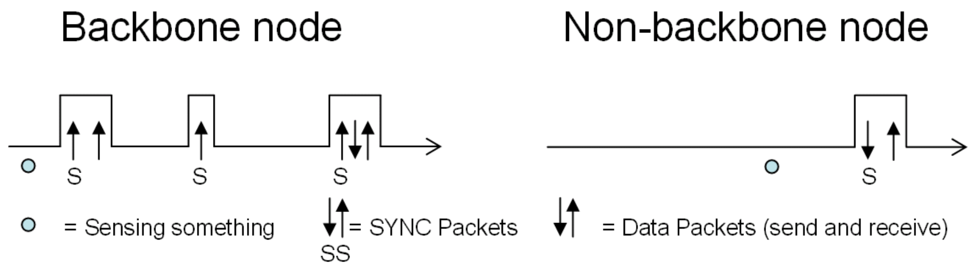


Figure 6.1: On the left: The backbone node sends data in the first frame. In the second there is no sensing at all. In the third frame it receives a data packet and sends it directly to a next hop node. On the right: Node is in sleep state. It wakes up at the next frame and after sensing something. After receiving a SYNC packet it sends its data.

All nodes in the backbone (= backbone nodes) are running the same way as in T-MAC or S-MAC. They wake up periodically and sometimes stay awake for an entire frame. These backbone nodes do not waste more energy than T-MAC or S-MAC nodes. They have little more traffic to handle, but they overhear fewer packets because fewer nodes have to send SYNC packets. The routing load of a backbone node is a little higher because it is less distributed than in T-MAC.

All non-backbone nodes save a lot of energy as long as there is no communication required due to events or sensor polling, because they remain disconnected and asleep for longer periods. This long sleep period has a duration of multiples of the frame length. If the sensors of a node detect something, the node has to transmit this information to the base station. The node wakes up at the next listen period and starts listening for a SYNC packet. This procedure is illustrated in Figure 6.1. The backbone node on the left side runs similar to T-MAC. It tries to send a SYNC packet at the beginning of each frame and sends and receives data if required. In the example above it sends its sensor readings in the first listen period. In the third listen period it routes a packet for another node. The non-backbone node (in the right) does not wake up at the beginning of each frame. After observing an event (activated sensors), it wakes up at the beginning of its next frame and starts to listen for a SYNC packet. At the moment, the node receives a SYNC packet, it sends its data after a short random delay. After sending its data, the node changes to sleep state again.

6.2 Synchronization of Non-backbone Nodes

Because of its clock drift a node can just estimate the next listen period. If for example the sensor on a node detects an event five seconds after the start of the last sleep period, it will predict the listen period quite exactly. The node can wake up close to the exact SYNC time. If the last sleep period was entered 30 minutes back, the clock drift might be too huge to predict the synchronisation period correctly. Therefore, the node wakes up immediately and starts listening for a SYNC packet. In other words: The longer the sleep period, the earlier (in relation to the next synchronisation time) it will wake up in order to listen for a SYNC packet.

If a non-backbone node, with some data pending for transmission, cannot send its data, because another node is sending its data, the node has to wait until this transmission is finished. Like in S-MAC and T-MAC the node tries to send its data with RTS/CTS after a random backoff. It is also possible that the first packet a node overhears is not a SYNC packet. In this case the node treats the overheard data message as a SYNC packet. The sender of this data packet might also be a non-backbone node. Therefore, it is possible that a non-backbone node accepts requests from another non-backbone node. This might occur if a non-backbone node looses connection to the backbone, e.g, because the according backbone node fails or its SYNC packet was disturbed by another packet. In this scenario the non-backbone node remains awake and scans for other nodes. If a non-backbone node appears, it connects to that node, forwards its data to it and goes to sleep again. The neighbourhood node must have a connection to the backbone, otherwise it would never send any packets. The benefit of this behaviour of the nodes in such a situation is obvious. Only one node needs to remain awake instead of both nodes and the risk of lost packets are minimised. This behaviour produces more traffic but the sender of the data can go to sleep earlier and therefore it will save energy regarding both nodes.

6.3 Periodic Backbone Reconstruction

It would be possible, in a static network, to maintain a CDS unaltered once it has been built. However, this would eliminate all the advantages over a system without CDS because a backbone node needs more energy than a non-backbone node. As soon as one of the backbone nodes would run out of energy, the entire branch of this network would be disconnected if there was no other backbone node which could overtake the work of the dead node. A part of the network would not be able to send their data to the base station anymore. Moreover, there would be no advantage of putting the non-backbone nodes into long sleep state if the CDS would never change, because this would not extend the lifetime of the entire network. Therefore, a solution that fuses the concepts of CDS and T-MAC for wireless sensor networks needs to rebuild the CDS from time to time. Furthermore, the solution should consider the current battery level of each node while establishing a new CDS to prevent that specific nodes are always selected into the backbone.

We propose two possible solutions to define a CDS in the next two sections. Both do not require additional packets. All data for building a CDS are attached to the standard SYNC packets. Both solutions focus on static networks. There are additional mechanisms required to handle mobility.

In contrast to pure T-MAC, it is important that all backbone nodes are able to send their SYNC packets periodically. Otherwise, if there was no other traffic overheard, a non-backbone node might have to wait too long until it detects a SYNC packet which enables it to connect to the backbone.

6.4 Negotiation-based CDS (N-CDS)

In pure T-MAC and also in pure S-MAC all nodes are able to learn their 1-hop neighbourhood from the SYNC packets. Every SYNC packet contains the ID of the sender and, therefore, each node learns the IDs of its neighbourhood over time. This collection of node IDs and the timestamp of the last receiving packet of specific node is the base information required by the CDS building process of N-CDS. Adding this timestamp has the additional benefit that a node is able to detect nodes which are unavailable for a longer period (for example by running out of energy). If a node does not receive a SYNC packet from another node for a certain amount of time, it assumes that the respective node is no longer available. A node has never a guarantee that another node is no longer available because the SYNC packets are broadcast packets and the transmission can always be disturbed, thus.

The base station is the root of a tree. The nodes of the tree are the backbone nodes. The base station initialises the building of the tree. The establishment of the backbone occurs ringlike away from the base station. Therefore, each node in the backbone knows its parent node and all packets for the base station are passed to these parent nodes.

6.4.1 CDS Building Process

In general the process consists of 4 steps. Each step is discussed in the following:

1. The CDS nodes broadcast a packet called CDSSYNC containing its neighourhood information.
2. Each receiver becomes dominated and calculates its priority.
3. The dominated nodes locally exchange their calculated priorities.
4. The node with the highest priority is elected into the backbone.

Step 1

The CDSSYNC packet is an extension of a normal SYNC packet. In addition to the normal parameters for synchronization, it also contains the ID of each non-backbone node in the 1-hop neighbourhood of the sender.

Step 2

All receivers are called dominated if their ID is listed in the CDSSYNC packet. A dominated node is a node that has a 1-hop connection to a backbone node and which has already received a CDSSYNC packet. The task of each dominated node is to figure out whether there is need to become a backbone node as well. There is only need if a node has not yet covered nodes in its neighbourhood. All receivers of a CDSSYNC packet will send all their subsequent data packets to the sender of the first CDSSYNC they have received in the current CDS building process, i.e., of the first dominator they have learned. Each receiver marks its neighbours that are also listed in the CDSSYNC packet as dominated. Each dominated node calculates a priority after the reception of a CDSSYNC. This priority is the product of its own battery level and the number of remaining nodes in its neighbourhood list that are not already backbone nodes or dominated.

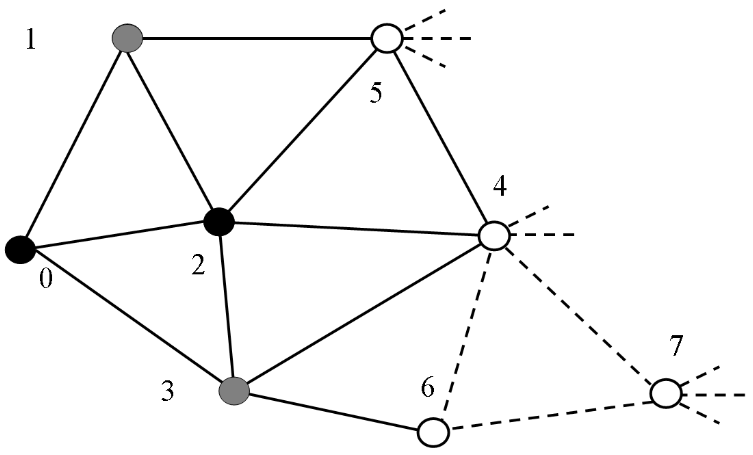


Figure 6.2: N-CDS: Alternative Path timer. If node 2 has the highest priority of the dominated nodes 1, 2 and 3, Node 3 has to start the alternative path timer to ensure that node 6 has a connection to the backbone.

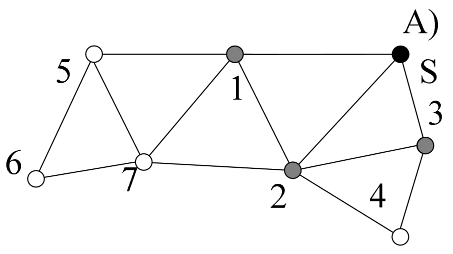
Step 3

The dominated nodes exchange their calculated priority with a CDSDOMINATEDSYNC packet. This packet contains the usual SYNC packet information and the priority of the node. The priorities are exchanged among neighbouring dominated nodes to determine the node with highest priority. All receivers of a CDSDOMINATEDSYNC packet update their neighbourhood list by marking the sender as dominated. If the calculated priority of a node is 0, the node knows that there is no need for routing traffic over it and it goes to sleep after having announced its state (like all other dominated nodes) with a CDSDOMINATEDSYNC packet. Nodes with a priority of 0 need to announce their state, otherwise their neighbourhood nodes would not know about their state and they would wait for the expiration of a timer (see next subsection).

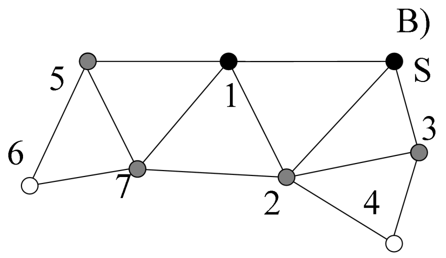
Step 4

At the moment a dominated node knows the priority of all dominated nodes in its neighbourhood, it checks if it has the highest priority. If this is the case, it becomes a backbone node and starts sending its own CDSSYNC packet. If a node has not the highest priority it might still be needed in the backbone. Therefore, it starts the alternative path timer and remains dominated. The alternative path timer ensures that all nodes will have a connection to the backbone. A situation where the alternative path timer is needed is illustrated in Figure 6.2.

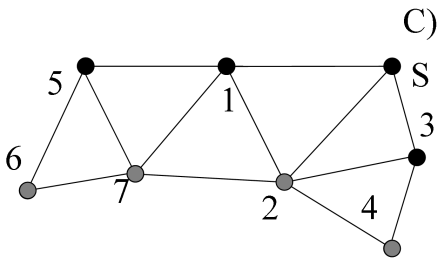
Node 0 is a backbone node. Nodes 1, 2 and 3 are dominated. Node 2 has the highest priority and therefore becomes a backbone node. Node 3 has to ensure that node 6 has a connection to the backbone. Node 3 does not know if there is a connection between nodes 4 and 6 or between nodes 6 and 7. Therefore, node 3 sets the alternative path timer. While this timer is running the



Node ID	Battery Level	Number of Neighbours	Priority
1	***10.9***	***2***	***21.8***
2	10.8	2	21.6
3	11.0	1	11.0



Node ID	Battery Level	Number of Neighbours	Priority
2	10.8	1	10.8
3	***11.0***	***1***	***11.0***
5	***11.2***	***1***	***11.2***
7	11.1	1	11.1



Node ID	Battery Level	Number of Neighbours	Priority
2	10.8	0	0
4	11.4	0	0
6	11.4	0	0
7	11.1	0	0

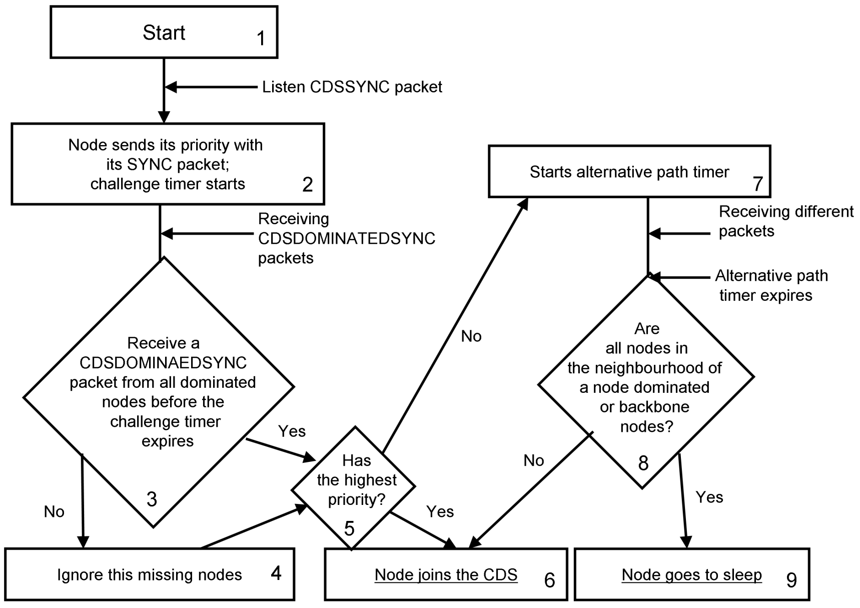
Figure 6.3: CDS building process over the entire network.

node updates its neighbourhood list. There are two possible final states for each node:

1. All nodes in the neighbourhood of a node become dominated or backbone nodes. In this case the node can go to sleep.
2. There are still some nodes in the neighbourhood list of the node that are not covered by the CDS building process when the timer expires. It seems that the node is the only connection to these nodes. Therefore, it becomes a backbone node itself.

An Example CDS Construction

Figure 6.3 shows an example of a CDS building process. In this example all packets arrive before a timer expires. Figure 6.3 (A) shows the situation just after nodes 1, 2 and 3 have received the CDSSYNC packet from the base station S. At the moment node 1 receives a CDSDOMINATEDSYNC packet from node 2 it becomes a backbone node because its priority is higher than the priority of Node 2. Figure 6.3 (B) shows the situation just after nodes S, 2, 5 and 7 have received the CDSSYNC packet from Node 1. Nodes 7 and 2 cannot become backbone nodes because Node 3 has a higher priority than Node 2 and the priority of Node 5 is higher than the one from Node 7. Figure 6.3 (C) shows the final situation. After nodes 1, 3 and 5 have



Start 1
Listen CDSSYNC packet
Node sends its priority with its SYNC packet; challenge timer starts 2
Receiving CDSDOMINATEDSYNC packets
Receive a CDSDOMINAEDSYNC packet from all dominated nodes before the challenge timer expires 3
No
Yes
Ignore this missing nodes 4
Has the highest priority? 5
No
Yes
Node joins the CDS 6
Starts alternative path timer 7
Receiving different packets
Alternative path timer expires
Are all nodes in the neighbourhood of a node dominated or backbone nodes? 8
No
Yes
Node goes to sleep 9

Figure 6.4: CDS building process for each node (except the base station).

become backbone nodes the priority of all other nodes drops to 0. Accordingly, they become non-backbone nodes for the current long sleep period.

Figure 6.4 shows a flowchart of a single node. At the end of this procedure a node knows if it has to join the CDS or not. Except for the base station the algorithm waits first for a CDSSYNC packet. After receiving such a packet it starts sending its priority within its subsequent SYNC packets (Figure 6.4; Box 2). The meaning and the reason for starting the challenge timer is explained in the next subsection. After receiving the CDSDOMINATEDSYNC packets from all dominated nodes in the 1-hop neighbourhood the node checks if it has the highest priority (Figure 6.4; Box 5). If this is the case, it joins the CDS (Figure 6.4; Box 6). Otherwise it remains awake for a fixed amount of time to make sure that all 1-hop neighbourhood nodes has a connection to a backbone. If all 1-hop neighbourhood nodes are connected after the alternative path timer has expired the node decides whether it has to join the CDS or not (Figure 6.4; Box 8).

Periodic CDS Reestablishment

The base station is initialising the establishment of a new CDS periodically to prevent that a node remains a backbone node all the time. All nodes in the long-sleep state have to return to a default T-MAC state several minutes before the base station initialise a new backbone building process. Thus, all nodes have enough time to exchange their SYNC packets to learn their 1-hop neighbourhood again. The routing path remains active as long as the new CDS building process does not overwrite the old tree.

6.4.2 Packet Loss Problem

Broadcast packets can get lost. Therefore, each node has to retransmit packets which are involved in the CDS building process to guarantee a proper CDS setup. A fixed number of retransmissions is not a good approach. In a network with low node density there is, in general, no need to retransmit the packets as often as in a network with high density. Our first simulations have shown that the sum of a small fixed value and the size of the neighbourhood list provides better results. If the number of retransmissions is too low, the CDS building process might be broken. In such a case the CDSSYNC packets never arrive at the designated receivers. Of course this is not a disaster, if there was a complete CDS built once before. However, if a node has never been affected by the building process, it will not know to which node it has to send its packets. All downlink nodes of this node which are not connectable over alternative links will remain in the default sleep and listen states of T-MAC which is a waste of energy. If the number of retransmissions is too high, the time for establishing the tree becomes too long.

Another possible approach is to confirm a CDSSYNC packet with an ACK packet, but this would increase the amount of control messages. Each receiver of a CDSSYNC would have to reply the packet with an ACK packet. This would require in average at least 14 ACK packets to confirm the CDSSYNC of a specific node in a network with an average node density of 15.

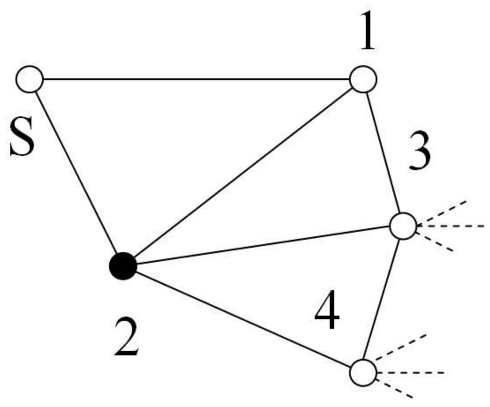


Figure 6.5: Network with a dead node (2). Node S has received a SYNC packet from 1 and 2 just before Node 2 runs out of energy. Therefore a challenge timer is introduced. Node 1 joins the backbone when the challenge timer expires.

A specific node might run out of energy during the CDS building process. To prevent that another node is waiting for an answer of that dead node, N-CDS needs another timer. It is called the challenge timer. The reason to do so is shown in Figure 6.5. Assume, the base station (S) in Figure 6.5 has already collected the IDs of the nodes in its 1-hop neighbourhood. After that, Node 2 runs out of energy. The base station starts the CDS building process thereafter. It broadcasts a CDSSYNC packet with the IDs of nodes 1 and 2. Node 1 is able to broadcast its priority and waits for the priority of Node 2. To prevent that Node 1 waits infinitely, Node 1 starts the challenge timer (Figure 6.4; Box 2). At the moment that this timer expires, Node 1 deletes all dominated nodes from its neighbourhood from which it did not receive any priority information (Figure 6.4; Box 4). In this example Node 1 becomes a backbone node when the timer expires (Figure 6.4; Box 6).

6.4.3 Discussion of N-CDS

The setup of the CDS is rather slow. Especially, if the network density is high, there are many nodes that are concurrently involved in the process. This causes problems because all nodes need the priority of all nodes in their neighbourhood. The delay is affected by adjusting the number of retransmissions of the CDSSYNC packets and by changing the duration of the timers. The N-CDS concept is a heuristic approach for establishing a CDS and, therefore, it rarely establishes a MCDS. However, the simulations have shown that N-CDS establish an adequate CDS which is quite close to the MCDS.

6.5 Multi-Point Relay based CDS (MPR-based CDS)

A backbone node in N-CDS knows only its 1-hop neighbourhood. On the other hand, the more information a backbone node has about its neighbourhood, the more advantage can be taken from the local connectivity. Accordingly, better results might be achieved, if a backbone node knows its 2-hop neighbourhood. Learning the 2-hop neighbourhood with all its connections implies more costs than learning only the 1-hop neighbourhood, though. On the other hand, the CDS building process becomes simpler in particular the timer handling is noticeable simpler.

6.5.1 Distribute Neighbourhood Information

At the moment a node becomes a backbone node, it should know its entire 2-hop neighbourhood or at lest a good approximation of it. Additionally, the node should know the battery level of all 1-hop neighbourhood nodes to avoid that a node would be selected as backbone node all the time. The simplest way to capture this information is to extend the SYNC packet with two fields. One contains the battery level of the sender and the other stores the ID of the sender of the last SYNC packet. This implies, that each node must receive from each 1-hop neighbour at least as many packets as there are nodes in the 1-hop neighbourhood of the neighbour. Thus, each node learns the 2-hop neighbourhood over time. As the focus of our work is on static networks, the learning over time is acceptable.

6.5.2 CDS Building Process

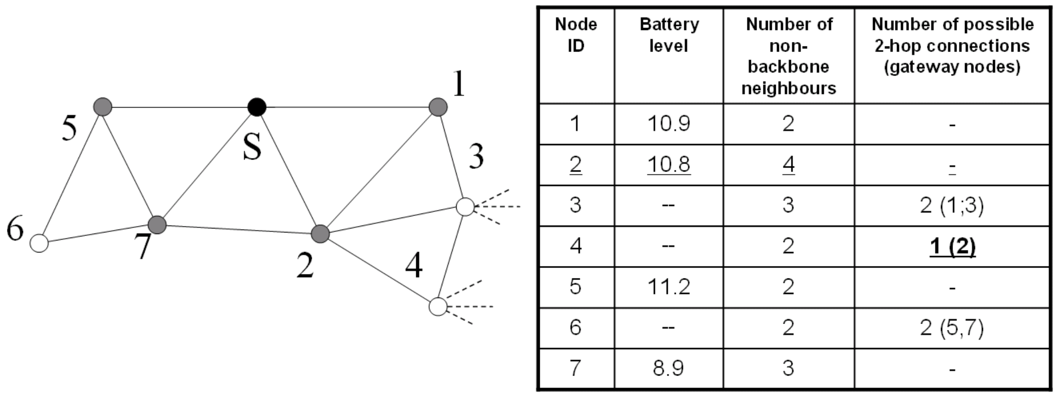
In difference to N-CDS, the backbone node in the MPR-based algorithm defines the next nodes which join the backbone. That means that any backbone node defines in an internal routine the further backbone nodes, at the beginning of its CDS building process. To implement the decision process on a single node, rather than distribute it over neighbouring nodes as in N-CDS, is the reason why the building process of MPR-based CDS becomes simpler. The main task of a dominator is to elect a subset of neighbour nodes as down-link dominators, so that the connectivity to all 2-hop neighbours of the dominator is guaranteed. Therefore the backbone node has first to figure out which non-backbone nodes in its 2-hop neighbourhood have the minimum possible connections. Afterwards the backbone node determines for these nodes the connecting node in the 1-hop neighbourhood. The nodes with highest product of battery level and the number of 1-hop neighbours, i.e., 2-hop neighbours in respect to the backbone node, are elected. The individual steps of the algorithm are as follows:

1. Setup a list with all known nodes. Add two columns to this list. The first contains the battery level and the second contains the number of non backbone neighbours of the specific node.
2. Mark all nodes that are within 1-hop range.
3. Mark all 2-hop nodes which have a connection to a backbone node. That means, they are a backbone node or there is a backbone node in their 1-hop neighbourhood.
4. If all 2-hop nodes are marked, there is no need for an additionally backbone node and the algorithm terminates.
5. Else, count the number of possible 2-hop connections for all unmarked nodes.
6. Select the node as a new backbone node with the lowest value of step 5.
7. Restart the algorithm at step 1.

Figures 6.6 and 6.7 shows an example of this algorithm run on Node S. Step 1 of the algorithm creates the first three column of the list. Step 2 marks all nodes that are within 1-hop range. Therefore nodes 1, 2, 5 and 7 are coloured grey in 6.6. In step 3 in Figure 6.6 no 2-hop nodes are marked, because there are no backbone nodes in the 1-hop neighbourhood of Node S yet. Node 3, 4 and 6 are still unmarked and therefore the algorithm does not terminate at step 4. Step 5 adds the last column to Figure 6.6. Node 4 is selected in step 6 because its number of possible 2-hop connections is only 1. Step 7 restarts the algorithm. The results of the first two steps in the second iteration are the same as in the first iteration. At Step 3 nodes 3 and 4 are marked in Figure 6.7 because they have a connection over Node 2. Node 6 is still unmarked and therefore the algorithm does still not terminate at step 4. Node 6 has two possible linking nodes (5 and 7). Node 5 is chosen at step 6 due to its higher product of battery level and number of 1-hop neighbours. Finally, the algorithm terminates in the third iteration at step 4 because all nodes have been marked in step 3.

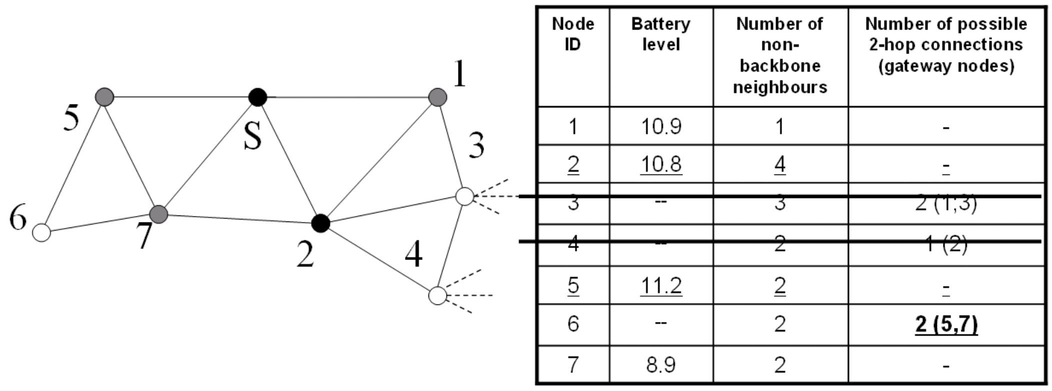
The backbone node broadcasts a CDSSYNC until all new backbone nodes start broadcasting their own CDSSYNC (passive acknowledgement). If the backbone node does not overhear the CDSSYNC from all new backbone nodes, it stops broadcasting CDSSYNCs after a predefined timeout. The CDSSYNC packet contains the usual synchronization information including battery level, last sender ID and the ID of all new backbone nodes. All receivers update their state as follows:

- Sender of the CDSSYNC is a backbone node.
- Receivers destined as backbone nodes in the CDSSYNC become backbone nodes too.



Node ID	Battery level	Number of non-backbone neighbours	Number of possible 2-hop connections (gateway nodes)
1	10.9	2	-
2	10.8	4	-
3	--	3	2 (1;3)
4	--	2	**1 (2)**
5	11.2	2	-
6	--	2	2 (5,7)
7	8.9	3	-

Figure 6.6: Decision table for the first iteration of the CDS building algorithm of Node S: Select Node 2 as first successor because it is the only node with a link to Node 4.



Node ID	Battery level	Number of non-backbone neighbours	Number of possible 2-hop connections (gateway nodes)
1	10.9	1	-
2	10.8	4	-
~~3~~	~~--~~	~~3~~	~~2 (1;3)~~
~~4~~	~~--~~	~~2~~	~~1 (2)~~
5	11.2	2	-
6	--	2	**2 (5,7)**
7	8.9	2	-

Figure 6.7: Decision table for the second iteration of the CDS building algorithm of Node S. Node 6 is the last node which must be connected. Node 5 has a higher battery level than 7. Therefore, Node S selects Node 5 as backbone node.

- Nodes with a direct connection to the sender become non-backbone nodes.

All non-backbone nodes can go to sleep after a certain delay. The delay is necessary, because it might be possible that a non-backbone node receives another CDSSYNC, where it is elected as backbone node. This situation can appear if a backbone node does not know its entire 2-hop neighbourhood because of, for example, a too high packet loss rate. The problem is depicted in Figure 6.8.

Node S sends a CDSSYNC and nodes 1 and 2 are elected as backbone nodes. Node 2 does not know (because of high packet loss rate) that there is a link between nodes S and 3. Furthermore, it does not know that there is a link between nodes 1 and 4. Therefore, it selects among other nodes, also Node 3 as a backbone node to cover Node 4. There are two possible consequences:

1. Node 3 overhears a CDSSYNC from Node 2 before it goes to sleep. In this case Node

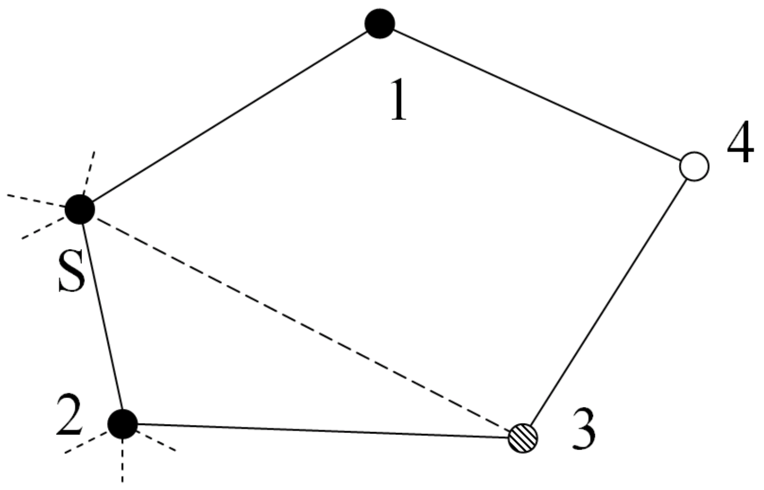


Figure 6.8: Node 2 does not know the link between nodes S and 3 due to packet loss. Node 3 either enters the backbone or not, depending on a timeout.

3 becomes a backbone node as well. This is a waste of energy, but else, there was an algorithm required which would allow a node to decline from becoming a backbone node. This would raise further complexity, though.

2. Node 3 does not overhear a CDSSYNC from Node 2 and enters the sleep state. In this case Node 2 continues to send CDSSYNC packets instead of switching back to shorter SYNC packets. This is some waste of energy at the backbone node, but not critical.

The duration of the long sleep period is the same as for N-CDS. Like in N-CDS, the MPR-based CDS has a period to learn the neighbourhood.

The MPR-based CDS has again been designed for static wireless sensor networks. Therefore, deleting the entire neighbourhood nodes list after a long sleep period is a waste of information. It also takes some time to recollect the entire 2-hop neighbourhood information with the SYNC packets. The MPR-based CDS mainly needs up-to-date information about the 1-hop neighbourhood. Therefore, MPR-based CDS deletes only the information of all 1-hop nodes and all 2-hop nodes where the last notification is older than a given threshold. The best choice of this threshold depends on the network structure. Even if the 2-hop neighbourhood is no longer correct because of mobility or of the dead of one or more nodes, the algorithm still works well. The algorithm makes sure that all 2-hop neighbourhood nodes have a direct link to a backbone node. If such a 2-hop neighbourhood node is no longer active the algorithm still works properly. If a 1-hop neighbourhood node x is no longer available, the CDS setup algorithm could raise problems. It would be possible that a backbone node selects exactly the dead node x as a new backbone node to cover some 2-hop neighbourhood nodes.

6.5.3 Consequences of Packet Loss

As with N-CDS there are some packet loss impacts to be handled. A backbone node broadcasts its CDSSYNC message without an automatic acknowledgement of the receivers. Therefore, the sender has to retransmit its CDSSYNC until it overhears a passive acknowledgement from its subsequent backbone nodes. There are two ways to confirm the transmission. The first one is by overhearing a CDSSYNC from a successor backbone node. If the initial dominator overhears such a packet, it is ensured that this successor has received the CDSSYNC packet. The other way is to confirm the CDSSYNC packet by a CDSACKSYNC packet. If the receiver does not need to send a CDSSYNC packet, because all its 2-hop neighbourhood nodes are already covered, it confirms the CDSSYNC with a CDSACKSYNC. This is a common SYNC packet with a special packet type ID that signalises that this is a CDSACKSYNC packet.

At the moment a CDSSYNC sender overhears a reply from a new backbone node it will mark it in its neighbourhood node list. It removes this node from its subsequent backbone node list as well. Because the CDSSYNC packets contain the IDs of the remaining down-stream backbone nodes, the next CDSSYNC packets become shorter.

6.5.4 Advantage and Disadvantage of MPR-based CDS

The MPR-based CDS needs some memory on each node. The average number of items Δ in the 2-hop neighbourhood list is related to the average number of 1-hop neighbourhood nodes d. To calculate these values following parameters must be given.

- n: Total number of sensor nodes in the entire network.
- r: Transmission range of the nodes.
- s: The side length of the network area.

The average node density d and the neighbour table size Δ are computed as follows:

$$d = \frac{n \cdot r^2 \cdot \pi}{s^2}$$

$$\Delta = d + d^2$$

Each node has to store the link informations about all 1-hop neighbourhood nodes (d) and the information about all 1-hop neighbourhood nodes of the 1-hop neighbourhood nodes (d·d). The equitation for d is taken from [26].

The memory allocation without hashing the data is 96 bits per item if the size of the node IDs is 32 bits. An item contains the ID of the node, the ID of the gateway node and a timestamp of 32 bits in each case. So in total n·96 bits are required for the neighbour table without hashing. There is another table required for the CDS building process. This second table is quite small. It contains the ID and the CDS building status (backbone, non-backbone and not affected) of a node. By hashing the data the size of each item becomes smaller. A hash code of 10 bits is sufficient for most applications. 10 bits allow up to 1024 nodes in the 2-hop neighbourhood.

This provides enough IDs for a network with maximum network density of about 32 neighbours, which is a high density.

A real-world implementation should take care of the variable signal quality. If a backbone node has to select a new backbone node, it normally prefers the ones closest to the border of its maximum transmission range. Such nodes have normally a higher number of uncovered neighbour nodes during the CDS building process. On the other hand, these connections have often poor link quality. Therefore, a real-world implementation should store the link-quality of each 1-hop node. During the CDS building process the backbone node should handle all 1-hop nodes with a poor link-quality as 2-hop nodes if there is a connection over another 1-hop node to these nodes.

MPR-based CDS is assumed to achieve better results than N-CDS because a MPR-based CDS should create a smaller CDS, due to having more information. Therefore more nodes could go to sleep for a long period.

Chapter 7

Global Clock Synchronisation Gravitation

7.1 Problem of Virtual Clustering

T-MAC and S-MAC do not provide global time synchronisation. Only nodes inside the same virtual cluster have their listen periods synchronised. All nodes regularly stay awake for an entire frame. Thus they have the possibility to detect other virtual clusters. If there are two or more schedules from two or more nodes, the overhearing node adapts all schedules, but sends its SYNC packet only in one listen period. With this mechanism the disjoint virtual clusters can be interconnected. However, this means that the interconnecting nodes have a significantly higher power usage. With our proposed algorithm we try to avoid interconnecting nodes with higher power usage by maintaining a global schedule among all nodes.

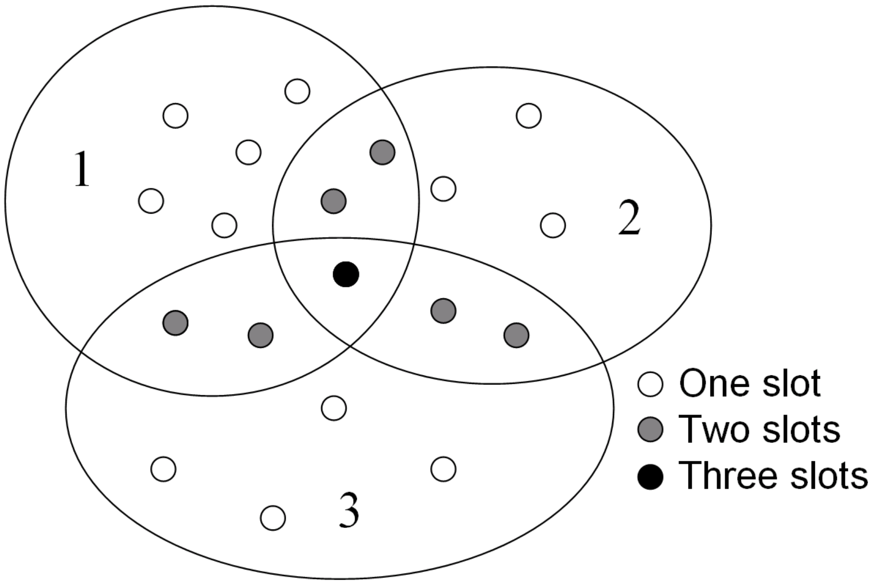


Figure 7.1: Problem of Clustering.

Figure 7.1 shows an example of a network. In this simple case the node in the middle has to take part in three virtual clusters and, accordingly, has to wake up 3 times in a frame. The power usage for synchronization of this node will increase by a factor of 3 without data traffic. The authors of [27] have shown that even in a network with consisting of only 50 nodes, some nodes have to wake-up up to 4 times in a frame. If a node with more than one schedule runs out of energy, there might be no other node which can take over its job. If the nodes with two or three slots in Figure 7.1 run out of energy, the network is divided into multiple parts. Therefore, it is desirable to avoid multiple schedules.

7.2 Local Adaptive Clock Assimilation Scheme (LACAS)

We try to achieve global synchronisation without a master node and without additional network traffic. The proposed solution is called Local Adaptive Clock Assimilation Scheme (LACAS). LACAS avoids the drawback of virtual clusters. It is simple and it performs well in all simulated scenarios. The basic principle bears resemblance to the physical gravitation of material. A huge bulk of material attracts a smaller one more than vice versa. Consequently, both bulks of material fuse after some time. In LACAS the cluster nodes represent the material and the number of sent SYNC messages the gravitation force. Huge virtual clusters, which consist of more nodes, in average send SYNC packets more frequently and thus pull the small ones more than vice versa. Finally, both virtual clusters fuse to one.

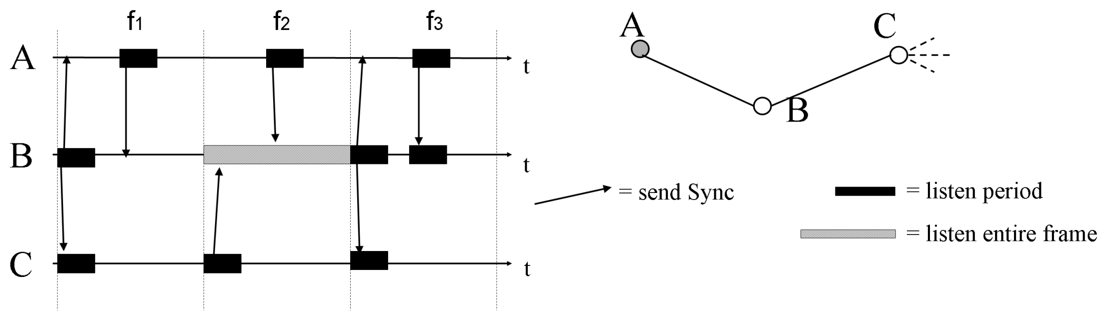


Figure 7.2: Virtual Clustering in T-MAC: Node A belongs to an different cluster than nodes B and C. Node B, periodically stays awake for an entire frame in f_2. It detects a SYNC packet from Node A, during this time. Afterwards, Node B wakes up two times per frame.

After deploying all nodes, each node runs initially a standard T-MAC protocol. Accordingly, a node starts to listen for already running nodes. If it hears another node, it adapts the schedule from the other node. Otherwise, it defines its own schedule and starts sending its SYNC packet at the beginning of its listen period. This implies the creation of a new virtual cluster. Sooner or later a node of a specific virtual cluster overhears a SYNC packet from a node of a different virtual cluster if the node shares the space of both clusters. Normally, this occurs during the period when a node is awake for an entire frame. T-MAC and LACAS react in a different way at the moment a node detects a SYNC packet from a different cluster. This is illustrated in Figure

7.2. In T-MAC, Node B listens to both schedules until death by awaking twice in each frame as soon a s both schedules are known, i.e., after frame f_3 in Figure 7.2.

A node with LACAS never has two schedules. It shifts its own schedule to a given gravitation adaptation percentage α (e.g. α = 5%) towards the schedule of the node it has received a SYNC message from. Additionally, it extends its listen period over both schedules. The length of this listen period is the time span over the own schedule and the schedule learned from the overheard SYNC from the other virtual cluster. Having spanned both clusters, the according node starts to contract its schedule again. Due to the schedule expansion, the power consumption temporarily goes up, but this effort is insignificant compared to the fact that the clusters can be eliminated in this way.

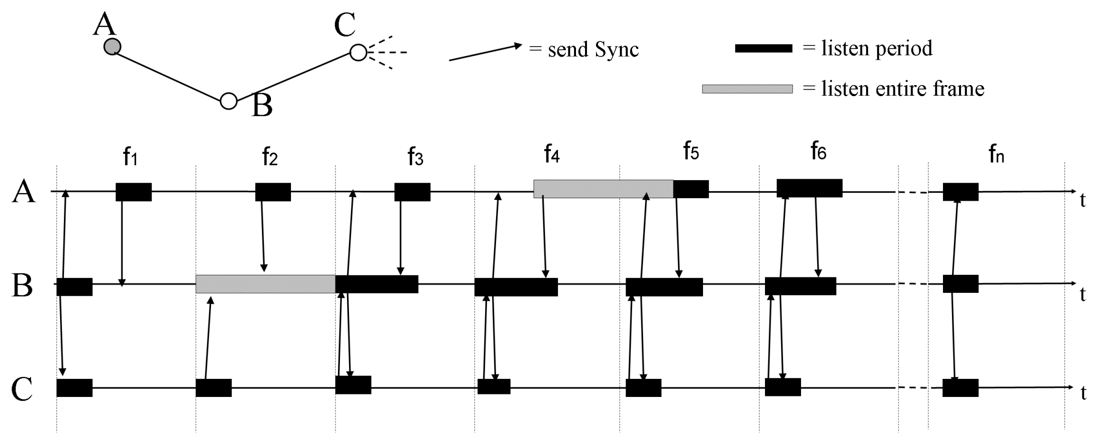


Figure 7.3: Operation of LACAS: Node B stays awake for an entire frame in f_2. During this time it detects a SYNC packet from Node A. It adapts this schedule by α. The next awake period of Node B is extended such that it spans the schedule of nodes A and C.

Figure 7.3 is an example of such a cluster fuse. Node A belongs to a different cluster than nodes B and C in f_1. In f_2 Node B stays awake for an entire frame and detects a SYNC packet from Node A. Therefore it extends its regular listen period over both schedules in f_3. Additionally, its own schedule shifts α towards Node A. Node A stays awake for an entire frame in the second half of f_4 and for the first half of f_5 and detects the SYNC packet from Node B. Like Node B, Node A extends its listen period over both schedules and shifts its schedule towards Node B in f_6. This means, that this small network is connected from now. Between f_6 and f_n both clusters mutually exchange SYNC packets and thus attract each other. Additionally the nodes contract their schedule again. The fusion process finishes in f_n, because all three nodes have the same schedule. Simulations have shown that in general it takes less than 3 minutes to achieve a common schedule in a network consisting of 200 nodes. On the other hand, a sensor network is in general designed for a long network lifetime, e.g., to work for a year or more. Moreover, the physical deployment of the nodes takes some time. Therefore, it is acceptable that it takes some minutes until the clocks are synchronized.

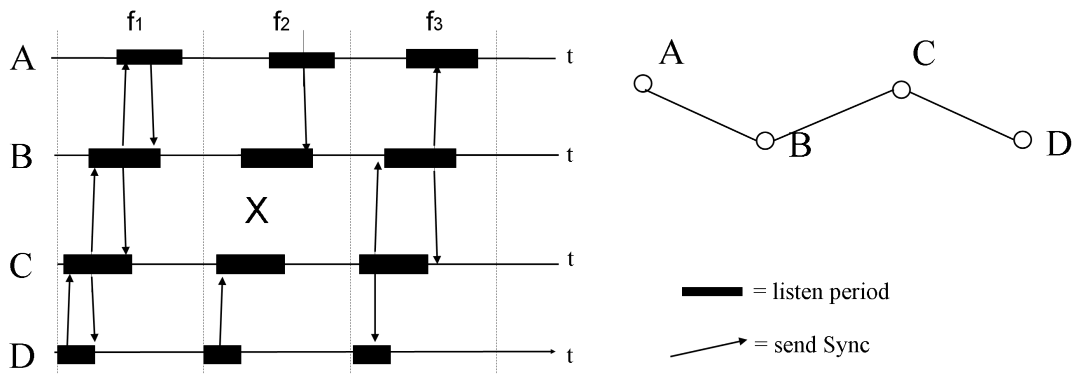


Figure 7.4: Operation of LACAS: The value for α is in this example too high and therefore the networks breaks in two parts.

The value for α has a huge influence considering performance. If α is too low it will take a long period until the schedules fuse to one global schedule. On the other hand, if α is too high, the connection between the nodes might be broken all the time. This problem is illustrated in Figure 7.4. In this example the value for α is too high. All nodes reach their neighbours with a SYNC packet in f_1. Therefore, the network is connected. Afterwards there are too many background noises between nodes B and C and the SYNC packets are lost in f_2. However nodes A and D are able to send their SYNC packets and therefore nodes B and C drift away from each other. This drift is too high and the SYNC packet of node B misses the listen period of the node C in f_3 and vice versa.

This disconnection is not a basic problem. The clusters will be reconnected again when they listen for an entire frame. However, the disconnection increases the period until the schedules fuse. At the moment there is only one global schedule a high α is better than a small one. LACAS has only to compensate the clock drift of each node at this point of time, as it has already eliminated the virtual clusters. Therefore, the value for α switches in our simulations to 50% at the moment the CDS building process is finished. At this time all nodes should be synchronized. Furthermore, all backbone nodes at the leaves of the CDS adapt the schedule learned from the SYNC packet to 100%. Such nodes need to be synchronized only with one node because all other nodes neighbours are sleeping. Therefore, these neighbours will never send a SYNC packet. This feature of LACAS applies only if the network provides a CDS as backbone.

7.3 Discussion of LACAS

LACAS prevents the virtual clustering and saves a lot of energy. The networks of T-MAC and S-MAC are broken at latest at the moment when all nodes at the borders of the virtual clusters die. In general, they die earlier than the other nodes because they have to listen to more than only one schedule. LACAS eliminates virtual clusters. It eliminates also the borders between the clusters. This increases the lifetime of a wireless sensor network.

Chapter 8

Implementation and Evaluation

To evaluate our approach we implemented the algorithms in the Omnet++ network simulator with the Mobility Framework version 1.0a5 (see Chapter 5).

8.1 Simulation Setup and Parameters

We implemented three different MAC protocols.

- T-MAC according to its authors [8]
- N-CDS with LACAS
- MPR-based CDS with LACAS

Nodes in sensor networks are unsynchronised in the beginning. To synchronize the nodes, N-CDS and MPR-based CDS implement LACAS. In all simulations LACAS added no significant overhead to the CDS-based protocols. Therefore, we decided to compare the protocols to the original T-MAC implementation, which provided fully synchronised nodes at simulation start. If T-MAC would be run with LACAS too, it might slightly loose performance, though this would hardly be measurable.

The payload generator and the network structure is the same for all three protocols. The payload generator creates approximately 1 packet per node per minute. There is no data traffic during the first 400 seconds. LACAS and the CDS-based approaches are designed for static networks. Therefore, there is no mobility in the network. The network consists of 50, 100 or 200 nodes. The average number of 1-hop neighbourhood nodes per node ,i.e., the network density, is 10 or 15. The network structure is random. We have chosen for each network size and network density 10 random networks, in which all nodes have a connection (some over several hops) to the base station. Totally, 60 different networks are simulated for the 3 different MAC protocols. Each simulation stops after 100 simulated hours. So, in total 18'000 hours are simulated. Each node has an individual clock drift. The maximum clock drift between any 2 nodes is 2 μs per second. This is still more than the maximum clock drift of the ESB sensor platform from [1]. The total drift is a sum of two parameters. One is the fixed drift of the node. Each node defines its fixed drift at the beginning of each simulation. This fixed drift simulates the incorrectness of each oscillator compared to the correct time. The other value is the variable drift that changes all the time. This is the jittering of the oscillator.

There is no check on the network layer, whether a specific packet has already been sent to the next node or not. This means the generation of duplicates is not prevented. For example, assume node A sends a packet to node B. B receives the data and starts sending its ACK packet. This ACK packet does not arrive at node A because of too much noise. Thus, A has to resend the packet. However B already received the packet and sends the message to the next node. After the retransmission of this packet from A to B, B does not realise that it has already received this data packet. Therefore it will send it twice. This happens rarely. Moreover, all three protocols have the same network layer and are therefore faced with the same problem.

The specification of the simulated hardware is according to experienced values from the ESB sensor platform developed at FU Berlin [1]. All important parameters are listed in Table 8.1.

SPECIFICATION	VALUE
Frame length	610 ms
Power consumption in sleep mode	0.005 mA
Power consumption in idle mode	4.7 mA
Power consumption in send mode	5.2 mA
Power consumption in receive mode	4.7 mA
Buffer size	25 packets
Battery level at start	45 mAh
Path loss model	TwoRay
Antenna height	0.17m
Bit rate	115.2E+3 bits/second
SNR threshold	5 dB
Transmitter power	0.75 mW
Carrier frequency	8.68E+8 Hz
Thermal noise	-98 dB
Sensitivity	-95 dB
Listen entire frame length	all 21.35 seconds
Maximum contention window	2.56ms
TA	6.444ms

Table 8.1: Sensor platform dependent parameters.

The radio consumes most power on a wireless sensor node. A node in sleep state (the radio is off), only uses 5 μA. A node in idle state (the radio is on) consumes 940 times more power (4.7 mA) than in the sleep state. The transmitter power is 0.75 mW. The two-ray model is chosen as path-loss model in our simulations.

With an antenna height of 17cm and a carrier frequency of 8.68E+8 Hz a node has an expected transmission range of about 37m. The interference range is about 52m. The ESB of [1] use, per default, a bitrate of 19200 kbits per second, but the hardware is ready for higher bandwidths of up to 115200 bits. The bandwidth in our simulations is, accordingly, set to 115200 bits per second. The base station is placed always in the centre of the playground.

8.1.1 Packet Definitions

The following tables define the specific packets for all 3 MAC protocols.

NAME	LENGTH IN BIT
Preamble	16
Packet Type	8
Source ID	32
Timestamp	32
CRC	16

Table 8.2: SYNC packet for T-MAC and N-CDS.

Table 8.2 describes the structure of a SYNC packet used in T-MAC and N-CDS. Like all other packets it contains a preamble, a packet type and a CRC part. The preamble makes sure that the receiver interprets the start of the data of this packet correctly. The CRC part checks whether the data of this packet has been transmitted correctly. The packet type informs the receiver that this packet is a SYNC packet. The source ID is the system-wide unique ID of the sender node. The timestamp contains the relative time to send the next SYNC packet. This is a float value between 0 and the maximum length of the frame.

NAME	LENGTH IN BIT
Preamble	16
Packet Type	8
Source ID	32
Timestamp	32
Last Node ID	32
Battery Level	32
CRC	16

Table 8.3: SYNC packet for MPR based CDS.

To setup the MPR-based CDS two additional values are required to learn the 2-hop neighbourhood as well as the battery level of all 1-hop neighbourhood nodes. The according SYNC message to setup the MPR-based CDS is shown in Table 8.3. The two additional parameters are the ID of the last node and the battery level. The ID of the last node has, like the ID of the source, a size of 32 bits. The battery level is a float value and describes the battery level of the sender. This could alternatively be an integer value if the hardware of the nodes provides the battery level as an integer value.

The RTS packet (Table 8.4) has the same structure for all three MAC protocols. RTS is a broadcast packet similar to a SYNC packet. Additionally, it has a field for the ID of the

Name	Length in Bit
Preamble	16
Packet Type	8
SourceID	32
DestinationID	32
NAV	8
CRC	16

Table 8.4: RTS and CTS packets.

destination of the packet, though. It furthermore provides the network allocation vector (NAV) to define the length of the subsequent data packet. This value is useful for all the other nodes which overhear the transmission. Using the NAV each receiver can calculate how long the entire transmission of the data packet will take. They have to be quiet during this time (except for the FRTS just after the CTS packet). A CTS packet has exactly the same structure as the RTS and is again the same for all three protocols.

Name	Length in Bit
Preamble	16
Packet Type	8
SourceID	32
DestinationID	32
CRC	16

Table 8.5: DS, FRTS and ACK packets.

All DS, FRTS and ACK packets (Table 8.5) have a similar structure as a RTS packet. The difference between these packet types and a RTS packet is only then missing NAV. There is no need for this field for this packet type and, therefore, it is not used.

Name	Length in Bit
Preamble	16
Packet Type	8
SourceID	32
DestinationID	32
Length of payload	8
Payload	variable
CRC	16

Table 8.6: Data packets.

Each data packet (Table 8.6) has a payload of variable length. Each field with a variable length needs another field, in which the length of the variable field is indicated. Therefore, a data packet has also a field (length of the payload) in which the size of the payload is defined. In the simulations all data packets have a fixed payload of 164 bits.

NAME	LENGTH IN BIT
Preamble	16
Packet Type	8
SourceID	32
Timestamp	32
Priority	32
CRC	16

Table 8.7: DOMINATEDCDSSYNC packet of N-CDS.

Only N-CDS needs a packet called DOMINATEDCDSSYNC (Table 8.7). It is similar to a default SYNC packet with an extended field called ”Priority”. It is an integer value as long as the battery level of a node is stored as an integer value. Otherwise, it might be mapped to a float value. The priority of a node is the product of the number of free nodes in the neighbourhood and the battery level of the node. It is used for the dominator election. This packet is sent by dominated nodes to exchange their priorities.

NAME	LENGTH IN BIT
Preamble	16
Packet Type	8
SourceID	32
Timestamp	32
length/number of dominated nodes	8
Destination IDs	$n \cdot 32$
CRC	16

Table 8.8: CDSSYNC packet for N-CDS.

The CDSSYNC packet for N-CDS (Table 8.8) is also used in the dominator election process. A backbone node sends a CDSSYNC packet to announce its state and to define the node which will be marked as dominated. The packet has a variable length due to the varying number of destination IDs (n).

The two remaining packet types are the CDSSYNC packet and the CDSACKSYNC packet for the MPR-based CDS. The CDSSYNC packet contains the lists of all subsequent backbone nodes. CDSACKSYNC is the confirmation packet of a CDSSYNC packet receiver which has no further backbone nodes. It has exactly the same structure as the default SYNC packet of the MPR-based CDS. The CDSSYNC packet for the MPR-based CDS is shown in Table 8.9. It has the following characteristics: All fields except the two added field called ‘length/number

NAME	LENGTH IN BIT
Preamble	16
Packet Type	8
SourceID	32
Timestamp	32
LastNodeID	32
BatteryLevel	32
length/number of dominated nodes	8
Destination IDs	$n \cdot 32$
CRC	16

Table 8.9: CDSSYNC packet for MPR based CDS.

of dominated nodes‘ and ‘Destination IDs‘ have the same structure as the SYNC packet of the MPR-based CDS (see Table 8.3). These two new fields have the same structure as the added fields from the SYNC packet to the CDSSYNC in N-CDS (see Table 8.8).

8.1.2 Simulation Details for T-MAC

A real-world T-MAC implementation would have to include virtual clustering. Each virtual cluster decreases the lifetime of the entire network. But, we want to compare our protocols with the standard T-MAC protocol as implemented in the original work, so that we can rely on those results. Due to deployment and radio transmission impacts, virtual clustering is difficult to be simulated. Therefore, virtual clustering would have to be evaluated in real world implementations. In our simulations we do not consider virtual clustering. In the evaluation of LACAS the impact of virtual clustering is considered, though. The T-MAC implementation does not provide routing support. Therefore, at the beginning of each simulation the routing tables for each node are precalculated. To achieve this, each node figures out which node in its neighbourhood is one hop closer to the base station than itself and defines this node as the backbone node, where all packets are routed to. This eliminates all routing issues and additional routing communication on the routing layer. This simple shortest-path algorithm is the same approach as used in the original paper of T-MAC [8]. Any implementation of a routing algorithm would imply impacts that would be difficult to be assessed. Accordingly, our approach is compared to a optimal T-MAC implementation with optimal routing support.

Finally, all nodes in the original paper are arranged in a fixed grid. There is no routing over links with poor link-quality. Because in our simulation all nodes are placed randomly, our shortest-path algorithm only considers links with good link quality to achieve a similar situation as in the original paper. Therefore, all nodes that are farther away than 95% of the maximum transmission range are not considered. If we would consider all possible links in the shortest path algorithm, the benchmark of the T-MAC simulations would be less meaningful because the influence of the poor links is difficult to sort out. As a consequence we compare our protocol to a T-MAC implementation with artificial and optimal solutions for the problems of virtual clustering and routing.

8.1.3 Simulation Details for N-CDS and the MPR-based CDS

SPECIFICATION	VALUE
Rebuild dominator set	each hour
Long sleep duration per hour	50 minutes
SYNC schedule adaptation	5% in starting period
SYNC schedule adaptation	50% if CDS exists
Neighbourhood table size	unlimited

Table 8.10: Technical specifications for N-CDS and MPR-based CDS.

Table 8.10 shows some configurable settings which are valid for the N-CDS and for the MPR-based CDS. For both protocols all nodes have 10 minutes for updating their neighbourhood tables. We assume that the memory of each node is large enough to store the entire neighbourhood table. All nodes have a random initial wake up time between 0.1 seconds and 30.1 seconds after the start of the simulation. This means that a node starts working within this time.

Specific Details for N-CDS

In N-CDS the challenge timer is set to 30 seconds. After a node becomes dominator it sends its relevant information 12 times plus once more for each node in the 1-hop neighbourhood. This is a rather high value, but the additional cost is low and it reduces the risk of breaking the CDS setup process. Before a node, which does not become a backbone node, goes to sleep it sends a CDSDOMINATEDSYNC packet only 5 times plus once more for each node in the 1-hop neighbourhood. The simulations wait 200 to 230 seconds until the CDS building process is started for the first time. N-CDS needs this time to let the nodes synchronise and to learn the 1-hop neighbourhood.

Specific Details for MPR-based CDS

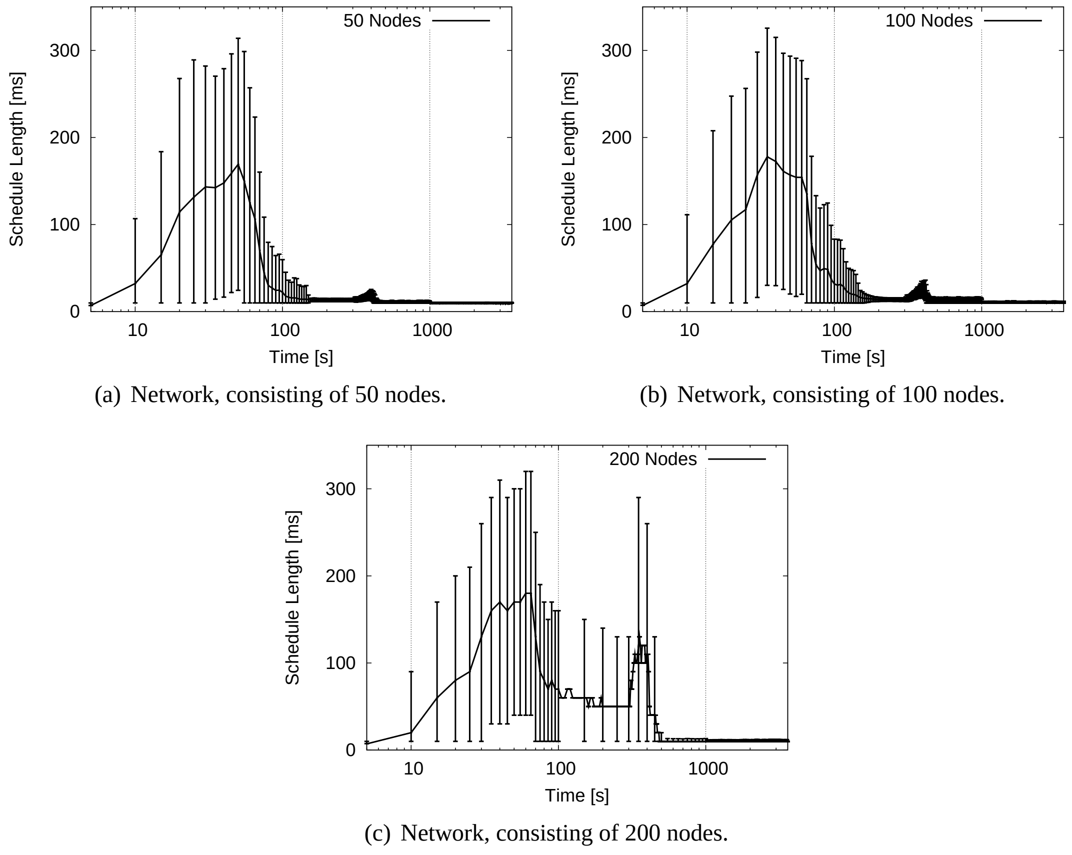
In the MPR-based CDS the simulations wait 300 to 330 seconds until the CDS building process is started for the first time. MPR-based CDS needs this time to let the nodes synchronise and to learn their 2-hop neighbourhood.

8.2 Simulation Results

The most important evaluation parameter we investigate is the power usage of the nodes. This value is analysed concerning two different aspects. First, there is the overall battery usage which describes how much power the entire network requires to fulfil all requested operations. The second value is the battery usage of each node.

8.2.1 Convergence of SYNC Period with LACAS

Before showing the results of the coordinated sleeping, we show the analysis of LACAS. Figure 8.1 shows the average duration of the listen period, i.e., of the schedule length. This is the time difference between the earliest detected schedule and the last schedule plus an additional backoff of 7ms. This backoff ensures that the nodes are able to detect the SYNC packets even after some shifts of the schedules. Three different network sizes, consisting of 50, 100 and 200 nodes, have been evaluated. We calculate the error bars each 5 second except for the simulation with 200 nodes. For readability reasons not all standard deviations are shown in Figure 8.1 (c).



(a) Network, consisting of 50 nodes.

(b) Network, consisting of 100 nodes.

(c) Network, consisting of 200 nodes.

Figure 8.1: Length of the SYNC packet exchange period in a MPR-based CDS network.

The graphs rise first to the maxima schedule lengths which are between 177ms and 189ms. The nodes wake up for the first time between 0 and 30s. Afterwards they choose their own schedule or adapt an already existing one. Accordingly, initially they have only one schedule. Therefore, the values are quite low in the beginning. Quickly the nodes detect other schedules and listen to all of them. At this point the graphs begin to raise to the maxima values. Afterwards the different schedules fuse to one and the graphs drop down again. The CDS establishing process starts at 320s. This decreases the number of SYNC packets and therefore the graphs rise to a local maximum. Without optimization, the gravitation adaptation percentage α would remain 5% (see Chapter 7) and the graphs would converge to 13ms. Due to the CDS algorithm it is possible to change the factor to 50%, though. Thus, LACAS finally converges to 10 ms. The duration of the SYNC packet exchange period in T-MAC is 7ms. According to the real world implementation of S-MAC with 50 nodes done in [27] some nodes have to wake-up up to 4 times in a frame. In a rather poorly performing network the authors have shown that 66% of all nodes follow 2 schedules and 34% follow even 3 schedules. We assume that these schedules are disjoint. T-MAC und S-MAC do not differ concerning virtual clustering. To give an idea of the power savings that could be achieved with LACAS, we estimate the performance of MPR-based CDS enhanced with LACAS and T-MAC with virtual clustering for the given network. T-MAC with virtual clustering requires a power supply of 3.85 mAs per frame over all network nodes and without data traffic. The equitation to calculate the average schedule costs Δ in T-MAC looks as follows:

$$\Delta = 50 \cdot 4.7mA \cdot ((2 \cdot 0.007s \cdot 0.66) + (3 \cdot 0.007s \cdot 0.34)) = 3.85mAs$$

The entire network consisting of 50 nodes needs 235mA in idle mode. This value must be multiplied with the mean schedule length (listen period length) in a frame. The schedule length of T-MAC consists of the time spent to follow all known schedules. On the other hand, MPR-based CDS with LACAS only requires 2.35 mAs (235mA $\cdot$ 10ms) in a frame if all nodes are awake. With LACAS the schedule length is fixed to 10ms for every network node. In 100 hours or more this would make a difference. However, also compared to the other experiments done in [27], LACAS would perform similar or better than an implementation with virtual clustering. In the next section the results concerning the CDS evaluations are shown.

8.2.2 Power Savings of N-CDS and MPR-based CDS

A good protocol tries to reduce the power consumption of some central nodes (those nodes with high power consumption), because the lifetime of a wireless sensor network is limited by the lifetime of such nodes. A network protocol with a low total power usage over the entire network, but containing some nodes which high battery usage could also fulfil its task, if there is a possibility to exchange the nodes which are almost out of battery power. In such a case the network connectivity could still be guaranteed. However, if this is not possible, it is very important that the load is uniformly balanced over all network nodes to avoid the fast depletion of specific nodes. The MPR-based CDS and N-CDS try to achieve this by rebuilding the CDS once per hour.

Figures 8.2, 8.3 and 8.4 show the average battery usage per MAC protocol and network size. The nodes in the simulations with 100 nodes have more power in their batteries than the nodes in the simulation with 50 nodes. The capacity is extended from 40 mAh up to 55 mAh. Otherwise there would be nodes running out of power before 100 simulation hours have finished. Hence, nodes in simulations with 200 nodes need even more power, so that the capacity for each node is extended to 75 mAh. These parameters ensure that no node runs out of power. The values for the power usage of each node are identical with the power usage values of the ESB sensor platform from Scatterweb [1].

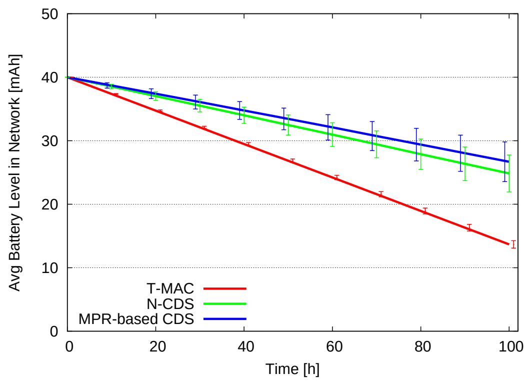


Figure 8.2: Average battery level per network for all simulations with 50 nodes.

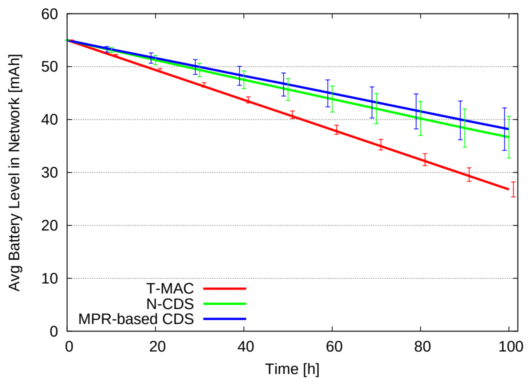


Figure 8.3: Average battery level per network for all simulations with 100 nodes.

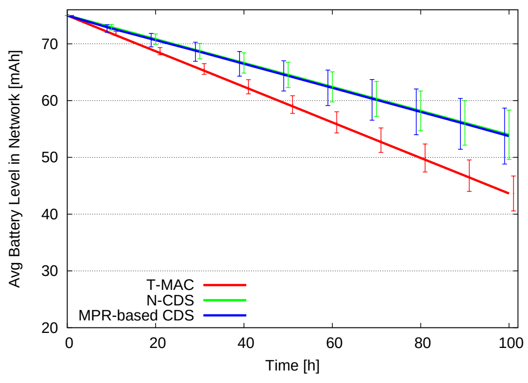


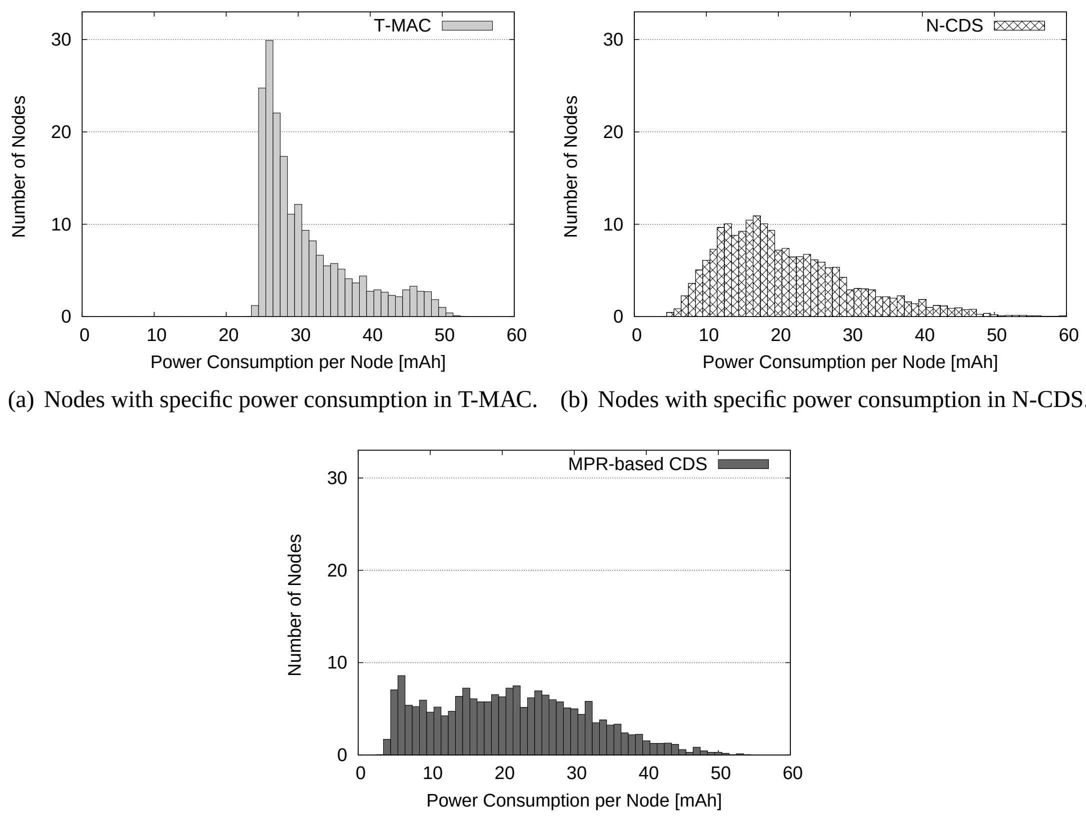
Figure 8.4: Average battery level per network for all simulations with 200 nodes.

The gradients of the graphs of N-CDS and MPR-based CDS are quite similar, although MPR-based is a little bit more efficient. T-MAC does not achieve these values. The figures above show that T-MAC uses more power than the two other protocols. The higher the number of nodes, the more similar are the results for MPR-based CDS and N-CDS. An explanation for this observation is presented in the Subsection 8.2.5, later in this work.

8.2.3 Distribution of Power Savings of N-CDS and MPR-based CDS

The battery usage per node gives an answer to two questions. First, it indicates the level of balance of the routing. Second, it shows that LACAS adds no drawback concerning the question of additional power usage. Figure 8.5 displays the power usage per node over all experiments with 200 nodes. We round the power usage of each node to an integer value and count the number of nodes with equal values. The figure shows that N-CDS and MRP-based CDS outperform T-MAC. T-MAC does not support long-sleep periods. Accordingly, there are no nodes with low power footprint, which can be well observed in Figure 8.5. Moreover, the maximum number of nodes (about 30) have a power consumption of approximately 25 mAh and all remaining nodes have even higher power consumptions. On the other hand, in N-CDS and MPR-based CDS networks, many nodes have low power consumption footprints. Moreover, the CDS-based approaches also have fewer nodes with high power consumption. The graphs for the network with 100 or 50 look similar to Figure 8.5. Hence, there is no significant difference between networks with higher and lower density.

Nodes with high power usage have a high routing traffic. For these nodes in N-CDS and MPR-based CDS no alternative path was available. Hence, these nodes can never sleep for a long period. However these nodes are rare.



(a) Nodes with specific power consumption in T-MAC.

(b) Nodes with specific power consumption in N-CDS.

(c) Nodes with specific power consumption in MPR-based CDS.

Figure 8.5: Number of nodes with a specific power consumption in a network consisting of 200 nodes.

The high number of nodes with low power usage in N-CDS and MPR-based CDS shows that there are many nodes of N-CDS and MPR-based CDS in long sleep state. Such nodes need less than 15 mAh during a period of 100 hours. If the radio of a node was switched on for the whole 100 hours, it would consume 470 mAh even without any traffic.

It remains to be highlighted that T-MAC would have a higher power consumption if there was more than one virtual cluster, because the nodes at the edges between two clusters would use at least as twice as much power than the other nodes.

8.2.4 Impact of Network Structure for N-CDS and MPR-based CDS

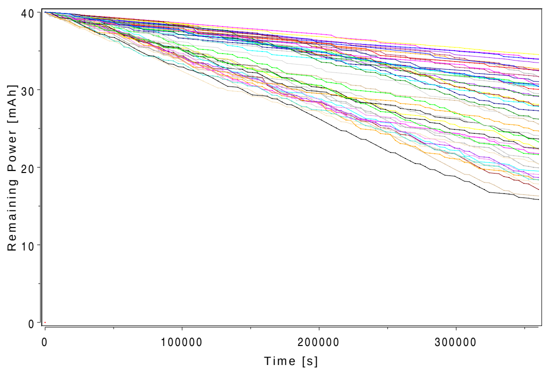


Figure 8.6: Inpact of a good network topology on N-CDS (50 nodes).

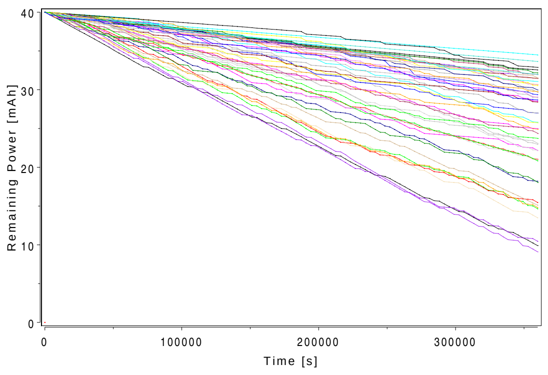


Figure 8.7: Inpact of a bad network topology on N-CDS (50 nodes).

Figures 8.6 and 8.7 show specific simulation runs of N-CDS. One figure is one simulation. It shows the battery level of each node over 100 hours. The comparison of both figures shows that the benchmarks are related to the network structure. The simulator captures the battery level of each node exactly once an hour. Therefore, it looks a bit serrated. If the simulator would grab the results, for example, secondly, the graph for a non-backbone node would look like an oblique staircase due to the higher power usage of the node during the CDS building process. The difference in the power usage per node is higher in the second figure than in the first figure. This depends on the structure of the network. The three nodes at the bottom in Figure 8.7, have to remain almost all the time in the backbone and therefore they will die early. The first network (Figure 8.6) does always have alternative routing paths to ensure that each node is able to leave the backbone sometimes. Accordingly, all nodes are able to remain outside the backbone sometimes and no nodes are heavily charged.

Figures 8.8 and 8.9 show that MPR-based CDS faces the same problem of highly varying power consumptions over the nodes. On the other hand, simulation runs with T-MAC (Figures 8.10 and 8.11) do not show this behaviour. This means that T-MAC does less depend on the network structure. This explains also why the confidence intervals of T-MAC in Figures 8.2 to 8.4 is smaller than for the other two protocols. The optimal routing of the T-MAC implementation reduces the statistical spread of T-MAC also. The higher power usage of specific nodes in N-CDS and MPR-based CDS is mainly related to the implementation of LACAS. Each node has to remain awake for some few milliseconds more than a node in T-MAC without LACAS.

To conclude this subsection, it has to be kept in mind that the average power consumption of T-MAC is higher that for the two CDS-based algorithms. Moreover, the number of poorly performing nodes in the CDS-based approaches is much smaller than for T-MAC.

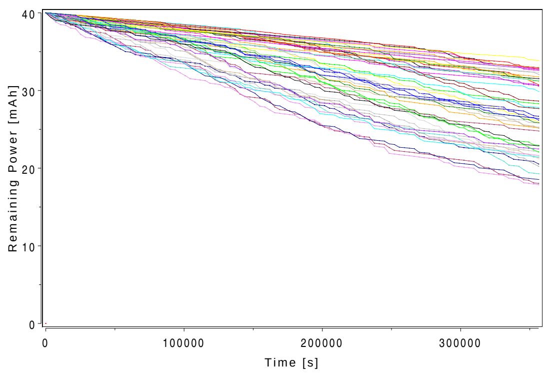


Figure 8.8: Inpact of a good network topology on MPR-based CDS (50 nodes).

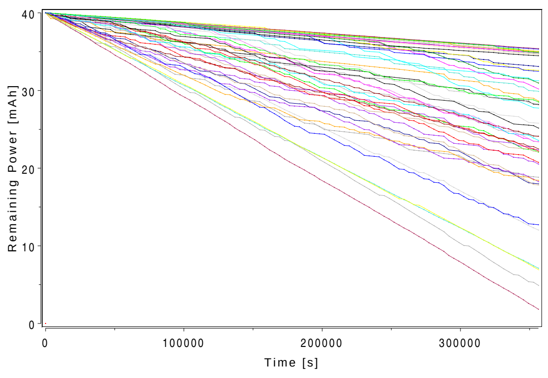


Figure 8.9: Inpact of a bad network topology on MPR-based CDS (50 nodes).

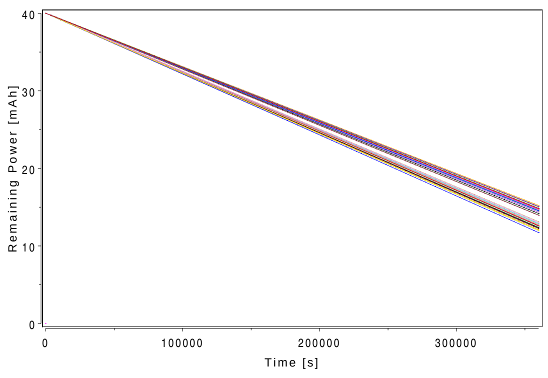


Figure 8.10: Inpact of a good network topology on T-MAC (50 nodes).

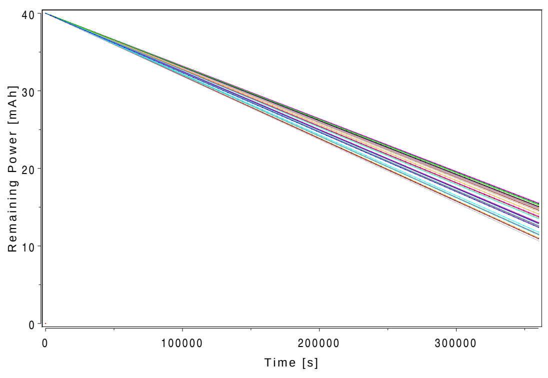


Figure 8.11: Inpact of a good network topology on T-MAC (50 nodes).

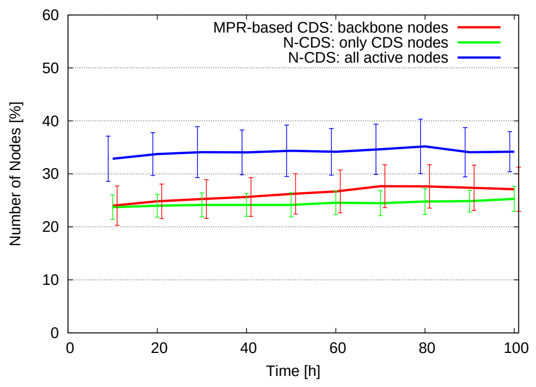


Figure 8.12: Average backbone size in network of 50 nodes.

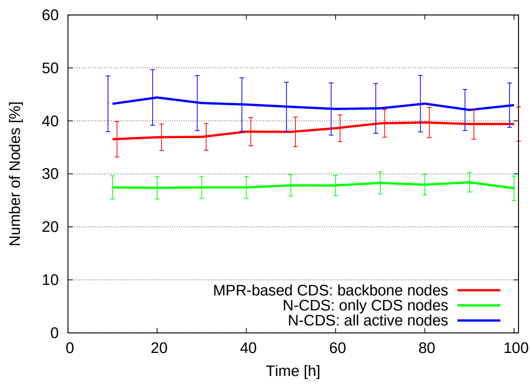


Figure 8.13: Average backbone size in network of 100 nodes.

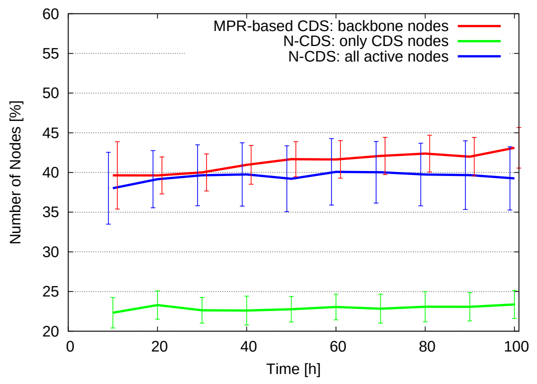


Figure 8.14: Average backbone size in network of 200 nodes.

8.2.5 Size of the CDS

The backbone size has been captured over 10 hours intervals. For example the value at time point 30 hours is the mean value of all values from hour 21 up to hour 30. According to Figures 8.12 to 8.14, N-CDS outperforms MPR-based CDS in the size of the backbone. At a first glance, this is surprising. However, the N-CDS algorithm is not always able to cover the entire network with the backbone. In this case non-dominator nodes have to become active and take over the functionality from the missing backbone nodes. Considering the results, MPR-based CDS outperforms N-CDS in the simulations with 50 and 100 nodes if all active nodes are considered.

MPR-based CDS has a large CDS in the simulations with 200 nodes. The packet loss rate in the simulation with 200 nodes is higher, because of higher amount of data packets and SYNC packets. Therefore, it seems that even over such a long period, not all nodes are able to completely learn their 2-hop neighbourhood. This fact results in an increasing number of new backbone nodes which are selected by the backbone nodes.

There are two possibilities to avoid the situation that the CDS building process does not run through the entire network, but both of them have some disadvantages. The first possible solution is to extend the number of CDSSYNC packet retransmissions. The disadvantage of this solution is that the time required for the re-establishment of the CDS increases. Another solution would be to confirm the CDSSYNC packet with a CDSSYNC-ACK packet. If this packet would be sent as an additional packet in the data period, it would increase the traffic. On the other hand, if the packet would be sent as SYNC packet, it would once again increase the amount of time necessary to re-establish the CDS. Additionally, the sender of the CDSSYNC packet would have to re-send the packet several times. The packet would be longer than a default SYNC packet and would need more power and hence would block the medium for a longer time.

The average size of the backbone is increasing slowly in all simulations. If the system would not calculate the priority with the battery level, this value would remain stable. However, the nodes in the CDS would be static and thus charged much more. At the beginning, all nodes have the same power level. Therefore, the system defines nodes as backbone nodes which have the best location or, in other words, the most connections to other nodes. After a while, the CDS building process tries to avoid these nodes to become backbone nodes because their priority is decreased due to their low battery level.

8.2.6 Number of Lost Packets with N-CDS and MPR-based CDS

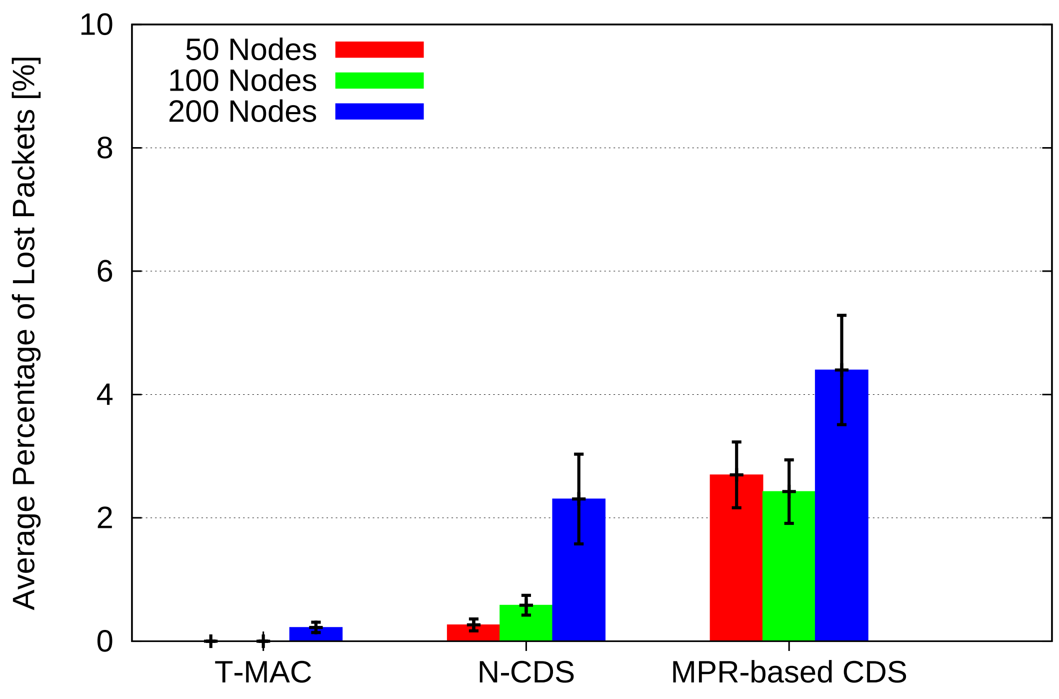


Figure 8.15: Average percentage of packet loss for all simulations.

T-MAC should have the best performance with this simulation setup because T-MAC uses shortest path routing, which only considers connections with good link quality. Consequently, there is no routing traffic at all because the simulation itself defines the global routing table. The higher percentage of packet loss in MPR-based CDS (Figure 8.15) is related to the fact that MPR-based CDS prioritises comparatively many far away nodes as new dominators. Such connections have poor link quality and it takes many attempts to pass messages over such links. A possible solution is presented in the future work chapter. N-CDS is not as strongly affected as MPR-based CDS. All nodes in MPR-based CDS can learn their 1-hop neighbourhood within 10 minutes. N-CDS has less than 30 seconds to exchange the priority. After this time, a timer expires and it defines a new backbone node without considering nodes that have not been updated. There is also a much higher possibility that no packets arrive over poor links. For this reason, the possibility that a node with bad link connection becomes a backbone node is lower than for MPR-based CDS. The average percentage of lost packets per simulation is below 6% for all simulations which is an acceptable value for most applications.

The packet generation is equal for all simulations. Each node creates 6080 packet over 100 hours in average (around 1 packet per minute). The average data throughput of the networks with T-MAC and 200 nodes is around 203.3 packets per minute. The base station of the simulation of N-CDS receives 198.1 packets per minute. Finally, MPR-based CDS has a data throughput of 193.9 packets per minute. These values are more ore less the mirrored values of Figure 8.15. All simulations generate in average 0.8 duplicate packets (see Section 8.1) per minute. The maximum throughput depends on the configuration (for example on the duration of TA).

8.2.7 Transport Delay in a Network with N-CDS or MPR-based CDS

The simulator measures the average transport delay between the source of the data and the base station for all packets. Except for the processing time, the simulator considers all other possible delay sources.

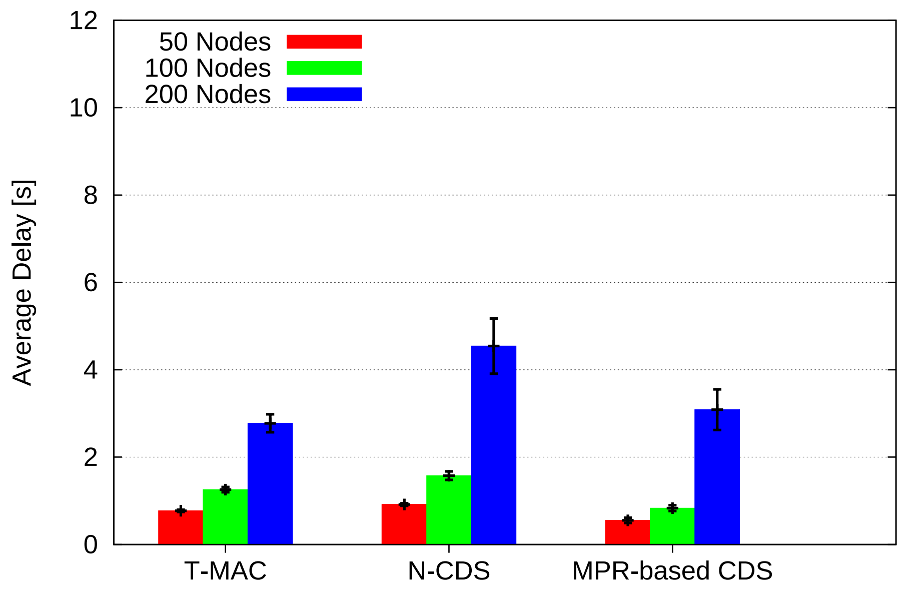


Figure 8.16: Average network delay.

Figure 8.16 shows that MPR-based CDS has a similar average transport delay to T-MAC. The high delay in N-CDS is related to the fact that the average number of hops to receive the base station is higher than in T-MAC and MPR-based CDS. The number of backbone nodes is lower in N-CDS than in MPR-based CDS, but N-CDS has also some non-backbone nodes which do not sleep for a long period. Over these nodes runs some traffic as well. Therefore, it takes more time for each packet to reach the base station.

Chapter 9

Conclusions

In this master thesis a clock synchronization protocol (LACAS) has been provided. LACAS eliminates the drawback of virtual clustering for T-MAC. Another advantage of LACAS is the fact that it is a flat system. There is neither a leader node nor a base station that will lead the synchronization process. The simulations show that the nodes get synchronized within 200 seconds. This is a short time in comparison to the entire lifetime of a wireless sensor network. It will be interesting to verify the functionality of LACAS in a real world implementation. The simulator creates a clock drift which is higher than clock drifts on real sensor nodes such as the ESB sensor boards from Scatterweb [1]. Therefore, LACAS could even be better in a real world implementation.

This master thesis provides two solutions on the MAC layer which increase the lifetime of a wireless sensor network. A perfect solution would eliminate all idle periods of each node without decreasing the maximum bandwidth and transportation delay. N-CDS and MPR-based CDS approximate this. The simulations show that the additional energy consumption of the extended SYNC packets for establishing a CDS is lower than the energy each node can save by sleeping for a long period. Nodes running N-CDS or MPR-based CDS only need about 60% of the energy required by nodes running an optimal T-MAC.

The MPR-based CDS achieves good power consumption results compared to N-CDS and T-MAC. But each node has to store the entire 2-hop neighbourhood. N-CDS achieves similar results as MPR-based CDS without allocating much memory. Finally, T-MAC even enhanced with LACAS would not achieve the performance of N-CDS or MPR-based CDS. But, T-MAC is still a good option if there is more than node-to-sink communication required. Generally, the MAC protocol should be chosen according to the application requirements. Considering static networks, with some redundancy and node-to-sink communication, N-CDS and MPR-based CDS are promising approaches.

N-CDS, MPR-based CDS and T-MAC achieve similar results for the worst-case scenario for N-CDS and MPR-based CDS. This is the case if some nodes never enter a long sleep period, because thus, the network would split up in two or more parts. In this case the additional traffic to establish the backbone for N-CDS and MPR-based CDS does not have more advantage than finding a route to the base station. In normal topologies, i.e., topologies with sufficient redundancy to alter routes from time to time, N-CDS and MPR-based CDS outperform T-MAC. In those scenarios, redundant nodes will take over the task for transmitting data from active nodes and vice versa. Generally, N-CDS and MPR-based CDS will outperform T-MAC as soon as there are many possible routing paths available, and all nodes have the possibility to sleep sometimes for a long sleep period.

The performance of N-CDS and MPR-Based CDS in questions of energy usage of each node is highly related to the structure of the network. It is important that all nodes have the possibility to sleep for a long period. This means that there must exist an alternative routing path replacing specific nodes. A simple solution for this problem is to deploy more nodes around the base station. This increases the redundancy and ensures that nodes with higher routing traffic are able to sleep from time to time.

One of the biggest drawbacks of MPR-based CDS is the fact that the algorithm often selects nodes with poor link quality into the backbone. This means a real implementation of this protocol would contain also a list with the link quality to each 1-hop neighbour node. If the quality is below a limit the algorithm ignores the according node in the backbone selection process.

Chapter 10

Future Work

This master thesis has been introducing new concepts in energy-efficient medium access control and routing support. During the work a number of questions and additional issues appeared. They are listed below and could be investigated in more detail in the future.

- There might be better results if the MAC protocol would not use RTS and CTS packets. In this case the nodes would directly send their data packet and waits for a ACK packets afterwards. In sensor networks the size of a RTS or CTS packet is often not much smaller than a default data packet.
- The simulations results of N-CDS and MPR-based CDS are promising but real world implementation needs to be provided to verify the simulation results. Such an implementation is a must before both concepts could be used in a commercial wireless sensor network.
- It seems possible to extend the MPR-based CDS for wireless sensor network with static and mobile sensors. In this case only the fix sensors would be able to become backbone nodes. This ensures that the backbone would not be broken.
- Parameters such as size of the contention windows, framelength, duration of the TA and drift correction might be optimized both, in general and for specific topologies. By tailoring the parameters to specific networks setups, performance would increase.
- Only links with satisfactory signal strength at the receiver should be considered in the CDS building process. In MPR-based CDS the receiver of a SYNC packet could check the signal strength before it adds the sender of the SYNC to the 1-hop neighbourhood list. That means, if the signal strength is below a threshold, the received packet could be handled as a default SYNC packet from T-MAC. This would eliminate backbone connections with poor link quality. This would need some modifications in the building process algorithm itself.
- When all nodes are synchronized, it could be possible to adjust the clock skew with the synchronization differences. This could increase the results of LACAS once again. The approach could be similar to RBS.

Bibliography

[1] Scatterweb - the self-configurating wireless communication platform. ScatterWeb GmbH. [Online]. Available: http://www.scatterweb.com (*)

[2] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, “MACAW: A media access protocol for wireless lans,” in *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM)*, September 1994, pp. 212–225.

[3] Omnet++. Discrete event simulation system. [Online]. Available: http://www.omnetpp.org (*)

[4] I. Demirkol, C. Ersoy, and F. Alagöz, “Mac protocols for wireless sensor networks: a survey,” *IEEE Communications Magazine*, vol. 44, no. 4, pp. 115–121, April 2006.

[5] J. Hill and D. Culler, “A wireless embedded sensor architecture for system-level optimization. Technical report, U.C. berkeley,” 2001. [Online]. Available: citeseer.ist.psu.edu/hill01wireless.html (*)

[6] IEEE 802.11 Working Group, “IEEE 802.11 wireless local area networks - the working group for WLAN standards.” [Online]. Available: http://grouper.ieee.org/groups/802/11/ (*)

[7] W. Ye, J. Heidemann and D. Estrin, “Medium access control with coordinated adaptive sleeping for wireless sensor networks,” *IEEE/ACM Transactions on networking*, vol. 12, no. 3, pp. 493–506, June 2004.

[8] T. van Dam and K. Langendoen, “An adaptive energy-efficient mac protocol for wireless sensor networks,” in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys ’03), Los Angeles*, Nov. 2003, pp. 171–180.

[9] V. Rajendran, K. Obraczka and J.J. Garcia-Luna-Aceves, “Energy-efficient, collision-free medium access control for wireless sensor networks,” in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys ’03), Los Angeles*, Nov. 2003, pp. 181–192.

[10] L. van Hoesel and P. Havinga, “A lightweight medium access protocol (LMAC) for wireless sensor networks,” in *Proceedings of the International Conference on Networked Sensing Systems (INSS)*, Tokyo, Japan, June 2004.

[11] J. Wu and H. Li, “On calculating connected dominating set for efficient routing in wireless ad hoc networks,” in *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M),* Seattle, Washington, United States, August 1999, pp. 7–14.

[12] P.J. Wan, K. Alzoubi and O. Frieder, “Distributed construction of connected dominating set in wireless ad hoc networks,” in *Proceedings of 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1597–1604.

[13] A. Cerpa and D. Estrin, “Ascent: Adaptive self-configuring sensor networks topologies,” *IEEE Transaction on Mobile Computing,* vol. 3, no. 3, pp. 272–285, 2004.

[14] I. Cidon and O. Mokryn, “Propagation and leader election in multihop broadcast environment,” in *Proceedings of the 12th International Symposium on DIStributed Computing (DISC98), Andros, Greece,* September 1998, pp. 104–119.

[15] D. Zhou, M.T. Sun and T.H. Lai, “A timer-based protocol for connected dominating set construction in IEEE 802.11 multihop mobile ad hoc networks,” in *Proceedings of the 2005 Symposium on Applications an the Internet (Saint 05), Trento, Italy*, 2005, pp. 2–8.

[16] RFC-Group 1305. (1992, March) Network time protocol ntp version 3. Internet Engineering Task Force. [Online]. Available: http://www.faqs.org/ftp/rfc/rfc1305.pdf (*)

[17] GPS-NAVSTAR. (1995, June) Gps navstar - global positioning system. [Online]. Available: http://www.navcen.uscg.gov/pubs/gps/sigspec/gpssps1.pdf (*)

[18] S. Technology. (2007, February) Technical documentation of SiRFstarIII. SiRF Technology, Inc. [Online]. Available: http://www.sirf.com/products/GSC3LTifProductInsert.pdf (*)

[19] L. Huang T.H. Lai, “On the scalability of ieee 802.2.11 ad hoc networks,” in *Proceedings of the 3rd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc ’02), Lausanne,* June 2002, pp. 173–182.

[20] J. Elson, L. Girod and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI 02), Boston, Massachusetts, USA.* UCLA, December 2002, pp. 147 – 163.

[21] S. Ganeriwal, R. Kumar and M. Scrivastava, “Timing-sync protocol for sensor networks,” in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys ’03), Los Angeles,* November 2003, pp. 138–149.

[22] N. Maloani, J. L. Welch and N. Vaidya, “Leader election algorithm for mobile ad-hoc networks,” in *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM 2000), Boston, Massachusetts,* August 2000, pp. 96–103.

[23] Omnest. Simulcraft Inc. [Online]. Available: http://www.omnest.com (*)

[24] B. McLarnon. (1997, December) VHF/UHF/Microwave Radio Propagation: A primer for digital experimenters. TAPR. [Online]. Available: http://www.tapr.org/ve3jf.dcc97.html (*)

[25] S. Loyka and A. Kouky, "Using two ray mutlipath model for microwave link budget analysis," *Antennas and Propagation Magazine*, vol. 43, no. 5, pp. 31–36, Oct. 2001.

[26] H. Takagi and L. Kleinrock, "Optimal transmission ranges for randomly distributed packet radio terminals," *IEEE Transaction Communication*, vol. 32, no. 3, pp. 246–257, March 1984.

[27] Y. Li, W. Ye and J. Heidemann, "Energy and latency control in low duty cycle MAC protocols," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, LA, USA*, vol. 2, March 2005, pp. 676 – 682.

(*) The last date of visit of all listed URLs was in October 2008.