AG VERNETZTE SYSTEME
FACHBEREICH INFORMATIK
TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Master-Thesis

QoS Multicast Routing in Partially Mobile, TDMA-based Networks

Johann Gebhardt

17.03.2015

QoS Multicast Routing in Partially Mobile, TDMA-based Networks

Master-Thesis

Arbeitsgruppe Vernetzte Systeme
Fachbereich Informatik
Technische Universität Kaiserslautern

Johann Gebhardt

Tag der Ausgabe : 18.08.2014
Tag der Abgabe : 17.03.2015

Erstgutachter : Prof. Dr. Reinhard Gotzhein
Zweitgutachter : Dipl.-Inf. Anuschka Igel

Ich erkläre hiermit, die vorliegende Masterarbeit selbständig verfasst zu haben. Die verwendeten Quellen und Hilfsmittel sind im Text kenntlich gemacht und im Literaturverzeichnis vollständig aufgeführt.

Kaiserslautern, den 17.03.2015

(Johann Gebhardt)

Abstract

Wireless communication systems play an increasingly important role in the modern world. Among other things, they begin to replace wired systems in manufacturing scenarios. Wireless networks offer a lot of additional challenges for a system designer, e.g., the interference between transmitting and receiving devices. Due to the limited communication range (which is usually smaller than the interference range) and varying signal quality, nodes usually cannot communicate directly with each other.

An integral part of every communication system is the routing algorithm, which is responsible for the creation of paths along which messages can be forwarded from a source to a destination. In mobile networks the mobility of devices adds further challenges to the system, e.g., because proactive discovery of routes which include mobile nodes is usually not feasible.

In this thesis, a routing protocol for a partially mobile network is developed. The TDMA-based network is the (already existing) communication system designed for the production line of a factory. The algorithm finds routes between different types of nodes, including mobile nodes, and creates a slot reservation schedule without inter-system interference. An acknowledgement mechanism is used to deal with packet loss. The algorithm supports Quality of Service requirements in the form of a minimum bandwidth of one time slot per super slot. Furthermore, it supports several optimization objectives (delay minimization and slot utilization). The algorithm uses multicast trees and local multicasting to reach several destinations at the same time.

As a proof of concept, a simulation for the important parts of the algorithm was implemented and used for its evaluation.

Zusammenfassung

Drahtlose Kommunikationssysteme spielen heutzutage eine zunehmend wichtige Rolle; unter anderem ersetzten sie vorhandene drahtgebundene Systeme im Produktionsbereich von Fabrikanlagen. Drahtlose Netzwerke führen zu neuen Herausforderungen für Systementwickler, wie z.B. Interferenzen zwischen Sende- und Empfangsgeräten. Aufgrund der eingeschränkte Kommunikationsreichweite (welche meistens kleiner ist als die Interferenzreichweite) und der schwankenden Signalqualität ist es in der Regel nicht möglich, Nachrichten von einer Quelle direkt zu einem Ziel zu senden.

Aus diesem Grund ist ein Routingalgorithmus ein integraler Bestandteil jedes Kommunikationssystems. Dieser ist für das Auffinden von Pfaden zwischen Quelle und Ziel verantwortlich. In mobilen Netzwerken wird dieser Prozess noch durch die Mobilität der Geräte verkompliziert. Beispielsweise ist es meistens nicht möglich, proaktiv Routen, die mobile Knoten beinhalten, zu finden.

In der vorliegenden Arbeit wurde ein Routingalgorithmus für ein teilweise mobiles Netzwerk entwickelt. Das auf TDMA basierende Netzwerk wird in dem (bereits existierenden) Kommunikationssystem einer Produktionsanlage verwendet. Der Algorithmus ist in der Lage, Routen zwischen Knoten verschiedener Typen, inklusive mobiler Knoten, zu finden. Zusätzlich wird ein interferenzfreier Slotschedule erstellt. Zur Behandelung von Paketverlusten wird ein ACK-Mechanismus genutzt. Quality of Service-Anforderungen werden von dem Algorithmus in Form einer minimalen Bandbreite von einem Slot per Superslot ebenso unterstützt. Zusätzlich werden verschiedene Optimierungsziele geboten (Latenzminimierung und eine möglichst gute Slotausnutzung). Der Algorithmus nutzt Multicastbäume und lokalen Multicast, um mehrere Empfänger gleichzeitig zu erreichen.

Um das Konzept zu evaluieren wurden für die wichtigen Teile des Algorithmus eine Simulation entwickelt und zu Prüfung genutzt.

Contents

Chapter 1

Introduction

Wireless communication systems play an increasingly important role in the modern world. Today, they can be found nearly everywhere. Many devices, like mobile phones or tablets, only offer very limited support for wired communication. Also, wireless solutions begin to replace wired systems in other application contexts, e.g., in private home networks (W-LAN), or even communication systems in factories.

From a user's point of view, wireless systems offer a lot of advantages. E.g., it is often simpler to connect a device to others in wireless systems. Not having to deal with cable management and the freedom to move a device around at will is very convenient. Furthermore, wireless communication enables a connection to a network in places where a wired connection is not possible or at least impractical, e.g., in large outdoor areas.

From a system designer's point of view, wireless systems lead to additional challenges compared to wired ones. Wireless communication is always broadcasted in a certain area (the communication range) and generally interferes with other communication in an even larger area (the interference range). It is not possible to target a single receiver without disrupting other wireless devices in range. However, it is possible to influence the size of the communication and interference areas by reducing the transmission power. Also, directional antennas can be used to target a specific area instead of transmitting in all directions [27].

Furthermore, wireless signals are influenced by obstacles in their path. Non-moving obstacles can be taken into account while designing the system, but mobile obstacles, e.g., people moving around, cannot be predicted perfectly. Even if the mobile obstacles can be predicted perfectly, calculating their influence on the system is very complicated. Additionally, other wireless systems or machines can interfere with the communication of the planned system.

In practice, this leads to changing signal quality at the receiver, depending on the circumstances. It is also possible that signals are lost completely. The important point is that the wireless medium is generally unreliable. Therefore, according to [10], giving hard guarantees to fulfill some requirements is hard or even impossible. Even statistical guarantees may not always be possible in networks with high mobility [20].

At a first glance, an imaginable advantage of a wireless system could be, that it is easier to communicate with specific other devices. In theory, messages can be sent directly from the sender to the intended receiver. In wired systems, this would

require that the sender and receiver are directly connected to each other. Unfortunately, due to the limited communication range and the varying signal quality, this is not often possible in wireless systems. Therefore, messages usually have to be forwarded by different devices until the destination is reached.

This problem becomes more difficult, when a system contains mobile devices, which are connected to different parts of the network at different times. So, in this case, choosing a static path along which a message can be forwarded to reach a destination, is not possible.

Designing communication systems that can deal with all the problems of wireless communication is a major challenge. Often, some of the problems are ignored, which means that presented solutions are not always convincing [17]. For example, an often ignored problem is that the range in which a transmission interferes with other communications is greater than the range in which it can be successfully received. Unfortunately, some simplifying assumptions usually have to be made to design a practical wireless system, e.g., the *Single Network Property* [14], which states that there are no other networks or protocols that interfere with the system.

In this thesis, a routing algorithm for a specific communication system is developed. The (already existing) communication system is designed for the production line of a factory. The factory contains stationary devices, such as sensors, actuators and general computation devices, as well as mobile devices, e.g., mobile robots. The stationary devices form a permanent, unchanging topology. The mobile devices, on the other hand, can move around and are not part of the fixed topology. Some information about the movement of the mobile nodes is known, for example the general area in which they can move as well as their velocity. This scenario and the organization of the communication system is further described in [19].
The developed routing algorithm is able to create routes from each device to all other devices and organizes the communication so that no inter-system interference is possible. It supports multicast routing, i.e., sending from one source to several destinations, and Quality of Service requirements in the form of a minimum bandwidth. The algorithm can either minimize the end-to-end delay of transmissions or maximize the chance of finding a feasible schedule for a route. Furthermore, an acknowledgement system is used to reduce the problems of the unreliability of the medium. As a proof of concept, a simulation was implemented and used to evaluate the important parts of the algorithm.

The initial problem, additional assumptions and related work will be further discussed in Chapter 2. The algorithm is split into two parts, route discovery and scheduling of transmissions, which will be presented in Chapters 3 and 4. An evaluation of the algorithm can be found in Chapter 5 and the thesis is concluded in Chapter 6.

Chapter 2

Problem Statement

The factory, as described in Chapter 1, can be modeled as a network of computation devices, where all machines and robots are modeled as communicating nodes. The different devices can be grouped into different types, which leads to a network consisting of a number of stationary nodes of miscellaneous types and (one or) a few mobile nodes.

The network is organized as a Service Architecture (SA). In a service architecture, the functionality of the network is modeled as a collection of services. Nodes can create and publish a service or a number of services, that contain their functions. When a nodes needs a service of another node, it can subscribe to it. As soon as the communication between the providing and using nodes is established, the subscribing node receives the results of the offered service.

For example, a sensor can offer its measurements by offering a service which propagates its sensor values. A computation device can supply a data analysis function by creating and offering a service which performs the analysis. If some node needs the processed sensor values, it can subscribe to the service of the computation device. The computation device has to subscribe to the sensor value provider, so that it can process the values and send the results to the node needing them.

A node that subscribes to a service usually continues to use the service for some time. This means that the connection between the service provider and the user is maintained until the service user unsubscribes. Another important property is that often several users subscribe to a single service. It is not unusual that several nodes need the same information or results of the same computations.

In a network model for the described scenario, the nodes obviously need to communicate with each other. However, due to the limited transmission range of the wireless medium, they often cannot communicate directly. Instead, transmissions have to be forwarded through several other nodes. The process of finding a path of nodes from one source to one or more destinations is called routing. The objective of this thesis is to develop a routing algorithm for the described scenario.

The algorithm needs to be able to find routes between all stationary and mobile devices. In addition to finding the routes, it needs to be able to organize interference-free communication along these routes. In this chapter, the problem is further specified and additional assumptions are introduced.

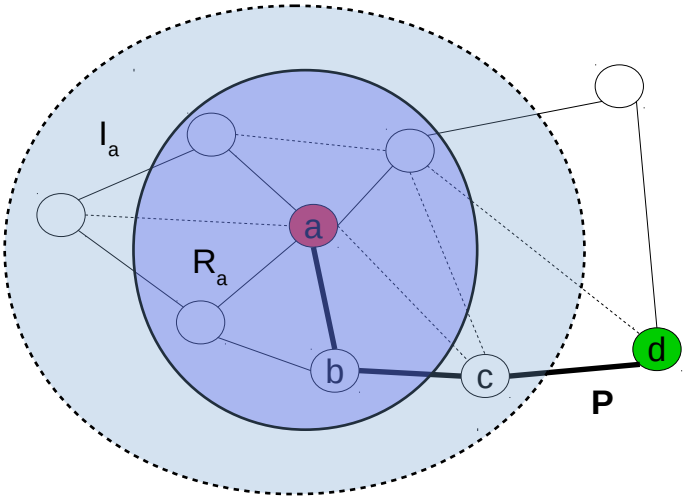


Figure 2.1: An example network showing the communication and interference range of node a and an example path **p**.

2.1 Network Model

In the following, the network is modeled as graph $G = (V, E, I)$ where V is the set of vertices and E and I are sets of edges. Every vertex $v \in V$ represents a node of the network. Every node is associated with a transmission range and an interference range. The transmission range - or communication range - is the maximum range, in which transmissions of a node can be received correctly. The interference range, on the other hand, is the range in which a sending node interferes with the communication of other nodes. Sometimes, nodes are also associated with a sensing range, which is the range in which another node can sense their transmissions. Here, the sensing range will be ignored, as it is not needed for the routing algorithm.

R_i denotes all (other) nodes in the transmission range of a node $i \in V$ and I_i denotes all (other) nodes in the interference range of a node $i \in V$.

Definition 2.1 *The following relations hold:*

- $R_i = \{a \in V | a \text{ is in communication range of } i\}$
- $I_i = \{a \in V | a \text{ is in interference range of } i\}$

For all nodes i, an edge (i, u) has to be added to the set E of edges for all nodes $u \in R_i$. Furthermore, for all nodes j, an edge (j, v) has to be added to the set I of edges for all nodes $v \in I_j$. In all figures in this thesis, the set E is represented with a continuous line and set I is represented with a dashed line, unless specified differently. The interference range is at least as big as the communication range of a node, therefore $R_i \subseteq I_i$ holds for all nodes i [14].

All graphs in this thesis are **undirected**. In an undirected graph, edges have no direction. Therefore, a **link** is defined as a pair $[a, b]$ of nodes that are con-

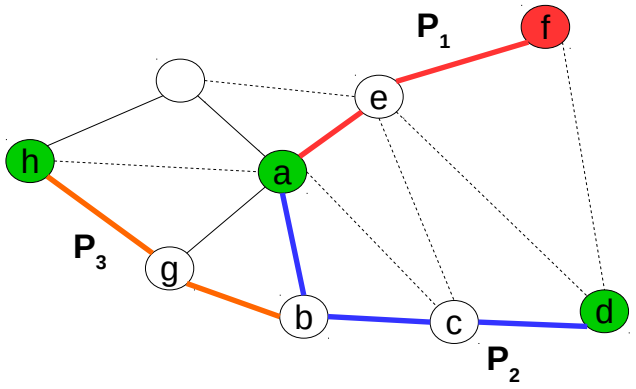


Figure 2.2: An example tree with three paths $p_1 = \langle f, e, a \rangle$, $p_2 = \langle a, b, c, d \rangle$, $p_3 = \langle b, g, h \rangle$.

nected, meaning that $a \in R_b$ and $b \in R_a$. In practice, wireless links are not always bidirectional, as shown in [17].

Sometimes it is assumed that the interference range is the same as the communication range, which means that $E == I$ holds and the network is represented as a graph G = (V,E). While this is not a realistic assumption, it is useful to explain the principles of an algorithm in a small example. Another common assumption is that the interference range of a node contains all nodes in the communication range of its neighbors. In that case the following relation holds:

- $I_x = \{a \in V | \exists y \in V : y \in R_a \wedge x \in R_y \}$

If this is the case, the interference range will not always be displayed in figures in this thesis.

All nodes are associated with a unique **id**. Furthermore, the nodes can have different **types**, with the default being no specific type for all normal nodes. Other important types will be introduced later.

A **path** p is defined as an ordered sequence of nodes $\langle a, b, c, \ldots, d, e \rangle$, where each node forms a link with the directly following node of the path. This means that $b \in R_a$, $c \in R_b, \ldots$, and $e \in R_d$ holds for all nodes in p.

For convenience, it is assumed that it is possible to address the same path $p = \langle a, b, c, d \rangle$ as a sequence of nodes $\langle a,b,c,d \rangle$ or a sequence of links $\langle [a, b], [b, c], [c, d] \rangle$. Hence, $a \in p$ refers to a node a that is part of path p and $[a, b] \in p$ refers to a link $[a, b]$ of path p. The length $|p|$ of path is defined as the number of links in the path.

Figure 2.1 displays an example network, showing the communication range R_a and interference range I_a of node a, and an example path $p = \langle a, b, c, d \rangle$.

A **tree** $mTree$ is an sequence of paths $\langle p_1, p_2, \ldots, p_4 \rangle$, that can be interpreted as a tree from a single source to multiple, different destinations. This means that p_1 is a path from the source to the destination d_1. On the other hand, p_2 is a path from

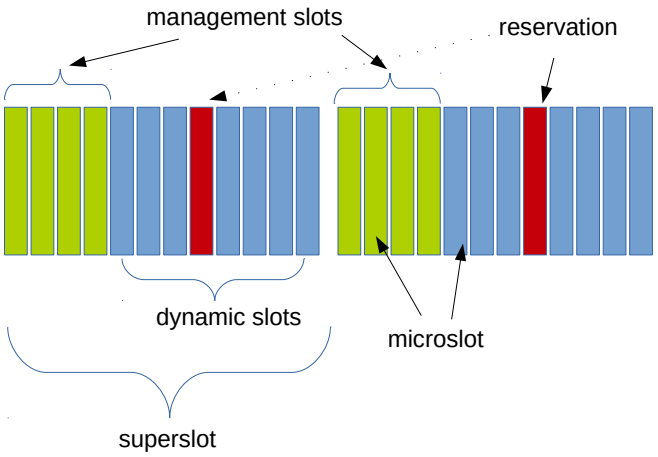


Figure 2.3: Example TDMA slotting with four management slots, eight dynamic slots and one reserved slot.

one of the nodes from p_1 to the destination d_2, while p_3 is a path from one of the nodes from either p_1 or p_2 to the destination d_3 and so on.... Figure 2.2 shows an example network with a tree consisting of three paths $p_1 = \langle f, e, a \rangle$, $p_2 = \langle a, b, c, d \rangle$, $p_3 = \langle b, g, h \rangle$. Note that p_3 is connected to the tree via node b, so the route from the source f to the destination h is $\langle f, e, a, b, g, h \rangle$ instead of just p_3.

2.2 Time Division Multiple Access

The network uses a Time Division Multiple Access [26] (TDMA) strategy for medium access. In the TDMA model, nodes can only transmit or receive when the transmission does not interfere with other transmissions in the interference range of participating nodes. This is achieved by organizing time into slots of a fixed length. In every slot, a node can either transmit, receive or do nothing. To achieve collision free communication, all transmissions have to be scheduled into these time slots (see Section 2.2.1).

The slots are furthermore grouped into micro-, macro- and superslots. Micro slots are the shortest slots. Macro slots contain several micro slots while superslots are a collection of macro slots.

For TDMA to be possible, the network needs to be synchronized, so that all nodes register the start and end of time slots at the same time. For this, tick synchronization is necessary. Proper tick synchronization ensures that all nodes have a synchronized reference tick, with which they can calculate the timings of subsequent time slots. In this thesis, it is assumed that Black Burst Synchronization (BBS) [13] is used. BBS uses periodic tick frames to achieve tick synchronization with a deterministic tick offset. The tick offset is the maximum time that the clocks of two nodes can differ in a network. There exist two different versions – a centralized

and a decentralized one – of BBS. Here, the centralized version, which relies on a master node to handle the synchronization, is used.

The synchronization happens at the beginning of every macro slot. The scheduling algorithm will reserve micro slots for transmitting or receiving frames. The reservations, unless canceled, are the same in every super slot. This means the communications are periodic, which fits the service architecture of the network.

Superslots are groups of macro slots, which can be used if the periods of the macro slots are too short. The macro slots have a maximum length, because BBS needs to re-synchronize the network frequently, otherwise the tick offset would become too large [13]. As the macro slots are only important for synchronization, which is not part of the routing algorithm, they will be ignored in the following. The length of super slots will vary, depending on the examples used. The length of the micro slots is fixed to be long enough for sending one data frame and up to three acknowledgments in a single slot.

Additionally, there is a difference between **management** and **dynamic** slots. Every node is assigned a fixed management slot, in which it has exclusive sending rights. These slots are used to guarantee a collision-free transmission of management traffic, such as route requests. The exact distribution of the management slots is not important in this thesis and therefore left open.

Dynamic slots on the other hand are used to schedule the general traffic. The slots are assigned to the relevant nodes by the routing algorithm. The reservations are kept until the route is canceled.

Figure 2.3 shows an example slot distribution. Here, one super slot contains twelve micro slots and is further split into four management slots, which implies that there are four nodes, and 8 dynamic slots. One slot (slot four of the dynamic slots) is reserved for a transmission in all super slots. The macro slots are not shown.

Space-Division Multiple Access

A possible way of avoiding conflicts is to allow only a single node to transmit in every slot. As long as there is only one transmitting node in the whole network at any time, collision-free communication is guaranteed under the assumptions that there are no other networks or protocols which interfere with the communication (*Single Network Property* [14]). However, as nodes have a known maximum interference range, it is possible to schedule multiple transmissions in the same slot, if they do not interfere with each other. This is called Space-Division Multiple Access (SDMA).

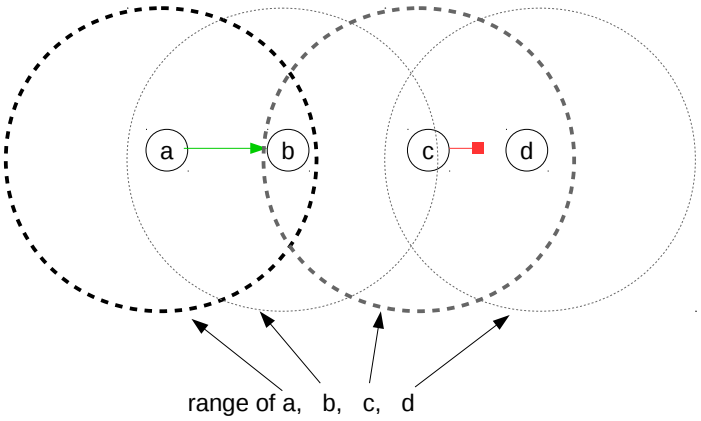


Figure 2.4: An illustration of the hidden station problem.

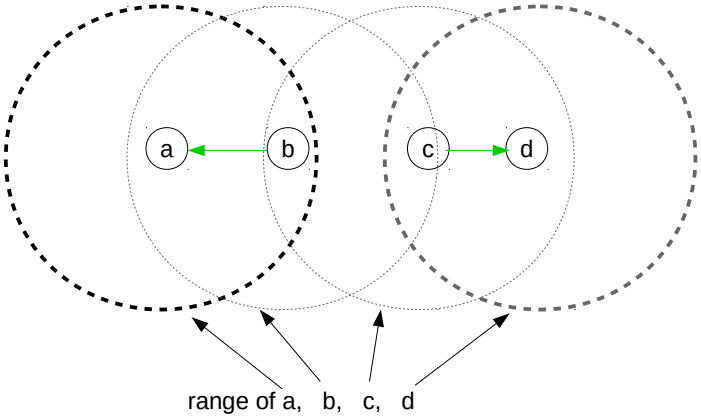


Figure 2.5: An illustration of the exposed station problem.

2.2.1 Interference-free Communication

To achieve interference-free communication in the TDMA model, several problems have to be considered while creating the schedule. First, wireless nodes can usually only either send or receive at any point in time.

Furthermore, transmissions interfere with other transmissions in a certain area (the interference range). If a node is transmitting in a time slot, no other node in its interference range can receive in this slot. This leads to the hidden station problem, as shown in Figure 2.4, where node c must not send to node d as long as a is sending, because c is in the interference range of node b. In the figure it is assumed that the interference range is the same as the communication range.

Another common situation is referred to as the exposed station problem. The exposed station problem states that two nodes can send at the same time even if they are in each other's interference range, as long as their intended receivers are not in the other node's interference range, as illustrated in Figure 2.5. These problems

are often addressed with a Request To Send / Clear To Send (RTS/CTS) mechanism, but can also be avoided by taking them into account while creating the schedule.

Similar to [31], this leads to the following property, which has to be fulfilled in a conflict free schedule.

Property 1 *Sending from node a to b in timeslot t is interference-free iff*

- *a and b do not send / receive in t*
- *(all) nodes in interference range of a do not receive in t*
- *(all) nodes in interference range of b do not send in t*

If TX_a is defined as the set of slots scheduled for transmissions from node a and RX_a the set of slots scheduled for receiving in node a then slot t can be used to send from a to b if:
$t \notin TX_a, t \notin TX_b, t \notin RX_a, t \notin RX_b$ and
$\forall x \in I_a : t \notin RX_x$ as well as
$\forall y \in I_b : t \notin TX_y$.

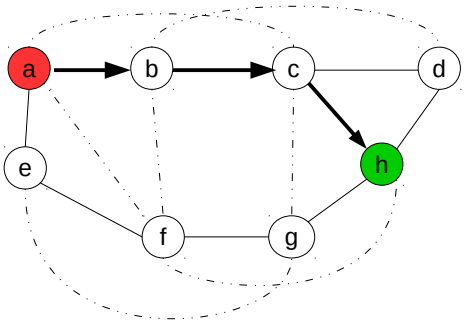


Figure 2.6: An illustration of the interference problem.

Slot:	0	1	2
node a:	S_b	x_{rs}	x_r
node b:	R_a	S_c	x_r
node c:	x_{rs}	R_b	S_h
node d:	x_s	x_{rs}	x_{rs}
node e:	x_r		
node f:	x_{rs}	x_r	x_s
node g:		x_s	x_{rs}
node h:		x_s	R_c

Table 2.1: A feasible schedule according to Property 1.

Figure 2.6 and Table 2.1 display an example schedule for a network. In the table, and in all following tables, the annotation S_b indicates that the corresponding node

sends from itself to node b in that slot and R_a indicates that the current node is scheduled to receive from node a in this slot. When a node is blocked from sending, receiving or both, the appropriate entry of the table is marked as x_s, x_r or x_{rs}. In this example, node a wants to send a message to node h along the path $\langle a, b, c, h\rangle$. Node a is scheduled to send in slot 0, which means that node b needs to be scheduled to receive in the same slot. All other interference neighbors of a are blocked from receiving in slot 0 (nodes c, e and f) and all nodes in the interference range of b (nodes c, d and f) are blocked from sending. Slot 1 is scheduled in a comparable manner. Node b and c send and receive, while nodes a, d and f are blocked from receiving and nodes a, d, g and h are blocked from sending. Node f is only blocked from receiving in slot 1, while node e is still free, so sending from node f to e in slot 1 would still be possible. This is an example of the exposed station problem. Slot 2 is scheduled in the same way as slots 0 and 1.

2.2.2 Interference and Acknowledgements

Sometimes it is necessary to use TDMA together with an acknowledgement mechanism, because even with interference-free communication, transmission failures can happen. To avoid problems, nodes send an acknowledgment (ACK) message back to the sender after every received message. This happens in the same timeslot in which the original message has been sent. From the point of view of the interference problem, this means that nodes act as sender and receiver in every used timeslot. This changes the previous property to the following:

Property 2 *Sending from node a to b in timeslot t is interference-free iff*

- *a and b do not send / receive in t*
- *(all) nodes in interference range of a and b do not send / receive in t*

This is identical to Property 1 applied two times, with both nodes acting as sender and receiver in every timeslot.

Slot:	0	1	2
node a:	S_b	x	x
node b:	R_a	S_c	x
node c:	x	R_b	S_h
node d:	x	x	x
node e:	x		
node f:	x	x	x
node g:		x	x
node h:		x	R_c

Table 2.2: A feasible schedule according to Property 2.

Table 2.2 shows a feasible schedule for the example in Figure 2.6 according to Property 2. Distinguishing between slots blocked for sending resp. receiving is not necessary anymore, because slots are either blocked completely or not at all. The

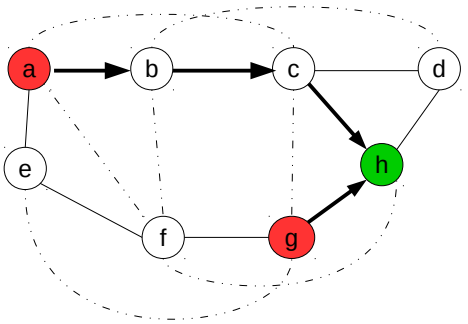


Figure 2.7: A slightly different illustration of the interference problem.

Slot:	0	1	2
node a:	S_b	x	x
node b:	R_a	S_c	x
node c:	x	R_b	S_h
node d:	x	x	x
node e:	x		
node f:	x	x	x
node g:	S_h	x	x
node h:	R_g	x	R_c

Slot:	0	1	2
node a:		S_b	x
node b:		R_a	S_c
node c:	x	x	R_b
node d:	x	x	x
node e:	x	x	
node f:	x	x	x
node g:	S_h		x
node h:	R_g		x

Table 2.3: A feasible (l.) and an incomplete schedule (r.) according to Property 2.

scheduling is similar to the last example. But here it would not be possible to send from node f to node e in slot 1 as previously, because node f is blocked from sending and receiving in slot 1. This shows one of the disadvantages of using the acknowledgement mechanism.

It is assumed that, if several nodes correctly receive the same message, the resulting ACKs are scheduled to be sent in a conflict free manner in the same timeslot (see Section 4.2). This is possible because nodes will know how many nodes need to receive their message, so that the ACKs from the different receivers can be scheduled to be sent at different times (still within the original timeslot)(see Section 4.2). To be more detailed, it is assumed that up to three acknowledgments fit in the same timeslot together with the original message. If more than three nodes need to be reached with one transmission, the data has to be sent several times. How exactly the ACKs are scheduled is out of scope for this thesis.

2.2.3 Timeslot Assignment for Routes

Figure 2.7 shows a slightly different example of the scheduling problem. In this scenario, two nodes (a and g) want to send to the same destination h. Table 2.3 shows two different schedules for this example. The left one shows a feasible schedule, in which nodes a and g transmit in the same slot, which is possible without interference according to Property 2. The right table shows the results of a different scheduling

strategy, in which no feasible schedule is found. Here, nodes a and g are scheduled for different time slots, i.e., there is no free slot left for node c. In a small example like this, it may be easy to find a feasible schedule, but in bigger examples this is not necessarily the case. A possible scheduling strategy could be to start with the shortest path, and then prefer completely unused slots while scheduling the links step by step, which in this case leads to a failed schedule.

Assigning timeslots, according to the above properties, for whole routes is an NP-hard problem [31]. Therefore, heuristics which try to find a conflict-free schedule are necessary. This will be presented in Chapter 4.

2.3 Quality of Service

Communications in a network usually have to fulfill certain characteristics. These characteristics can be formally specified as the **Quality of Service (QoS) parameters** of the network.

Section 2.2.1 explained that transmissions have to be scheduled into time slots to achieve conflict-free communication. In the TDMA model every transmission has a bandwidth requirement of a number of micro slots needed for a transmission in every super slot. In this case, the bandwidth is a QoS parameter. In other medium access strategies, the bandwidth requirement will often be given in bits per second or another suitable measure of the throughput needed for a transmission.

In a SA, it is possible that different services have different bandwidth requirements, e.g., because they have a varying amount of data to send. This could be the case when a service only contains the result of some calculation, (usually a single value to be sent) while another service publishes a whole data set (e.g., sensor values). To simplify the requirements, it is assumed that every service has a bandwidth requirement of one slot, but changing it to a variable number of slots should be possible without changing the fundamental principles of the presented routing algorithm.

The delay for sending from a source to a destination is another useful QoS parameter. The required delay is usually given as unit of time. In TDMA this can be a number of slots, which can be used as a time unit due to their fixed length. The delay is important whenever a task is time critical. In a SA this can depend on the service. A maximum delay is often needed in safety-relevant tasks. For example, a procedure that stops actuators needs to be completed as fast as possible in the case of an emergency. On the other hand, a service that simply outputs the results of some functions, does not need to be completed in a specific time. In the specific scenario the delay of transmissions is a concern. There will be methods which try to reduce the delay as much as possible, during routing and scheduling, but a specific delay requirement will not be given.

There are of course other QoS parameters in other scenarios, e.g, in a wireless sensor network, with battery powered sensors, the energy consumption of nodes is often important. But in this thesis, only bandwidth and delay will be considered.

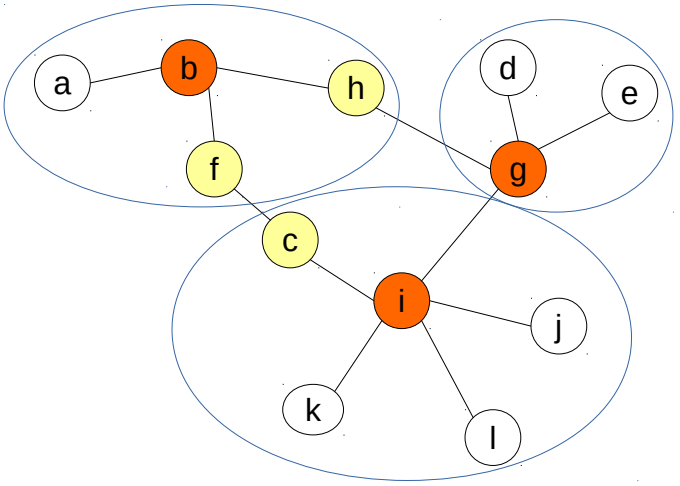


Figure 2.8: An example clustering, with nodes b, g and i as cluster heads and nodes h,f and c as gateways.

2.4 Node Types and Network Topology

The network is assumed to be clustered. From a clustering point of view there are three different node types: cluster heads, followers and gateways. Every follower and gateway node is in **1-hop** distance (communication range) to a cluster head. Cluster heads have a maximum distance of *3 hops* to another cluster head. Gateways are nodes that connect two different clusters to each other. In other words, the network is organized as a *3-hop* connected **1-hop** dominating set.

Figure 2.8 shows a 3-hop connected 1-hop dominating set. More details on the clustering algorithm can be found in [19].

For the developed routing algorithm the clustering is irrelevant. Therefore, these clustering node types will mostly be ignored in the rest of the thesis. More important is the distinction between stationary and mobile nodes.

2.4.1 Stationary Nodes

Stationary nodes, as the name suggests, do not move and form a permanently fixed topology. On a functional level, stationary nodes can be grouped into Full Functional Nodes (FFN) and Reduced Functional Nodes (RFN). FFNs offer a higher degree of functionality compared to RFNs. Among other things, FFNs can act as cluster heads and gateways from a clustering point of view. RFNs, on the other hand, can only act as followers.

As mentioned in the introduction of the current chapter, the network uses a service architecture. Both FFNs and RFNs can act as service providers and users. In order to subscribe to a service, a node needs to know which nodes offer the services. Therefore, nodes register their services at a Service Registry. Only FFNs can become Service Registries. While these clustering and functional types are not

used directly during the routing algorithm, they do play a role while designing the algorithm and will be mentioned when explaining the algorithm.

In the next chapters, three new node types will be defined. These node types are important for the routing protocol and are called routing node types. Here, a short overview over these types will be given. They will be explained in more detail in the following sections.

- **MasterNode:** The proposed routing algorithm uses a centralized routing strategy. The complete routing process takes place in one node, which is called **MasterNode**. There will only be one, predetermined, MasterNode in the whole network.

- **AccessNode:** The AccessNodes are essentially gateways from the stationary network to a mobile node. All traffic to or from the mobile nodes will go through the **AccessNodes**. A network can have multiple AccessNodes for every mobile node. The AccessNodes for every mobile node are predetermined and do not need to be appointed by the routing algorithm.

- **DistributorNode:** Messages sent from a mobile node to several stationary nodes can theoretically be sent from all AccessNodes and need to reach all receiving nodes. This would lead to a complicated tree-like structure with multiple roots. To simplify the matter, a **DistributorNode** will be introduced so that traffic from mobile nodes will be routed to this node and then distributed to all destinations.

The rest of the stationary nodes are treated as normal nodes and do not have a specific routing node type. Furthermore, it is assumed that all nodes know the complete network topology. This includes that all nodes know with which nodes they can communicate, as well as which nodes will be interfered by their transmissions. As shown in [18], gathering this information is possible.

2.4.2 Mobile Nodes

Mobile nodes are classified as an own type. They are not part of the clustering, and therefore cannot be classified as one of the clustering node types. They can act as service providers and users, but not as service registries. As they still need to register their service, they choose a single service registry as their home registry. There are several potential strategies which nodes they could choose. As the optimal home registry is not important in context of the routing algorithm, this decision is left open. A straightforward solution would be to choose the first FFN they contact as the home registry. Another possibility would be to chose a node that is often in range of the mobile node.

Mobile nodes can move in a certain area. They move with a relatively slow and known speed. The geographical position of mobile nodes is known, which may be used to predict their movements to some degree. While this is not used currently, it may prove useful in future work on this subject. The mobile nodes communicate with the rest of the network through the already mentioned access and distribution nodes.

2.5 Related Work

Algorithms for mobile networks are a well studied field. A comprehensive study of the imperatives and challenges of mobile ad-hoc networks can be found in [10]. An overview over general routing protocols for mobile networks can be found in [24] and for wireless sensor networks, which have slightly different requirements, such as a low energy consumption, in [4].

One of the major problems of wireless networks is the interference at receiving nodes. The effect of the interference on the maximum throughput of the network has been studied by Jain et.al. [15].

As described in Section 2.2.1, in a TDMA-based network the interference problem can be solved by scheduling the transmissions into time slots. An analysis of this problem can be found in [14]. This paper states reservation criteria, similar to the ones in Section 2.2.1, which have to be fulfilled to achieve conflict-free transmissions. Another solution to this scheduling problem is presented in [30]. Here, an integer linear programming (ILP) formulation is used to find optimal solutions to the scheduling problem. The authors also define heuristics to solve the problem in shorter time. Another approach is shown in [28], where the scheduling is done according to three slot decision policies to increase the chance of finding a feasible schedule. A variation of these policies, adjusted to the specific requirements of this scenario, will be used in one of the scheduling algorithms in Chapter 4.

In Section 2.3, the Quality of Service requirements for this thesis have been explained. Routes found by the routing algorithm need to satisfy the QoS requirements, e.g., every link of the route must have enough free bandwidth. In this thesis the bandwidth requirement is one time slot per super slot for all links. The generally most common QoS parameter is bandwidth, but some protocols offer different QoS parameters, such as the end-to-end delay of transmissions [5] or define other parameters such as the network cost (in that case the consumed bandwidth) of a route [16]. Other QoS routing protocols for mobile networks can, e.g., be found in [6, 21, 22].

Another requirement for the routing algorithm is the ability to create routes from a single source to multiple destinations at the same time. This is known as multicast routing, which normally, but not necessarily [11], leads to tree-like routes. Example multicast routing protocols are “SOM” [7] or “MAODV” [25]. The advantages of multicasting will be further discussed in Chapter 3.

There are not many routing protocols that fulfill all requirements, i.e., that offer QoS-multicast routing for TDMA-based networks with mobile nodes. Examples for existing protocols are “PSLCB” [31] and “Hexagonal-Tree-Routing” [9]. Both protocols fulfill the requirements and offer bandwidth as the QoS parameter. The “Hexagonal-Tree-Routing” protocol also includes the use of multiple paths to increase the chance of finding a feasible schedule. This means that data can be split into multiple parts and sent along different routes, which reduces the bandwidth requirements of some links of the route. Due to the assumption that transmissions only need one time slot in this scenario (see Section 2.3), this will not be used here.

One problem is that the scheduling of management packets (e.g., route requests) is not part of these two algorithms. Also, the fact that the interference range of nodes is, in general, greater than the communication range, is either ignored or not solved

correctly in the case of "PSLCB" (not enough links are blocked by a transmission). A more detailed analysis of these two protocols has been done as a preparation for this thesis and can be found in [12].

More routing protocols exist for networks with a CDMA-over-TDMA model, as described in [21, 22], in which a CDMA (Code Division Multiple Access) mechanism is used on top of TDMA. In the CDMA model, messages are encoded with special codes so that other nodes, with access to the right codes, can receive the messages correctly despite of overlapping transmissions. This has the effect that the interference between nodes can mostly be ignored, but introduces the need to distribute the codes used for CDMA between the nodes. One example QoS-multicast routing protocol for networks based on CDMA-over-TDMA is the "Wu-Jia" [16] routing protocol, which offers several strategies to find bandwidth-satisfying routes that minimize the delay or bandwidth consumption of the routes. Another protocol for this network model is the "Lantern-Tree based QoS Multicast Protocol" [8], which offers a high chance of finding a bandwidth-satisfying route with the help of multipath tree structures (called lanterns) based on the spiral-fat-tree on-demand multicast (SOM) [7] protocol, which is another multicast protocol for mobile networks that uses trees with increasingly more bandwidth available near the root of the tree. Like "PSLCB" and "Hexagonal-Tree-Routing", these two protocols have been analyzed in [12]. Due to the requirement of a TDMA-based network, these protocols cannot be used in their original form in this thesis. However, some of the ideas used in these two protocols were helpful starting points for the development of the proposed algorithm.

2.6 Objective of this Thesis

The objective of this thesis is to develop a routing protocol for the described scenario. The routing algorithm has to fulfill several requirements:

- it needs to find routes from one source to several destinations
- it needs to find a feasible slot schedule for those routes
  - the schedule needs to be conflict-free
  - the schedule has to fulfill a bandwidth requirement of one slot per link
  - it should be possible to minimize the delay of a transmission

The objective can be partitioned into several sub goals.

2.6.1 Sub Objectives

Routing from Stationary Node to Stationary Node

The first objective is to find routes between stationary nodes, including the possibility to have several destinations for a single source. The case involving mobiles nodes is treated separately.

Routing including Mobile Nodes

Due to the movement of the mobile nodes, the route discovery as well as the timeslot assignments are more difficult. The proposed solution extends the stationary to stationary algorithm, to achieve communication with the mobile nodes with the help of AccessNodes and DistributorNodes.

Timeslot assignments

After a route has been created, timeslots for sending and receiving messages have to be assigned to all nodes that are part of the route. This is handled as a sub objective to increase the modularity of the algorithm.

Chapter 3

Centralized Multicast Routing

In the previous chapter, the requirements for the routing algorithm were explained. One requirement is the ability to send data from one source to several destinations. A simple way to reach more than one destination would be to create several unicast routes. In this case, parts of the paths to several destinations could be identical. This leads to a waste of resources while sending the data. A better solution is the use of **multicast** trees. Here, a tree-like structure with the source of the route acting as the root of the tree and the destinations acting as leaves will be created. This leads to a decrease in resource consumption by avoiding redundant transmissions of messages and corresponding slot reservations.

When using wireless communication, which is a broadcast medium, resources can be saved by sending from one transmitter to several receivers at the same time. As mentioned in Section 2.2.2, this will be used in the algorithm, allowing nodes to send to up to three different receivers at the same time. Figure 3.1 shows an example multicast tree from the source k to the destinations h, e and a. The blue path is the same for all destinations and node g makes use of local multicasting to reach three nodes (h, d and c) at the same time.

Generally speaking, algorithms can be classified as either centralized or decentralized. In an centralized algorithm, the function of the algorithm is executed in a single device, while in decentralized solutions the algorithm is executed separately

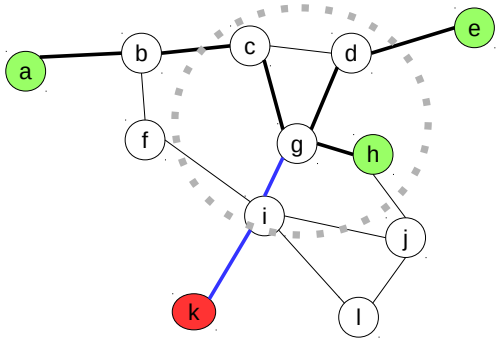


Figure 3.1: An example multicast **tree** with shared paths and local multicast in node g

by all relevant nodes. A centralized approach has several downsides, e.g., it leads to a single point of failure. When the master device fails for some reason, the algorithm cannot be used anymore until a new master has been chosen. On the other hand, a centralized solution also offers advantages. For example, it can lead to a decrease in management traffic compared to decentralized algorithms, because nodes often do not have to exchange information required by the algorithm.

In this thesis, it was decided to use a centralized approach. A new node type, the **MasterNode** is introduced, which handles the route discovery and timeslot scheduling. For the route discovery in this scenario, a centralized solution does not have as many advantages as usual, because of the assumption that all nodes know the complete network topology. With that knowledge, discovering routes is not that complicated. However, the MasterNode will be significantly more important for the scheduling algorithm (see Chapter 4).

One of the remaining advantages of the MasterNode is that it reduces the importance of the clustering and functional node types. For the routing algorithm, the clustering topology of the network is irrelevant, which means that distinguishing between the different clustering node types is not necessary.

One concern for the RFNs is, that they do not have enough computation power to handle the route discovery and scheduling, but with the introduction of the MasterNode, this is not important anymore and all clustering and functional node types are treated equally.

The MasterNode obviously does not know which nodes need which routes. Therefore, it has to be able to receive routing requests from all other nodes. It also needs to inform the nodes of the chosen routes and schedules. While the focus of this thesis lies on the discovery of the routes and their transmission schedule, these two steps cannot be ignored and a brief solution will be presented in Sections 3.3 and 3.4. However, the route request and route confirm phases have not been implemented.

MasterNode

According to the assumptions, the MasterNode knows the topology of the network and the communication and interference neighborhood of all nodes. As the MasterNode handles all route requests, it also knows the complete schedule for all nodes in the network. With this information, the MasterNode has to handle the following tasks:

- receiving route requests from nodes
- discovery of routes and timeslot assignments
- informing nodes about the routes and the schedule

To handle these tasks efficiently, a suitable node has to be chosen as MasterNode. There are several criteria, depending on the application scenario, which can be used to determine a candidate.

One of the disadvantages of a centralized routing approach is that it leads to a single point of failure. In the application scenario for which this algorithm was developed BBS [13] is used. BBS, in the applied form, already has a single master

node and the synchronization fails when that node fails. Without synchronization, conflict-free transmission cannot be achieved in a TDMA based network. Therefore, the network already has a single point of failure. So using the same node as the MasterNode for the routing algorithm does not introduce a new point of failure.

In other scenarios there can be other, better, criteria. A good candidate would usually be a node with a low average distance to all other nodes, to reduce the management overhead. In some contexts, e.g., battery powered sensor networks, the available energy / computation power may be a good criterion, because the master node has to do more work compared to the rest of the nodes.

3.1 Assessment criteria for the Route Discovery

Before designing the algorithm, it has to be decided which objectives it has to fulfill, i.e., what a good multicast tree looks like. There are several criteria that are useful to evaluate a tree. It is not possible to create a single evaluation criterion that fits all application contexts. Therefore, several factors are proposed.

3.1.1 Number of Hops

The first criterion is the total number of hops in the multicast tree. The number of hops in the tree is related to the total number of transmissions and receptions necessary to reach all destinations. Due to the use of local multicasting, the numbers may be different. As sending and receiving data consumes resources (in this case time slots), this is a reasonable factor to approximate the total resource consumption of a route.

Therefore, one objective of the routing strategy is the minimization of the total number of hops in a tree.

3.1.2 Length of Paths

In Section 2.3, it was mentioned that the delay of a transmission is of importance. When using multicast routing, the delay is the time it takes to send a packet from the source to all destinations. Due to the use of TDMA as the medium access strategy, the delay will be measured in the number of slots it takes to complete the transmission. The routes themselves are not the only factors determining the delay. It also depends on the schedule (see Section 4.4), but the routes can be used to get a lower bound for the delay. The lowest possible delay of a path is equal to its length. Therefore, minimizing the length of the paths from the source to the destinations should have positive influence on the delay.

Therefore, the route discovery algorithm tries to optimize the length of the paths from the source to all destinations.

3.1.3 Cost

The last criterion is aimed at networks with a high utilization. Every transmission that is made blocks other transmissions in its scheduled slot in a rather large area.

If more slots are blocked, it becomes increasingly difficult to find a feasible schedule for future routes. Generally speaking, the number of slots blocked by a transmission correlates with the number of interference neighbors of the participating nodes. Therefore, nodes with a low number of interference neighbors have less impact on the scheduling of future routes. On the other hand, the cost also heavily depends on already existing slot assignments. It is possible that all neighbors of a node are already blocked from sending / receiving in a specific slot. In this case, using this slot does not lead to an increased cost. The cost is another criterion that will be kept in mind.

3.2 Common Functions

Before introducing the routing algorithm, several often used functions have to be specified.

- **getShortestPaths(node source, node destination)**: Returns an array of shortest paths from the source node to the destination node. If only one path exists, the array has the size one. The individual paths can be addressed by their position in the array. So path p = getShortestPath(source,dest)[1] refers to the second path. All paths start with the source and end with the destination.
- **minValue(list values)**,**maxValue(list values)**: Returns the minimum or maximum value out of a list of values.
- **minIndex(list values)**,**maxIndex(list values)**:Returns the index of the first occurrence of the minimum or maximum value of a list of values.
- **getDistance(node a, node b)**: Returns the distance from node a to node b.
- **getDistance(node a, node b, path p)**: Returns the distance from node a to b, when using a specific path p.

3.3 Route Request Phase

A route request is issued whenever a node wants to join a multicast tree. In the context of a SA, this is the case whenever a node wants to subscribe to a service of another node. If no other node is subscribed to this service from this specific provider, a new tree has to be created. It is not possible to use a single tree to send different data to different destinations. Therefore, a service provider can be the source of several multicast trees, if it offers different services.
The trees are assigned a tree id, which together with the source node, serves as a unique identifier for the tree. In a SA the tree id can be related to service ids, which are necessary to identify a service. The route requests are sent from the requesting node to the MasterNode, after they have identified the service id and the id of the service provider. They can get this information from their assigned service registry.

The exact operating mode of the service architecture is not part of this thesis and the routing algorithm should be as general as possible. Hence, every node, which knows the right tree id and the id of the root of the tree, is able to request a route.

The MasterNode receives the route request and either creates a new tree, or expands an existing one (see Section 3.5).

3.3.1 Packet Types

The basic idea for the route request phase is to use the management slots of the nodes to send route request packets (**RRP**) to the MasterNode. One of the assumptions is, that every node has complete knowledge of the network topology as well as one management timeslot in which it has exclusive sending rights (conflict free transmissions). With the knowledge of the topology, every node can calculate a shortest path from itself to the MasterNode. The packet is then sent to the next hop on this path during the management slot. The next node can also calculate the shortest path to the MasterNode and repeat the process.

The RRP is defined as follows:
RRP(nodeId, nextHopId, treeRootId, destinationNodeId, treeId, requestId)

- **nodeId**: The id of the node currently sending the packet. This field will be updated by every node forwarding the packet as it is used to properly address the acknowledgments.
- **nextHopId**: The id of the next destination of this packet, which is necessary for the receivers of the packet to determine whether they are responsible for forwarding the packet.
- **treeRootId**: The id of the root of the requested multicast tree, used to identify the tree.
- **destinationNodeId**: The id of the new destination which has to be added to the tree.
- **treeId**: An id of the multicast tree, used together with the treeRootId to identify the requested tree.
- **requestId**: An incremental id to identify this request, which is necessary to distinguish between multiple requests of the same node.

It was already mentioned that the RRPs are acknowledged. Therefore, an acknowledgement (ACK) has to be specified:
ACK(nodeId,receiverId,requestId)

- **nodeId**: The id of the node sending the ACK.
- **receiverId**: The id of intended receiver.
- **requestId**: The same as the requestId from the RRP that triggered this ACK. Together, these fields can be used by the receiver of the ACK to identify which RRP is acknowledged.

3.3.2 Protocol

The first step of the protocol is the creation of the RRP packet, which is done by the node requesting a route. It is assumed that the requesting node knows the root and id of the tree it wants. This is shown in the following listing.

```
/* Creates an RRP packet for a given tree mTree with a known root */

createRRP(node root, id treeId):
        /* Get an array of shortest paths from itself to the MasterNode
           */
        paths = getShortestPaths(myNodeId,MasterNode)
        /* Choose one of the paths, which one is irrelevant */
        shortestPath = paths[0]

        /* set all values of the packet*/
        RRP.nodeId = myNodeId
        RRP.nextHopId = shortestPath[1].getId() // next hop on the path
        RRP.treeRootId = root.getId()
        RRP.destinationNodeId = myNodeId
        RRP.treeId = mTree.getId()
        RRP.requestId = lastRequestID+1

        /* schedule the packet for the next management slot */
        schedule(RRP)
        /* set a timeout for this RRP and wait for the ACK */
        setTimer(RRP,lengthOfASuperslot)
```

The function *createRRP* prepares and schedules the RRP to be sent in the next management slot. The first node of the first shortest path is chosen as the next hop. It is not important to choose a specific one, because all paths have the same length and no SDMA is used during the management slots.

Nodes receiving a RRP in any time slot need to forward it, which is shown in the following listing.

```
/* Sends an ACK and forwards the packet */
receiveRRP(RRP):
        /* check if itself is the intended receiver */
        if(nextHopId == myNodeID):
                /* send ACK */
                sendACK(myNodeId,RRP.nodeID,RRP.requestID)
                /* send RRP */
                forwardRRP(RRP)
```

ReceiveRRP first checks if the node is the intended receiver of the packet. In that case, it creates an ACK which will be sent immediately (i.e., in this time slot) and forwards the RRP:

```
/* Called when a node receives an RRP and is the intended receiver */
forwardRRP():
        /* Get an array of shortest paths from itself to the MasterNode
           */
        paths = getShortestPaths(myNodeId,MasterNode)
        /* Choose one of the paths, which one is irrelevant */
        shorestPath = paths[0]

```

```
8           /* update necessary values of the RRP packet */
9           RRP.nextHopID == shortestPath[1].getId()
10          RRP.nodeId == myNodeID
11
12          /* schedule the packet for the next management slot */
13          schedule(RRP)
14
15          /* set a timeout for this RRP and wait for the ACK */
16          setTimer(RRP, lengthOfASuperslot)
```

To forward the RRP, the node has to determine the shortest path from itself to the MasterNode, update the relevant fields of the packet and schedule it to be sent in its next management slot. Additionally, the node sets a timer for one super slot. If it does not receive the ACK within one super slot, the RRP has to be retransmitted. Waiting for one super slot is enough, because every node has at least one management slot per super slot. Waiting for less time could mean that this node's management slot is after the timer runs out (e.g. if this node's management slot was directly before it received this RRP).

The ACK creation and transmission is relatively straightforward:

```
1 /* Creates and sends an ACK immediately */
2
3 sendAck(id nodeId,id receiverId, id requestID):
4           /* set ACK values */
5           ACK.nodeId = nodeId
6           ACK.receiverId = receiverId
7           ACK.requestID = requestID
8
9           /* send the ACK immediately */
10          send(ACK)
```

The ACK is created and the relevant fields are set to the values determined in the *receiveRRP* function sent immediately (in this micro slot).

Finally, ACKs have to be received by other nodes:

```
1 /* Called when receiving an ACK */
2 receiveAck(ACK):
3           /* identify relating RRP packet */
4           search RRP where:
5                   RRP.nextHopId == ACK.nodeId
6                   myNodeId == ACK.receiverID
7                   RRP.requestId == ACK.requestId
8
9           /* If true, the corresponding RRP can be discarded */
10          true: delete(RRP)
11          /* and the timer stoped */
12          stopTimer(RRP)
```

When receiving an ACK, the nodes check if its fields correlate to an RRP they have sent. If that is the case, they delete the RRP and stop the timer, as both are not needed anymore. If the timer is triggered before the ACK is received, the RRP is resent in the next management slot.

Figure 3.2 illustrates the route request phase. The nodes are identified by their nodeId. Node 10 was chosen as the MasterNode and node 1 issues the route request.

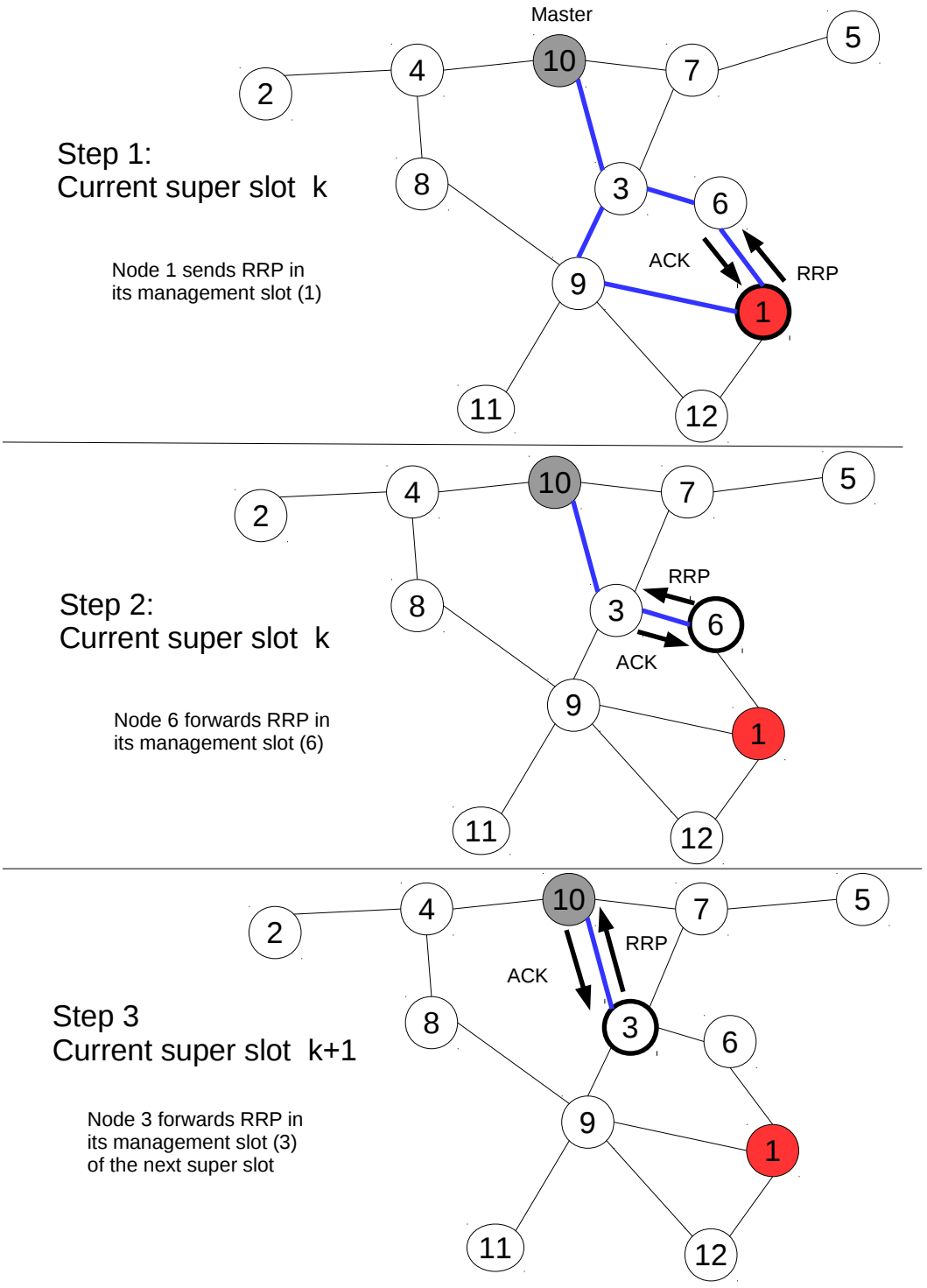


Figure 3.2: Node 1 sends a route request. It is assumed that the corresponding management slot has the same number as the node id.

Every node has reserved the management slot with the same number as its nodeId. It is also assumed that the route request is started in super slot k. Node 1 wants to join a multicast tree as a destination. It knows that node 7 is the root and that the treeId of this tree is 0.

A reason for this request could be that it wants to subscribe to a service from node 7. Since this is the first request from Node 1, the request id is 0. Node 1 determines all shortest paths (blue in the figure) and chooses the one with the lowest next hop node id. It then creates the RRP packet: **RRP(1,6,7,1,0,0)**

The RRP is scheduled and sent in the management slot of node 1 (slot 1).
Node 6 and 9 both receive the packet in slot 1. They both execute *receiveRRP* and node 9 discards the packet, because it is not the next hop on the path. Node 6, on the other hand, realizes that it is the next hop. Therefore, it sends an acknowledgement packet: **ACK(6,1,0)**.

Node 1 receives the ACK and compares it to the RRP it sent, which is then deleted.
Node 6, after determining the shortest path to the master node, updates the RRP as follows: **RRP(6,3,7,1,0,0)**.

The packet is then scheduled and sent in the management slot of node 6 (slot 6). A difference here is that the management slot of node 3 (slot 3) is before the management slot of node 6. Therefore the re-transmission of the RRP packet is delayed by nearly one super slot. This highlights one of the problems of using static management slots.

The MasterNode receives the ACK in slot 3 of super slot 2, and after acknowledging it, starts the route discovery (see Section 3.5).

3.4 Route Confirm Phase

After a route has been established by the MasterNode, all nodes that are part of the route have to be informed of the updated routes and the corresponding schedule. This is achieved by a Route Confirm Packet (RCP), which is sent by the MasterNode after route discovery and slot scheduling are finished. The RCP contains the involved nodes and the complete schedule for all of them.

The RCP needs to reach all nodes of the route, which is another multicasting problem. Possible ideas are the creation of another multicast tree to reach all nodes, which would require creating a new schedule for the RCP, or the use of the management slots to send them.

Creating another multicast tree and schedule is a costly activity and using the management slots has a potentially high delay. Also, the RCP packets would be in conflict with the RRP packets, because if a node has packets of both types to send, it has to decide which of them it will send in the current slot.

However, the RCP packet already contains the route and schedule to reach all nodes of the route from the source of the tree. Therefore, it is proposed that the RCP packet is sent to the root of the relevant tree using the management slots, and then sent along the created route, using the created schedule, to reach all other nodes in the tree. This requires that all nodes listen to the medium, even in idle slots. As minimizing energy consumption is not an objective of the algorithm, this

solution is used. Depending on the application, it may also be possible to send the RCP packet together with the first data packet to save time. The RCP packet is defined as follows: **RouteConfirmPacket (RCP):**
(nodeID, nextHopId, destinationId, treeId, mTree, schedule, requestId)

- **nodeId**: The id of the node currently sending the packet. This field will be updated by every node forwarding the packet as it is used to properly address the acknowledgments.
- **nextHopId**: The id of the next destination of this packet, which is necessary so that the receivers of the packet can determine whether they are responsible for forwarding the packet. Without that, a lot of redundant packages would be send.
- **destinationId**: The id of the destination of this packet, which is the same as the treeRootId from the routing request.
- **treeId**: The id of the multicast tree.
- **mTree**: A list of paths forming the multicast tree.
- **schedule**: The updated schedule for all nodes.
- **requestId**: The same as the requestId of the routing request.

The scheduling and acknowledging of the RCP is very similar to the route request phase and therefore will not be discussed in detail. The main difference is that the RCP has to be sent to the root of the created or expanded tree. As this information is part of the packet, this is not a problem.

Once the packet has reached the source, the created tree and updated schedule can be used to send the packet to all other nodes that are part of the relevant tree, identified by the id and the list of paths mTree.

The RCP is sent to all nodes that are part of the corresponding routes, starting in the first super slot after the root of the tree has received the RCP. This means, that the first transmission in a new or updated tree does not contain a data packet. It is not possible to send an RCP and a data packet in the same micro slot, therefore the data transmission is delayed for a super slot. The RCP is always sent to all nodes of the tree, even if a destination has been added to an already existing tree. As it is not possible to send an RCP to some of the nodes of a tree, and a data packet to other nodes of the tree at the same time, sending to all nodes in the tree does not cost more time (slots). A possible optimization would be to send the RCP only to the nodes that have to update their schedule. This would lead to a decrease of energy consumption by saving some redundant transmissions.

3.5 Route Discovery

The route discovery phase handles the creation or extension of multicast trees. The RRP always contains a request for a route between one source and one destination.

Therefore, the MasterNode has to either create a tree with one root and one destination or add a single destination to an existing tree. Also, it is distinguished between trees containing mobile nodes and trees containing only stationary nodes. Mobile nodes are only part of a route if they are the source or one of the destinations of a tree. Otherwise, the route only contains stationary nodes. Section 3.5.1 and following contain the algorithm for the stationary tree, while Section 3.5.3 contains the mobile case. The scheduling of timeslots is explained in Chapter 4.

3.5.1 Creating a New Stationary Multicast Tree

The first case that will be discussed is the creation of a new multicast tree. This situation occurs whenever the MasterNode handles a route request that does not fit into an existing multicast tree, i.e., there is no existing multicast tree with the relevant id and root. When using a SA, this is the case when a node requests the service of a service provider that currently does not have other users, or all other users are subscribed to a different service. There are two different methods for the creation of a stationary multicast tree, which result in slightly different trees.

Create a new Tree depending on the existing Schedule

The first method finds the shortest path from the source to the destination using the getShortestPaths function from Section 3.2. Usually there will be multiple shortest paths with the same length. Therefore, one of the paths has to be chosen to form the initial multicast tree.

Here, a function *getMinNumberOfFreeSlots(path p, threshold)* is defined. The function determines the number of free slots for all links on a path. If the minimum number of free slots is greater than a given *threshold*, the function returns this number as the minimum for this path. Otherwise, the path is discarded.

With the help of this function, the *createNewTree* algorithm is defined:

```
1  /* Create a new mulitcast tree mTree with one source and one
       destination */
2  createNewTree(node source, node dest):
3          /* Get an array of shortest path from the source to the
               destination */
4          paths = getShortestPath(source, dest)
5
6          /* get the minimum number of free slots for all of the paths,
               if it is greater then a threshold */
7          for(path p:paths):
8                  minFs[p] = getMinNumberOfFreeSlots(p,threshold)
9
10         /* and choose the one with the lowest number as the multicast
               tree mTree */
11         mTree = paths[minIndex(minFs)]
12         return mTree
```

The *createNewTree* function chooses the shortest path with the help of the *getMinNumberOfFreeSlots* function. The idea behind *createNewTree* is to minimize the impact that the new tree has on the rest of the network. Choosing a path with a low number of free slots – as long as there are still enough slots to find a feasible

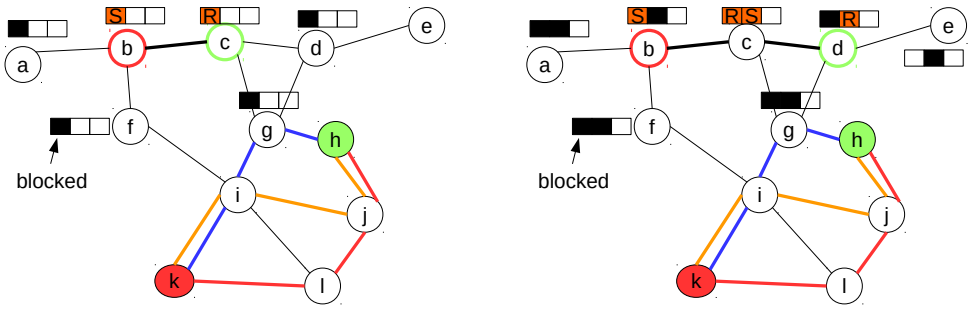


Figure 3.3: An illustration of the tree creation with *createNewTree*.

schedule – should increase the chance of finding a suitable path / schedule for later route requests. Paths with a low number of free slots are in parts of the network with already existing traffic. Existing traffic means that a lot of slots are already blocked, so adding new transmissions will block less new slots.

The result of this algorithm depends on the *threshold* for the *getMinNumberOfFreeSlots* function. A safe threshold is the number of links in the route. The maximum number of slots needed for a path is the number of links in it. If every link has this many free slots, it is guaranteed that a schedule can be found. Another possibility is to choose the maximum number of nodes in the interference range of any node on the path as the threshold. As a used slot can not be reused by nodes in the interference range of the sender (and this includes nodes that are in this path), this leads to a high chance of finding a schedule for this path with the right scheduling strategy (see Section 4.5.2.2). Choosing a lower number than this can also lead to a feasible schedule, but the chance of finding it decreases with a lower threshold.

Another possibility is to choose paths that have a high number of free slots, by exchanging *minIndex(minFs)* with *maxIndex(minFs)* in line 11. In this case, the chance of finding a feasible schedule is generally increased, but it will probably also result in more slots being blocked.

An example for the *createNewTree* method is shown in Figure 3.3. Here, it is assumed that the interference range of all nodes is the same as their communication range. In the left image, there is an already scheduled transmission from node b to node c in slot 1. This means that nodes g and f (and also nodes a and d) are blocked in slot 1. A new tree with node k as source and node h as destination will be added to the network. Three shortest paths from k to h are found. Examining the number of free slots in all nodes of those paths (assuming that no specific threshold is set) leads to the blue path as the chosen one, because it has one free slot less than the other two paths. Here, link (k, i) could be scheduled for slot 1 as the slot is free in both nodes. As slot 1 is already blocked in node f and g (both neighbors of node i) only two additional slots are blocked (in node j and l).

The downside of the function can be seen in the right figure. Adding another transmission from c to d results in node g being blocked in slots 1 and 2, in this case using the blue path means that no schedule will be found, because node g needs two slots (one for receiving and one for sending), but only has one slot available.

This is why the possibility to use a specific threshold, or preferring paths with a high number of free slots, is given. In this example, both the red and orange paths could be used and both of them have more slots available and therefore one of them would be chosen.

Create a new Tree depending on the Degree of Nodes

The previous method depends on the already existing schedule in the network, to either increase the chance of finding a feasible schedule or minimize the impact of the path for the rest of the network. Here, another method which tries to find a good tree for this particular request is proposed.

createNewTreeDegree also finds a list of shortest paths from the source of the tree to the destinations. However, it chooses a different path to form the initial multicast tree.

```
1  /* Create a new mulitcast tree mTree with one source and one
       destination */
2  createNewTreeDegree(node source, node dest):
3          /* Get an array of shortest path from the source to the
               destination */
4           paths = getShortestPath(source, dest)
5          /* Calculate the degree for all paths as the sum of the degree
               of the nodes */
6          for(path p:paths)
7               for(node n : p)
8                           degree[p] = degree[p]+numberOfNeighbors(n)
9
10         /* Choose the path with highest degree as the multicast tree
               mTree */
11         mTree = paths[maxIndex(degree)]
12         return mTree
```

For all paths, the degree of all nodes in the path is calculated and summed up, to get a total degree for every path. Since a node has a high degree when it has a large number of neighbors, it follows that a path with a high degree goes through a dense part of the network (i.e., many nodes are in each other's communication range). When adding further destinations to the tree, this should lead – in general – to shorter paths, because the chance of the new destination being near a node already part of the tree is higher, compared to paths with a lower degree.

Figure 3.4 shows an example of the tree creation depending on the degree of paths. In this example a new tree with node k as source and node h as destination is created. The shortest path search returns three different paths, shown in blue, red and orange. The degrees of the paths are as follows:

- blue: $2 + 5 + 4 = 11$
- orange: $2 + 5 + 3 = 10$
- red: $2 + 3 + 3 = 8$

Therefore, the blue path is chosen as the initial multicast tree. It can be seen that this can lead to an improvement, compared to the other paths, as soon as new

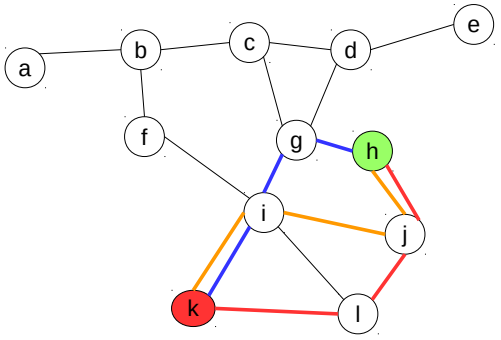


Figure 3.4: An illustration of the tree creation with *createNewTreeDegree*.

destinations are added. The majority of the nodes are closer to the blue path than to the other paths. Obviously, this will not always be the case.

It is possible to construct examples where this method will lead to a tree with more links than a method that chooses the path differently, as soon as more destinations are added to the tree. Assuming that every node issues the same number of route requests, it will usually lead to a tree with a lower number of hops, because, on average, new destinations will be closer to the chosen path.

This method also has disadvantages. When taking the slot reservations into account, a node with many neighbors will also block many other nodes when transmitting. Technically, a node with a lower number of neighbors can have a higher number of interference neighbors than a node with a higher number of communication neighbors, but that should rarely be the case.

3.5.2 Adding Destinations to Existing Trees

Another needed functionality is adding a destination to an already built multicast tree. When adding a destination to the tree, the first goal is to minimize the overall number of links in the tree. Therefore, the shortest paths from the new destination to all other nodes of the tree are determined. After that, the shortest possible length is determined and all longer paths are discarded. The result is a list of paths that all have the same (shortest possible) length. It is possible, although unlikely, that they all end on the same node. More often there are several nodes that can be reached with the same number of hops.

The second goal is to minimize the potential delay of sending from the source to all destinations. The delay (when measured in the number of hops) is at least as long as the length of the route from the source to the destination. To achieve this, the algorithm chooses the path that has the shortest route from the source to the destination from the determined list of shortest paths.

```
1 /* Adds a destination dest to an existing tree mTree */
2 addDestToTree(tree mTree , node dest)
3         /* Go through all nodes of the tree */
4         for( node x : mTree) {
```

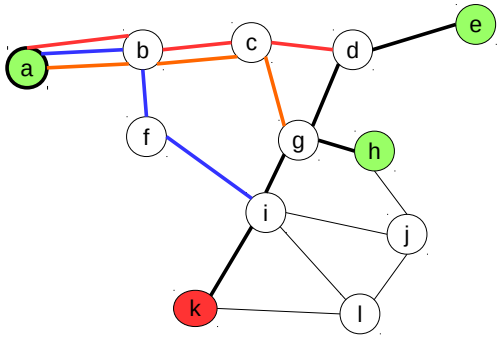


Figure 3.5: An example of adding a new destination to an existing multicast tree

```
5                   /* and get an array of all shortest paths from this
                        node to the destination */
6                   paths = getShortestPath(x,dest)
7                   /*save the length and the paths */
8                   tempLength[x] = |paths[0]|
9                   shortestPath[x] = paths
10          }
11          /* Get the minimum possible length to reach the tree from the
                destination */
12          minLength = minValue(tempLength)
13
14          /* and go through the tree again and save all paths with the
                minimum length*/
15          for(node x:mTree):{
16                  if(tempLength[x] == minLength):
17                          finalPaths[x] += shortestPath[x]
18          }
19          /* finally, find the path that has the minimum length and the
                shortest distance to the source */
20          minLength = maximumNetworkDiameter
21          for(node x:mTree):
22                  for(path p:finalPaths[x]):
23                          length = getDistane(x,mTree.source,p) //number
                                of hops from x to source using path p
24                          if(length < minLength):
25                                  length = minLength
26                                  result = p
27          /* result contains the chosen path and has to be added to the
                tree */
28          mTree += result
```

In Figure 3.5, a new destination a is added to an already existing multicast tree with source k and destinations h and e. The first step is to determine the shortest path from the new destination a to all nodes already part of the tree. The picture only shows the paths to the nodes i, g and d, but in in the initial step the paths to the other nodes k, h and e are also included. The next step is to identify the length of the shortest possible paths to one of the nodes, in this case three. All paths longer then three are now discarded. The remaining paths are displayed in blue, red and

orange. In this example, there is each one shortest path from a to d, g or i. This is usually not the case.

The next step is to choose which path will be added to the multicast tree. Therefore, the number of hops from the new destination a to the source using the examined path has to be calculated. In this case the resulting paths and distances are as follows:

- blue: (a, b, f, i, k) with the distance 4
- orange: (a, b, c, g, i, k) with the distance 5
- red: (a, b, c, d, g, i, k) with the distance 6

Obviously, the blue path will be chosen, since it has the smallest distance from the destination a to the source k. If there are still several paths left at this point, the same heuristics as in the *createNewTree* and *createNewTreeDegree* functions can be used to choose one of them.

3.5.3 Creating a Mobile Multicast tree

The route discovery is different when sending from or to mobile nodes. The mobile nodes move and therefore it is not possible to include them in a normal multicast tree. As soon as a mobile node moves out of the range of the nodes currently connecting it to the tree, it would be necessary to create a new tree or at least to repair the current tree. To solve this problem, a new type of nodes is introduced.

AccessNodes

Each mobile nodes is assigned a number of AccessNodes, which form gateways from the stationary network to the mobile nodes. The AccessNodes have to forward the data from stationary nodes to the mobile node, or from the mobile node to the stationary nodes. This means, that at least one of the AccessNodes has to be in communication range of the mobile node at all times. If several AccessNodes are currently in the range of the mobile node, one of them has to be chosen to act as the currently active AccessNode.

A number of AccessNodes has to be chosen, so that at least one of them is in range of the mobile node at all times. One possibility is to choose the minimal number of nodes which is necessary to cover the whole movement area of the mobile node with their combined communication range. A problem with this strategy is that the communication quality between the mobile node and the AccessNode can change depending on the movement of the mobile node, for example due to obstacles. Therefore, it is hard to define how many / which nodes are needed when only looking at the theoretical communication ranges.

Another way to choose the AccessNodes could be measuring the link quality from the mobile node to all stationary nodes, while the mobile node moves around, and then choose enough AccessNodes, so that the mobile node always has a stable link to at least one of them. This would be done during the preparation, not while the network is actually in use.

Another important point is that the AccessNodes as well as the mobile node have to know which AccessNode is currently in range of the mobile node and if more than one is in range, which one is currently acting as the gateway to the mobile node (this node will be called active).

Therefore, the mobile node and the AccessNodes exchange periodic beacon frames. The mobile node can use the beacon frames to determine which nodes are in its range and which currently has the best link quality, and then choose that node. A measurement of the link quality is the received signal strength, which is a good indicator for the resulting packet reception rate [29]. The currently active AccessNode has to be informed.

For future work, the geographical position and movement speed of the mobile node could be used to predict which nodes are active at which time, in order to save traffic and therefore resources.

As mentioned in Section 2.4.1, it is assumed that the AccessNodes are predetermined, and are distributed such that at least one of them is in range of the mobile node at all times.

3.5.3.1 Route Discovery with a Mobile Node as Destination

The algorithm to create a multicast tree with mobile nodes is further split into two cases. In the first case the mobile node acts as a destination of the multicast tree. As explained earlier, the mobile node is always in range of one of the AccessNodes, but the root of the multicast tree can not predict which of the AccessNodes is currently active. Therefore, the data is always sent to all AccessNodes at once and only the last hop, from the AccessNode to the mobile node, changes depending on its position. Therefore it is sufficient to create a multicast tree with all AccessNodes as destinations. The last hop (from the AccessNodes to the mobile node) will be handled by the scheduling algorithm in Chapter 4.

The only difference to the stationary routing is, that now a tree with several destinations at once has to be created. First, a tree with one of the AcessNodes is created and then the other AccessNodes are added as destinations. To do this the *createNewTree* and *createNewTreeDegree* functions from Section 3.5.1, as well as the *addDestToTree* function from Section 3.5.2 are used.

At this point, it has to be decided which AccessNode is used to create the initial tree and in which order the rest of the AccessNodes are added to the tree. It was decided that the AccessNode with the shortest distance to the source is added first. The rest of the AccessNodes are ordered by their distance to the source and added consecutively. When two AccessNodes have the same distance, the one with the lower id is added first. The algorithm that creates a tree with the mobile node as the destination is shown below:

```
1 /* Creates a multicast tree with the mobile node as the destination */
2 createMobileTreeDest(node source, node mobile):
3         /* calculate the distance from the source to all AccessNodes */
4         for(node a:AccessNodes):
5                 distance[a] = |getShortestPath(a,source)[0]|
6         /* choose the node with the shortest distance */
7         shortestDistanceNode = minIndex(distance)
8         /* and use it to create a new tree */
```

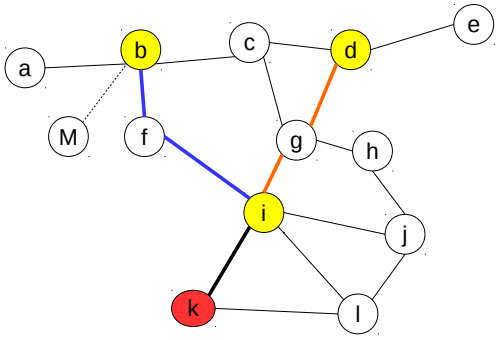


Figure 3.6: Creating a multicast tree with a mobile node M as destination.

```
9           mTree = createNewTree(shortestDistanceNode, mobile)
10          /* remove this node from the distance array */
11          distance.remove(shortestDistanceNode)
12          /* iterate to the rest of the nodes */
13          while (|distance| > 0)
14                  /* choose the node with shortest distance from the
                        remaining nodes */
15                  shortestDistanceNode = minIndex(distance)
16                  /* add it to the tree */
17                  addDestToTree(mTree, shortestDistanceNode)
18                  /* and remove it from the distance array */
19                  distance.remove(shortestDistanceNode)
```

Figure 3.6 shows an example of a multicast tree with the mobile node M as destination. Here, the nodes b, d and i act as AccessNodes. The source of the tree is k. Node i is closest to the source, and is used to create the initial tree (shown in black). The other two nodes are both two hops away from the tree, therefore node b is added first, followed by node d. The paths are chosen by *createNewTree* and *addDestToTree* functions. The mobile node M is currently in range of node b.

Adding a Mobile Node as Destination to a Tree

Adding a mobile node as destination to an already existing tree, which can be a stationary or mobile multicast tree, is done in the same way. Instead of creating the new tree with one of the AccessNodes, all AccessNodes are added to the existing tree with the function *addDestToTree* in the order of their distance to the root of the tree.

3.5.3.2 Route Discovery with a Mobile Node as Source

For trees with a mobile node as source the route discovery is different. Again, the AccessNodes are used as gateways from the mobile node to the stationary network. A problem with this is that it would require multiple trees, one tree from every AccessNode to all destinations. To simplify the matter, a new node type is introduced.

DistributorNode

The DistributorNode is a new node type, that acts as a forwarder from the AccessNodes to the rest of the network. The AccessNodes send their data packets to the DistributorNode, which then forwards them to the destinations. This means that there is one multicast and one concast[1] tree in this case (One mulitcast tree from the DistributorNode to all destinations and one concast tree from all AccessNodes to the distributor node). The tree from the DistributorNode to the destinations is a stationary multicast tree and will not be discussed in detail.

However, the concast tree from the AccessNodes to the DistributorNode is created differently. The main difference is that only one branch is used at any time, since only one AccessNode is active at any time. This means that the only important property of this tree is to have the shortest possible paths from the AccessNodes to the Distributor Node. Minimizing the total number of links in the tree is not necessary.

There will also be a different slot scheduling algorithm for this tree, because nodes in the different branches cannot interfere with each other, as they will never be active at the same time.

Choosing the right DistributorNode is also important. The number of hops in the resulting trees, as well as the delay, will depend on the chosen node. There are several strategies for choosing the DistributorNode. One strategy is to choose the AccessNode which is currently active at the time the routing request is issued. This will most likely result in a good tree as long as the mobile node stays in range of the current AccessNode. As long as it does, the resulting tree is not different from a completely stationary tree with the AccessNode as the source. However, as soon as the mobile node moves out of range of the current AccessNode, another path will become active, and then the resulting tree will not be optimal.

Another strategy would be to choose a DistributorNode which is always good. A good DistributorNode is a node which allows the creation of routes, which are optimized according to the assessment criteria (see Section 3.1). The chosen node will be responsible for forwarding the data to potentially every node in the network. Assuming that all nodes in the network are potential destinations, choosing a node which is close to as many nodes as possible leads to trees with a low number of links. Therefore, the average distance to all other nodes is calculated and the one with the shortest average distance is chosen to be the DistributorNode as shown in the following listing:

```
1 /* Determines the node with the lowest average distance to all other
      nodes */
2 getDistributorNode():
3         /* iterate over all nodes in the network */
4         for(node a: network G)
5                 mean = 0
6                 /* sum up the distance from this node to all other
                      nodes */
7                 for(node b: network G)
8                         mean +=  getDistance(a,b)
9                 /* and calculate the average distance */
```

[1] a tree from several sources to one destination

```
10                  mean = mean / numberOfNodes
11                  meanDistance[a] = mean
12          /* return the node with the smallest average distance */
13          return minIndex[meanDistance]
```

For future work, it is possible to optimize this selection if there is more information available about the services of a mobile node. This information can be taken into account and the nodes could have different weights during the average distance calculation depending on how often they need the service of the mobile node.

For the rest of the thesis, it is assumed that the DistributorNode is chosen depending on the unweighted average distance to all other nodes, unless specified otherwise.

After the DistributorNode is determined, the route discovery for the tree with the mobile node as the root can be executed. As mentioned in the beginning of this section, two trees are needed here. The first is the tree from the AccessNodes to the DistributorNode. It was already explained that the only important property of the tree from the AccessNodes to the DistributorNode is the length of the individual paths. This is exactly what the function *createNewTree* does. It finds all shortest paths from one source (here an AccessNode) to a destination (here DistributorNode) and chooses one of them to form the initial tree. For this tree, it is not useful to choose a path with a high degree, because all destinations are connected to the DistributorNode and not to other nodes that are part of this tree. So, choosing a path with a higher degree has only disadvantages (higher interference).

Besides the tree from the AccessNodes to the DistributorNode, the tree from the DistributorNode to the destinations has to be created. This is a stationary tree that can be created with the usual functions (*createNewTree* and *createNewTreeDegree*). The complete function for this case is shown in the following listing.

```
1  /* Creates a mobile tree with a mobile node as the root, a stationary
       node as the destination with the help of a DistributorNode */
2  createMobileTreeRoot(node mobile,node dest, node distri):
3          /* the resulting tree is a combination of other trees */
4          tree result
5          /* iterate over all AccessNodes */
6          for(node a: AccessNodes):
7                  /* and create a new tree from this node to the
                       DistributorNode */
8                  tree mTree = createNewTree(a,distri)
9                  /* add this tree to the resulting tree */
10                 result+=mTree
11         /* after the tree from the AccessNodes to the DistributorNode
              has been created, the tree from the DistributorNode to the
              destinations has to be created */
12         createNewTree(distri,dest) //can also use createNewTreeDegree
```

Figure 3.7 shows an example of a mobile tree with the mobile node as source. The nodes b, d and f are AccessNodes. The DistributorNode was chosen with *getDistributorNode*. Node g has an average distance of 1.58 hops to other nodes, which is the lowest in this example. The tree from the AccessNodes to the DistributorNode is pictured in black.

It can be seen, that nodes b and f are each connected to the DistributorNode g over a two hop path, even though they are in one hop distance to each other.

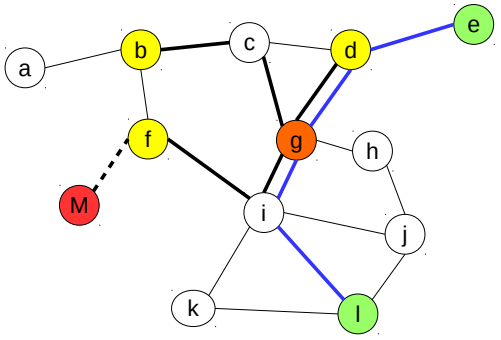


Figure 3.7: An example of tree with the mobile node M as the source, the node g as the DistributorNode, the nodes b, d and f as the AccessNodes and the nodes l and e as the destinations.

So the total number of hops in this (black) tree is not optimized, because by, e.g., connecting f to b the tree would have one link less, but than the length of the path from f to g would be three instead of two. As explained at the beginning of this section, this would be inefficient for the tree from the AccessNodes to the DistributorNode.

The figure also shows the tree from the DistributorNode to the destinations in blue. It shows that this leads to redundant transmissions in some cases. If node f is the currently active AccessNode, the data is sent from i to g and then again from g to i. Unfortunately, the paths have to be reserved anyway, because they are needed when node b or e is active. A similar situation occurs when node e is active. As the reservations do not change when the active AccessNode changes, the redundant transmissions are sent anyway. A small advantage is, that if one of the destinations receives the message while it is traveling towards the DistributorNode, it was received after a shorter delay, but that only happens when there are no other transmissions (e.g., from other trees) that interfere with receiving the message.

Chapter 4

Centralized Timeslot Assignments

Besides the route discovery (see Section 3.5), the second important part of the routing algorithm is the scheduling of timeslots. In order to be able to transmit in a conflict-free manner in a TDMA-based network, transmissions and receptions of messages have to be scheduled into timeslots. In Section 2.2.2, Property 2 was introduced, which states the rules a schedule has to fulfill in order to be conflict-free. The property states that a slot can be scheduled for a transmission if it is free in the sender and all receivers, as well as in all nodes in their interference range.

It was also mentioned in Section 2.2.3 that the scheduling problem is NP-hard, which means that scheduling heuristics are needed, which will be introduced in this chapter. The scheduling algorithm is always executed after a path has been added to a tree.

The slot schedule is created by the MasterNode of the network. The main advantage of a centralized approach for the scheduling is the global knowledge of the MasterNode, regarding both routes and schedules. The main downside is that the schedule has to be sent to all relevant nodes as soon as it is updated, which creates management overhead. On the other hand, a decentralized version would also create overhead, because the nodes would need to exchange their schedules with all nodes in their interference range, in order to create their own schedules.

Before presenting the algorithm itself in Section 4.5, several concepts and definitions will be discussed (Sections 4.1 to 4.4).

4.1 Slot Types

In Section 2.2 it was explained that there are several important slot types. The actual transmissions happen in micro slots of a fixed length. These slots are the ones relevant for the scheduling algorithm. Micro slots are organized into super slots, which contain a fixed number of micro slots. The super slots are repeating, i.e. if a micro slot is reserved in one of them, it will be reserved in all following super slots until canceled. Furthermore, there is a difference between management and dynamic slots.

Management Slots

As the name suggests, management slots are used for management traffic, such as the RRPs and RCPs. The management slots are static and predetermined for all nodes, i.e. the management slots are distributed once and will not change during the run time of a network. Every node has at least one management slot in every super slot in which it has exclusive sending rights. This means that for management traffic, no SDMA is used.

According to the assumptions (in Section 2.2), the exact distribution of the management slots is left open. The scheduling for the RRPs and RCPs was already discussed in the relevant sections (Sections 3.3 and 3.4) and the scheduling of the beacon frames from the mobile nodes is explained in Section 4.3. Besides that, this chapter will focus on the scheduling of the dynamic slots for the routes found in the route discovery phase.

Dynamic Slots with SDMA

The second group of slots are the dynamic slots, which are used for the general traffic of the network, i.e., all traffic on the routes found in the route discovery phase. Here, SDMA is used to increase the throughput of the network. A slot can be reserved by a node to send or receive a data packet.

All messages in this thesis are acknowledged, i.e. both the sender and the receiver of a data packet send in the reserved slot. In the following, the ACKs will not be explicitly mentioned in the schedule.

For a better understanding of the scheduling algorithm, slots will also be blocked from sending and receiving, even though it is not necessary for the functionality of the algorithm. Due to the acknowledgement mechanism, it is not necessary to distinguish between slots being blocked from sending or blocked from receiving (see Property 1 and 2 in Sections 2.2.1 and 2.2.2). The dynamic slots can be addressed by their slot number, starting at zero. The total number of dynamic micro slots in every super slot is known (**numberOfSlots**).

4.2 Local Multicast

It was already mentioned in the previous chapter that the algorithm uses local multicasting to send to up to three receivers in a micro slot to increase the throughput of the network. One of the problems of this is that the sender needs to receive ACKs from all receivers. The ACKs have to be scheduled at different times, so that they do not interfere with each other.

All nodes, that are part of a multicast tree, know the complete schedule for this tree as soon as they receive the RCP packet. Therefore, they also know in which nodes and slots local multicasting is used. With this information it is possible to schedule the ACKs at different times in the same micro slot. If the times for the sending of the first, second and third ACK are fixed (within the same micro slot), this results in conflict-free transmissions of all acknowledgments. The ACKs could be ordered by the nodeIds of the receivers, so if two nodes with the ids 2 and 4 know

that they both receive a packet from the same node in the same slot, node 2 would send its ACK first.

According to the design of the routing algorithm, local multicasting can only occur at specific nodes. If a path creates a new tree, no local multicast can occur. If a path is attached to an already existing tree, the first node of the path is always part of that tree. All other nodes of that path cannot be part of the existing tree, because if this was the case, there would have been a shorter path from the current destination to the already existing tree, which would have been chosen in the route discovery phase. Therefore, local multicasting can only occur in the first node of a path added to an already existing tree.

4.3 Mobile Nodes

Due to the movement of a mobile node, the nodes that are in its communication and interference range also change over time. If it is assumed, that a node can move through the whole network, transmissions of the mobile node can potentially interfere with all other nodes. Also, transmissions of all others nodes can potentially interfere with receptions of the mobile node. If a mobile node is confined to a certain area, it may be possible to allow some nodes to transmit together with the mobile node as long as those nodes are never in its interference range and the mobile node is never in the interference range of those nodes. In the rest of the thesis, it is assumed that mobiles nodes can potentially interfere with the communication of all other nodes, therefore all slots used for communication with a mobile node have to be blocked in the whole rest of the network, i.e., no SDMA can be used in those slots.

For future work, it may be interesting to find a way to use SDMA together with the mobile node. As the geographical position of the mobile is known, it may be possible to create a dynamic schedule that adapts to the current position of the mobile node to allow SDMA in those slots for an increased performance.
Section 3.5.3 explained how a mobile node communicates with the rest of the network through AccessNodes and that it needs to know which AccessNode is currently active, which is determined by periodic beacon frames of the mobile node. These beacon frames can be scheduled in the mobile node's management slot, as long as it is assumed that it does not interfere with the sending or receiving of other management packets, such as route requests. If it is assumed that route requests are relatively rare, the beacon frames can be skipped if a route request is issued. If multiple AccessNodes receive the requests and all of them forward it, the MasterNode will be able to detect the duplicate RRP, because the field *requestId* is the same in all of them. Also, mobile nodes move at a relatively slow speed (see Section 2.4.2), so switching from one active AccessNodes to another does not occur often, i.e., skipping a few beacon frames should usually not lead to problems.
Another possibility is to reserve another management slot for the beacon frames.

4.4 Assessment Criteria for the Schedule

Before introducing the scheduling algorithm, its objectives have to be discussed. The question that has to be answered is what defines a good schedule. The answer to this question depends on the application context. A universal solution that applies to all problems does not exist. One important feature is a low **delay**. The delay is defined as the time (in slots) it takes to send a packet from the source to the intended destination, or – in the case of multicast routing – the time until all destinations have received the message. Here, it is distinguished between the delay of path and the delay of a tree (if it has multiple destinations).

A lower bound for the delay of a path is the number of links in it. The worst case scenario is that every hop is delayed by nearly a whole super slot. This happens when a link is scheduled in an earlier slot than its preceding link.

For example, if there are 10 dynamic slots and in a path $p = \langle a, b, c, d \rangle$ the slots are assigned as follows:
Link $[a, b]$ is assigned slot 2, link $[b, c]$ is assigned slot 8 and link $[c, d]$ is assigned slot 4. In this case, the transmission starts in slot 2 of the first super slot and ends in slot 4 of the next super slot. Since every super slot contains 10 dynamic micro slots, this leads to a delay of 12 dynamic slots. Here, and in future examples, the management slots, which would further increase the delay are ignored. As they are fixed and therefore add the same delay in all cases, ignoring them is not a problem for the evaluation of a schedule.
The delay of a tree is defined as the time it takes to reach all destinations.

Another useful criterion is the **utilization** of the slots, which is comparable to the cost of a route, explained in Section 3.1.3. Here, the utilization is defined as the relation between slots used for sending and receiving and blocked slots. A schedule with a high utilization has less blocked slots compared to a schedule with a low utilization. This criterion is useful in networks with a lot of traffic. The more slots are blocked, the less likely it is to find a schedule for future routes.

The utilization is calculated for the whole schedule as the number of transmissions plus the number of receptions divided by the number of blocked slots. Slots that are blocked by multiple transmissions / receptions are only counted once, i.e., choosing slots that are already blocked in a large number of interference neighbors leads to a higher utilization. If the utilization is high, a low number of slots are blocked, while a low utilization means that a relatively large number of slots are blocked.

4.5 Timeslot Assignments for Trees

In the following, the scheduling algorithm will be explained. In Section 4.5.1, notations used by the algorithm will be introduced. Section 4.5.2 explains the basic algorithm, used for paths that created a new stationary tree. Finally, Section 4.5.3 discusses the necessary additions to the algorithm needed to handle other scenarios, such as adding destinations to other trees (local multicast) and handling mobile nodes.

4.5.1 Definitions

The following notations will be used by the scheduling algorithm:

- TX_a is a list of all slots in which node a is scheduled to send
- RX_a is a list of all slots in which node a is scheduled to receive
- NX_a is a list of all slots in which node a neither sends nor receives
  - $NX_a = \{x | x \notin RX_a \text{ and } x \notin TX_a\}$
- $NX_{[a,b]}$ is a list of all slots where a and b (link from a to b) neither send nor receive
  - $NX_{[a,b]} = NX_a \cap NX_b$

A slot can be scheduled:

- $\text{SEND}_{a,b,c}$ to send to up to three nodes a,b,c
- RECEIVE_a to receive from node a
- BLOCKED to be blocked from sending and receiving. Explicitly marking a slot as BLOCKED is not necessary for the algorithm, however it is useful to visualize the results of the scheduling
- the function *setSlot(node a, slot i, STATUS)* sets the slot i of node a to the specified STATUS. If a node is already scheduled to send in this slot, the new receiver is added instead of overridden.

Property 2 can be reformulated to:

- $IFS_a = NX_a \cap NX_b\big|_{b \in I_a}$
  - IFS_a is a collection of all interference free slots of node a. This means that node a can be scheduled to send or receive in all of these slots.
- $IFS_{(a,b)} = IFS_a \cap IFS_b$
  - $IFS_{(a,b)}$ is a list of all interference free slots of a link (a, b). This means that the link can be used to send / receive in all of those slots.

4.5.2 Scheduling Algorithm – Creating a New Tree

To explain the principles of the scheduling algorithm as clearly as possible, the algorithm for the most basic scenario is introduced first. The scenario is that a new stationary tree has been created and needs to be scheduled.
There are two different scheduling algorithms, which fulfill different objectives.

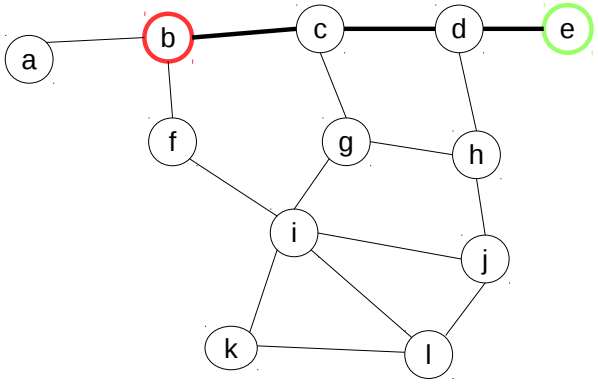
4.5.2.1 Minimizing the Delay

The first algorithm tries to minimize the delay of the added path. The minimal delay of a path is equal to its number of links. This is achieved whenever successive links reserve consecutive slots. E.g, if for a path $p = \langle a, b, c, d \rangle$ the slots 2, 3 and 4 are reserved for sending from a, b and c, the resulting delay is 3 micro slots, which is equal to the lower bound. So, to minimize the delay of a path, the procedure tries to reserve slots for a link that are as close as possible to the slot used for the preceding link. The slot for the first link of a path depends on the context. For now, it is assumed that *getStartingSlot()* returns 0, so that the scheduling starts in the first slot. The *getStartingSlot()* function will be further discussed in the following sections.

```
1  /* Assigns slots to a path p, minimizing the delay */
2  assignSlotsDelay(path p):
3          /* The scheduling starts in a slot depending on the situation
               */
4          currentSlot = getStartingSlot() // for now assumed to be zero
5          /* go through all links in the path */
6          for(l=(a,b) ∈ p) {
7                  /* iterate over all slots and check if the currentSlot
                       is free in this link */
8                  i = 0
9                  while(i<numberOfSlots ∧ currentSlot ∉ IFS_[a,b]) {
10                         /* if the current slot is not free, try the
                               next slot */
11                                 if(currentSlot < numberOfSlots) {
12                                         currentSlot = currentSlot +1
13                                 }
14                                 else {
15                                         currentSlot = 0
16                                 }
17                                 i++
18                 }
19                 /* If the while loop is finished, either the
                       currentSlot is free or all slots where checked */
20                 if(currentSlot ∈ IFS_[a,b]) {
21                         /*reserve slots for sending and receiving */
22                         setSlot(a, currentSlot, SEND_b)
23                         setSlot(b, currentSlot, RECEIVE_a)
24                         /* and block the slots in all nodes in the
                               interference range of the sender and the
                               receiver */
25                         for( node x ∈ I_a ∪ I_b)
26                                 setSlot(x, currentSlot , BLOCKED)
27                         /* next link => start scheduling in the next
                               slot */
28                         currentSlot++
29                         /*continue with the next link */
30                 }
31                 /* if the currentSlot is not free, no slot was free */
32                 else{
33                         return(SlotAllocationFailed)
34                 }
35         }
```

```
36          return(ScheduleFound)
```

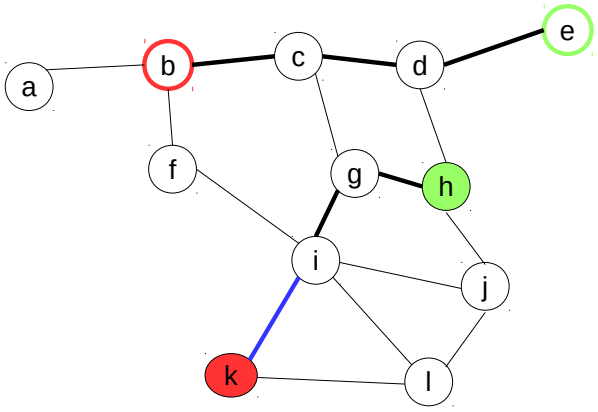
The algorithm starts in the first link of the path and the first slot (slot 0) as the current slot. For every link, it checks if the current slot is free. If the current slot is free, the algorithm assigns it to the link, i.e., the slot is set to $\text{SEND}_{receiver}$ in the sender and to RECEIVE_{sender} in the receiver. Additionally, the slot is set to BLOCKED in all nodes in the interference range of the sender and receiver. If that is not the case, the next slot is checked. If the end of the slots is reached, it starts again at the first slot, until all slots are checked. If an interference-free slot is found for a link, the algorithm continues in the next link of the path. Here, it checks if the next slot (the one after the slot that was assigned to the previous link) is interference-free first and then continues as described above. If for any link, no free slots are found, the algorithm terminates unsuccessfully.



Slot:	0	1	2	3	4
node a:	x				
node b:	S_c	x			
node c:	R_b	S_d		x	
node d:	x	R_c		S_e	
node e:		x		R_d	
node f:	x				
node g:	x	x			
node h:		x		x	
node i:					
node j:					
node k:					
node l:					

Figure 4.1: An example scheduling problem with an already scheduled path from b to e.

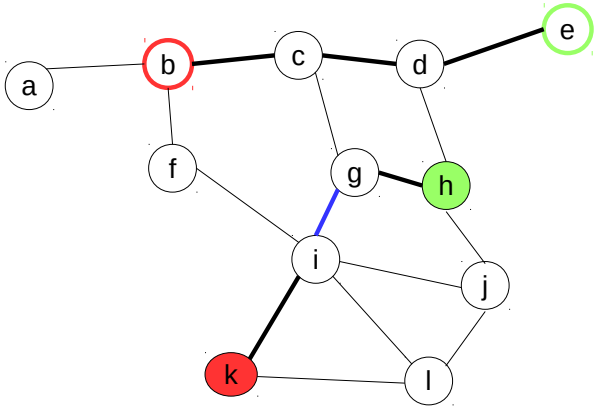
Figure 4.1 shows the starting point of an example scheduling problem. In this example, it is assumed that the interference range of all nodes is the same as their communication range. The figure shows a network with an already scheduled path from b to e and an example schedule for this path. Normally, link $[d, e]$ would have been scheduled in slot 2 instead of three, but it is possible that this slot was blocked (e.g., by another tree) at the time that the schedule for this path was created.
Now, another route $p = \langle k, i, g, h \rangle$ was discovered and the transmissions for this route have to be scheduled. The scheduling algorithm starts with the first link $[k, i]$ of the path p and checks if the first slot (slot 0) is interference-free in this link. As slot 0 is free in both nodes, the transmission is scheduled for this slot and blocked in all interference neighbors of k and i (nodes f, g, j and l). The updated schedule is displayed in Figure 4.2.



Slot:	0	1	2	3	4
node a:	x				
node b:	S_c	x			
node c:	R_b	S_d		x	
node d:	x	R_c		S_e	
node e:		x		R_d	
node f:	x				
node g:	x	x			
node h:		x		x	
node i:	R_k				
node j:	x				
node k:	S_i				
node l:	x				

Figure 4.2: Continued scheduling example.

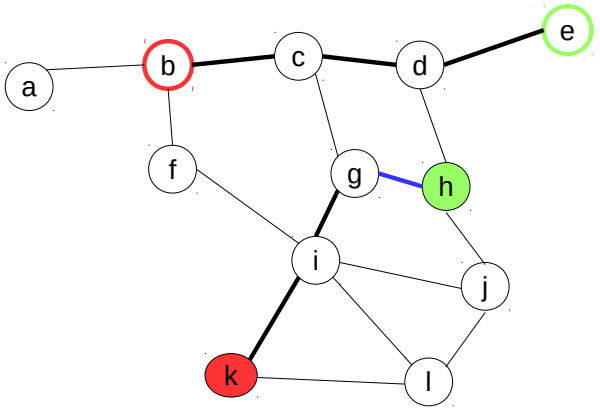
Next, the link $[i, g]$ is scheduled. Here, the algorithm starts to search the free slot in the current slot set in the last step (slot 1). As slot 1 is blocked in node g, it is not possible to use it. Therefore slot 2 is tested, which is still free in both nodes. So, slot 2 is chosen for link $[i, g]$. Node i is scheduled to send to node g and g is scheduled to receive from i. All nodes in the interference range of i and g are blocked, and the current slot is set 3. Figure 4.3 shows the resulting schedule.



Slot:	0	1	2	3	4
node a:	x				
node b:	S_c	x			
node c:	R_b	S_d	x	x	
node d:	x	R_c		S_e	
node e:		x		R_d	
node f:	x		x		
node g:	x	x	R_i		
node h:		x	x	x	
node i:	R_k		S_g		
node j:	x		x		
node k:	S_i		x		
node l:	x		x		

Figure 4.3: Continued scheduling example.

In the next step, the last link $[g, h]$ is scheduled in the same way as the previous links, since slot 3 is blocked in h, slot 4 is used. The final schedule is shown in Figure 4.4.



Slot:	0	1	2	3	4
node a:	x				
node b:	S_c	x			
node c:	R_b	S_d	x	x	x
node d:	x	R_c		S_e	x
node e:		x		R_d	
node f:	x		x		
node g:	x	x	R_i		S_h
node h:		x	x	x	R_i
node i:	R_k		S_g		x
node j:	x		x		x
node k:	S_i		x		
node l:	x		x		

Figure 4.4: Continued scheduling example.

The transmission from the source k to the destination h starts in slot 0 and ends in slot 4. Therefore, the delay of this path is 5 micro slots. The complete schedule contains 22 blocked slots and 6 transmissions and receptions, therefore the utilization of this schedule is 0.54.

4.5.2.2 Maximizing the Utilization

The second version of the scheduling algorithm tries to maximize the utilization of the schedule as well as increase the chance of finding a schedule for the given and future routes. To achieve this, 3 Slot Decision Policies (SDPs) are defined, similar to [28]. The SDPs are slighty different than in the paper, because the MasterNode has global knowledge of the routes and schedules.

The first policy is to start the scheduling in a link that has the smallest number of interference-free slots available to increase the chance of finding a schedule for this path. When scheduling a link that is part of a route, the chosen slot cannot be used in other links of this route, which are in the interference range of the nodes of the current link. Depending on the size of the interference range and the length of the route, most slots can only be used a few times on a route. This way, scheduling links with a small number of free slots first, increases the chance of finding a feasible schedule. E.g., a path $p = \langle a, b, c, d \rangle$ has to be scheduled and the link $[a, b]$ has slot 1, 2 and 3, link $[b, c]$ has the slots 2 and 3 and link $[c, d]$ has only slot 1 free, then if slot 1 is scheduled for link $[a, b]$, there are no free slots left for link $[c, d]$. Scheduling the link with the smallest number of free slots first, in this case link $[c, d]$, means that all links in this example can be scheduled. The following listing shows the *getCurrentLink* function which chooses the link according to this policy.

```
1 /* Returns the link with the smallest number of interference-free slots
      */
2 getCurrentLink(path p):
3         /* Count the number of interference-free slots for all links */
4         for(link l= [x,y] ∈ p) {
5                 if(!alreadyScheduled[l] {
```

```
6                       freeSlots[l] = |IFS_[x,y]|
7                   }
8           }
9           /* and choose the one with the lowest number of free slots
                first */
10          return  minIndex(freeSlots)
```

getCurrentLink counts the number of interference-free slots of all links that have not been scheduled yet and returns the one with the smallest number of free slots.
After a link has been selected, a slot for this link has to be chosen. Here, the first heuristic is a least conflict first policy. If several slots are free, slots that have the least conflict with other, not already scheduled, links of the route are preferred. E.g., in an example path $d = \langle a, b, c, d \rangle$, link $[a, b]$ has slot 1 and 3 and link $[b, c]$ and link $[c, d]$ both have slots 1 and 2 free. As all links have the same number of free slots, the scheduling starts in link $[a, b]$. If slot 1 is scheduled for link $[a, b]$, it can not be reused for the other links. Since both other links have only one other free slot (2), no feasible schedule can be found. Applying the least conflict first policy, slot 3 is chosen for $[a, b]$, because slot 1 is free in two other links of the route, while slot 3 is not free in any other links. This way, a feasible schedule can be found. The following listing shows the *getLeastConflictSlots* function that applies this policy.

```
1  /* Returns the free slots of the currentLink, that have the least
       conflict with the rest of the route */
2  getLeastConflictSlots(path p, link currentLink = [a,b]) {
3           /* iterate over all interference-free slots of the currentLink
                */
4           for( slot i ∈ IFS_[a,b]) {
5                   /* and count how often it is free in all other, not
                        already scheduled, links of the route */
6                   for( link l=[x,y] :p ){
7                           if(!alreadyScheduled[l] ∧ i ∈ IFS_[x,y]) {
8                                   count[i]++
9                           }
10                  }
11          }
12          /* return all slots that have minimal conflict */
13          /* it is possible that several slots are free in the same
                number of links */
14          return  allMinIndex(count)
15 }
```

getLeastConflictSlots counts how often the free slots of the current link are free in other, not scheduled, links of the route and returns the slot that is free the least number of times in them. If there are still multiple slots available, it returns all of them.

The last heuristic tries to minimize the utilization of the schedule. If a slot is already blocked in a large number of nodes in the interference range of a node, transmitting in this slot will lead to less additional blocked nodes and therefore increase the utilization of the schedule, which leads to an increased chance of finding feasible schedules for other routes. This most reuse first heuristic is shown in the following:

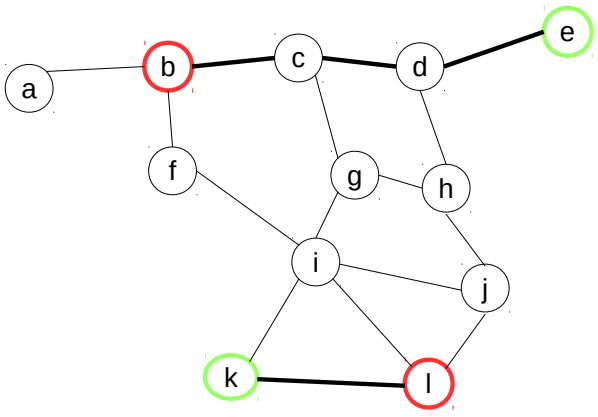
```
/* Returns the slot that is blocked in the most interference neighbors
    of the current nodes */
getMostReusedSlot(link currentLink = [a,b], list remainingSlots) {
        /* iterate over all remaining Slots */
        for(slot i : remainingSlots) {
                /* and count how often the slot is free in all
                    interference neighors */
                for(node x ∈ I_a ∪ I_b){
                        if(i ∈ IFS_x) {
                                count[i]++
                        }
                }
        }
        /* choose and return the slot that is free least often */
        return minIndex(count)
}
```

getMostReusedSlot counts how often a slot is free in all interference neighbors of sender and receiver of the current link and returns the slot that is free the smallest number of times, i.e., the slot that is blocked the most.
These three principles are combined to form an algorithm that has a high chance of finding a schedule and maximizes the utilization, so that scheduling following routes is easier:

```
/* Assigns slots to a path p, maximizing the Utilization*/
assignSlotsUtil(path p): {
        /* Continue scheduling until all links are scheduled */
        for(i=0; i < |p|: i++) {
                /* determine the link with the least available slots */
                currentLink = [a,b] = getCurrentLink(p)
                /* if no slots remain, the scheduling failed */
                if(|IFS_[a,b]| == 0) {
                        return(SlotAllocationFailed)
                }
                /* otherwise, determine the chosen slot according to
                    the least conflict and most reused first policies
                    */
                leastConflictSlots = getLeastConflictSlots(p,
                    currentLink)
                chosenSlot = getMostReusedSlot(currentLink,
                    leastConflictSlots)
                /*reserve the chosenSlot for sending and receiving */
                        setSlot(a, currentSlot, SEND_b)
                        setSlot(b, currentSlot, RECEIVE_a)
                        /* and block the chosenSlot in all nodes in the
                            interference range of the sender and the
                            receiver */
                        for( node x ∈ I_a ∪ I_b)
                                setSlot(x, currentSlot , BLOCKED)
                /* and mark the currentLink as scheduled */
                alreadyScheduled[currentLink] = true
        }
        return(ScheduleFound)
}
```

assignSlotsUtil schedules slots for a given path p according to the policies explained previously. If a slot is found, it is reserved for sending and receiving and blocked in all nodes in the interference range of sender and receiver. The scheduling fails if one of the links has no free slot left and is successful if all links are scheduled.

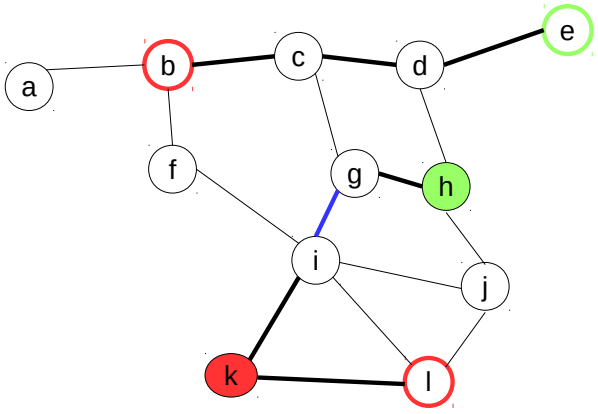
Figure 4.5 and following, show an example of this scheduling algorithm. The starting point for this example is a network with two already scheduled paths as shown in Figure 4.5. It is assumed that the communication range of all nodes corresponds to their interference range.



Slot:	0	1	2	3	4
node a:	x				
node b:	S_c	x			
node c:	R_b	S_d		x	
node d:	x	R_c		S_e	
node e:		x		R_d	
node f:	x				
node g:	x	x			
node h:		x		x	
node i:			x		
node j:			x		
node k:			R_l		
node l:			S_k		

Figure 4.5: The starting point of a scheduling example for *assignSlotsUtil*.

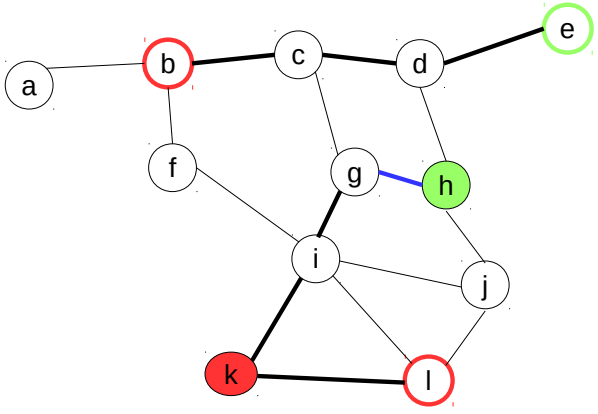
Now, the links for another path $p = \langle k, i, g, h \rangle$ have to be scheduled. The first step of the algorithm is to count the number of interference-free slots for all links on the path. Here, link $[k, i]$ has four interference-free slots. Link $[i, g]$ has two interference-free slots, because node g is blocked in two slots and node i is blocked in another slot. Link $[g, h]$ also has two interference-free slots, because at least one of the nodes is blocked in three slots. Therefore, the algorithm starts allocating slots in link $[i, g]$.
The next step (least conflict first policy) is to count how often the free slots of link $[i, g]$ are free on other links of the route. Slot 3 is free in link $[k, i]$, but is blocked in node h and therefore not free in link $[g, h]$. Slot 4 is free in both other links of the path. According to the least conflict first policy this means that slot 3 is chosen for this link. Since there is only one slot left at this point, the most reused first policy is not needed. Therefore, slot 3 is scheduled to send resp. receive in nodes i resp. g and blocked in all of their interference neighbors (nodes c, f, h, j, k and l). The updated schedule is shown in Figure 4.6.



Slot:	0	1	2	3	4
node a:					
node b:	S_c	x			
node c:	R_b	S_d		x	
node d:	x	R_c		S_e	
node e:		x		R_d	
node f:	x			x	
node g:	x	x		R_i	
node h:		x		x	
node i:			x	S_g	
node j:			x	x	
node k:			R_l	x	
node l:			S_k	x	

Figure 4.6: Continued scheduling example.

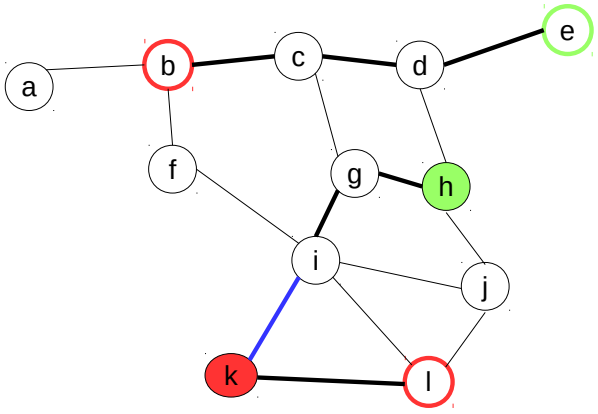
Next, the interference-free slots of the remaining links have to be counted. Link $[k, i]$ has three interference-free slots left and link $[g, h]$ has two free slots, so link $[g, h]$ is chosen. Again, the least conflict first policy has to be applied. Slot 2 is blocked in link $[k, i]$, while slot 4 is free. So, slot 2 is chosen for this link and scheduled and blocked accordingly. The resulting schedule is shown in Figure 4.7.



Slot:	0	1	2	3	4
node a:					
node b:	S_c	x			
node c:	R_b	S_d	x	x	
node d:	x	R_c	x	S_e	
node e:		x		R_d	
node f:	x			x	
node g:	x	x	S_h	R_i	
node h:		x	R_g	x	
node i:			x	S_g	
node j:			x	x	
node k:			R_l	x	
node l:			S_k	x	

Figure 4.7: Continued scheduling example.

After this, only link $[k, i]$ has to be scheduled. Three slots are still free in this link. Applying the least conflict first policy is unnecessary, because there is no other link to be scheduled. Therefore, the most reused first policy is applied. It is counted how often the free slots (slot 0, 1 and 4) are free in the interference neighbors of nodes k and i (nodes f, g, j and l). Slot 0 is free in two of them, slot 1 is free in three of them and slot 4 is free in all of them. According to the most reused first policy, this means that slot 0 is chosen. The final schedule is shown in Figure 4.8.



Slot:	0	1	2	3	4
node a:					
node b:	S_c	x			
node c:	R_b	S_d	x	x	
node d:	x	R_c	x	S_e	
node e:		x		R_d	
node f:	x			x	
node g:	x	x	S_h	R_i	
node h:		x	R_g	x	
node i:	R_k		x	S_g	
node j:	x		x	x	
node k:	S_i		R_l	x	
node l:	x		S_k	x	

Figure 4.8: Continued scheduling example.

The resulting delay for a transmission from k to h is 8 micro slots, because it starts in micro slot 0 of one super slot and ends in micro slot 2 of the next super slot. There are seven transmissions and receptions and 19 blocked slots in the schedule, therefore the utilization of it is $\frac{7+7}{19} = 0.74$.

4.5.3 Scheduling Algorithm – Additions

4.5.3.1 Adding a Destination to a Tree

Section 4.2 explained that whenever a path is added to an existing node, local multicasting can occur in the first node of path. Therefore, whenever a destination is added to tree, the scheduling has to be altered for the first node on the added path.

The first step is to determine in which slot the first node is scheduled to send in this tree. It has to be distinguished between transmissions that belong to the current tree and transmissions from the same node that belong to another tree. After identifying the right transmission, the number of nodes scheduled to receive this transmission has to be counted.

If there are less than three receivers, local multicasting can be used. In that case, the receiver (second node of the first link of the added path) has to be scheduled to receive in the right slot (the slot in which the first node sends the transmission) if possible. The new receiver may already be blocked in that slot by a transmission of another node that is not part of the local multicast. If the slot can be used, all nodes in the interference range of the receiver have to be blocked (unless they are also receivers). At this point, the nodes in interference range of the sender are already blocked.

Additionally, if using *assignsSlotsDelay*, the *getStartingSlot* function has to be altered. The currentSlot value has to be set to the slot directly after the one in which the first node sends, so that the scheduling of the rest of the links continues to minimize the delay.

If there are already three receivers or the new receiver is already blocked in the rele-

vant slot, no local multicasting can be used. In this case, the currentSlot has to bet set to the slot in which the first node is scheduled to receive in the existing tree, so that the scheduling continues as usual (in the slot directly after the slot used in the previous link). In the *assignsSlotsUtil* function, the available slots are determined independently for all links, and so only the first link has to be scheduled differently.

4.5.3.2 Scheduling Including Mobile Nodes

Scheduling a tree that contains a mobile node is treated as a different case. There are two ways, in which including mobile nodes changes the scheduling algorithm. The first difference occurs in the slot used for the communication between AccessNodes and a mobile node. Another addition to the algorithm is needed for the path from the AccessNodes to the DistributorNode in a tree with a mobile node as the source.

Scheduling the Slot from AcceessNodes to Mobile Node

Scheduling the slot for communication between a mobile node and AccessNodes is different, because – as explained in Section 4.3 – SDMA cannot be used for this slot. Therefore, the scheduling algorithm has to be altered:

```
1  assignMobileSlot(node mobile): {
2          /* go through all slots*/
3          for(currentSlot < numberOfSlots){
4                  /* if the slot is free in all nodes */
5                  if(currentSlot ∈ IFS_a|_{a∈G}){
6                          /* reserve it for sending in the mobile node */
7                  setSlot(mobile,currentSlot,SEND_accessNodes)
8                  /* and for receiving in all AccessNodes*/
9                  for(node an :AccessNodes):{
10                         setSlot(an,currentSlot,RECEIVE_mobile)
11                 }
12                 /* and block it in all other nodes */
13                 for(node x : network
                       G ∧ x ∉ AccessNodes ∧ x! = mobile) {
14                 setSlot(x,currentSlot,BLOCKED)
15                 }
16             return(ScheduleFound)
17             }
18         }
19         return(SlotAllocationFailed)
20 }
```

Since no SDMA is used, the used slot has to be free in all nodes of the network and is scheduled to send from the mobile node to all AccessNodes. If the mobile node is the receiver, all AccessNodes schedule the used slot for transmitting and the mobile node schedules this slot for receiving. The first slot that is completely free can be used regardless of the scheduling strategy for the rest of the tree. For the utilization it does not matter which slot is chosen, because the number of nodes being blocked is the same in all completely free slots.

In the *assignSlotsDelay* function for a path with the mobile node as sender, all other links will be scheduled after this link and the delay will be minimized as much as possible. If the mobile node is a receiver, then all other links have already been

scheduled, which means that the first slot that is still completely free is later than the slots used for the previous links.

Scheduling Paths involving a Mobile Node

Besides scheduling the slot for the link from the AccessNodes to the mobile node, the scheduling for the trees involving a mobile node has to be discussed. The first case is trees with the mobile node as a destination. As explained in Section 3.5.3.1, these trees are essentially stationary to stationary trees with the AccessNodes as destinations. Therefore, besides the last hop from AccessNodes to the mobile node (explained in the previous section), the scheduling is the same as the scheduling for purely stationary trees and will not be discussed in detail.

Additionally, there are trees with a mobile node as the source, which is the case when nodes subscribe to a service published by the mobile node. In Section 3.5.3.2 it was explained that the trees are a bit different in this case, because only one branch of this tree can be active at any time. Therefore, different branches can never interfere with each other. All of them can, however, interfere with communication from other trees. This leads to a different scheduling strategy, because the usual strategies would be ineffective.

The scheduling starts in one of the branches, using one of the algorithms for stationary trees. However, while checking if the a slot is free, transmissions from other branches of this tree are treated as non existing. This means, if a slot is blocked, it has to determined whether it is blocked by a transmission from a node in another branch of this tree or by a node from this branch or a completely different tree. If the slot is blocked by a transmission from another branch of this tree, and only from this transmission, it can be treated as free, because this transmission will never occur when current branch is active. If it is blocked for other reasons, it is not free.

For the slot assignments, the *assignSlotsDelay* and *assingSlotsUtil* function can be used, with the additions explained above. This can, but does not necessarily, lead to the same used slots in all branches.

The order in which the paths are scheduled is not important in this case, because the schedules do not depend on each other. Local multicasting will never be used here, because all routes are essentially unicast routes.

The tree from the DistributorNode to the rest of the destinations is scheduled as a stationary multicast tree. If using the *assignSlotsDelay* function, the starting delay should be set to the first slot, after the latest slot in which the DistributorNode is scheduled to receive from one of the branches from the tree from the AccessNodes to itself, so that the delay from all AccessNodes to all destinations is as small as possible.

4.5.3.3 Scheduling Multiple Trees

One point that has not been discussed up to now, is if the scheduling should change if there are already existing trees in the network. The main point of interest is, in which slot the scheduling starts in this case. Using *assignSlotsDelay* the scheduling of a new tree currently starts in slot 0. If there is no other tree in the network, this

will continue to be the case. However, as soon as there is another tree, it is possible to start scheduling in the first slot after the last slot of the existing tree. In that case, the resulting delay will usually be shorter, because there normally will be more completely free slots in a slot region that is not used thus far. This is a heuristic, because it is possible that the trees do not interfere with each other anyway e.g., when two trees are in completely different regions of the network. Therefore, it is possible that starting the scheduling in a slot later than the slots used by the other trees in the network, does not yield an advantage. In relatively small networks where a transmission often interferes with nearly all nodes in whole network, that should rarely be the case.

Chapter 5

Evaluation

To evaluate the presented algorithms, the route discovery and slot scheduling functionalities from Chapters 3 and 4 were implemented as a stand-alone C++ application. The network has to be pre-configured as a number of nodes with their communication and interference links. The topology is fixed and for mobile nodes, the AccessNodes have to be specified.

The input of the application is the source and destinations of a multicast tree as well as the total number of micro slots available. All presented algorithms and functions can be applied to the paths of the trees. It has to be decided which route discovery and scheduling algorithms should be applied to which trees / branches. Nodes can be used to create a new tree with one source and destination. Other destinations can be added to this tree or the scheduling can be applied to one of the paths in the tree. The scheduling has to be applied to the paths in the order in which they have been added to the tree, to obtain correct results. This is necessary, because the first node on all but the first path is scheduled differently from the rest of the nodes (see Section 4.5.3.1). The route request and route confirm phase are not simulated, neither is the data traffic on the generated routes.

The output of the application is a graph, generated with the igraph package [1] for R [3] and the corresponding schedule in LaTeX [2] table format. Additionally, the values needed for the evaluation according to the assessment criteria given in Sections 3.1 and 4.4 are returned. This includes the number of hops in a tree (taking multicasting into account), the length of the individual paths and the distance from the destinations to the source (using the paths in the tree). Also, the utilization of the schedule and the delays for each destination can be produced. The cost of a path, as defined in Section 3.1.3, is not needed anymore, because (while used to discuss the trees) it is replaced by the utilization of the complete schedule.

5.1 Evaluation Setup

To evaluate the algorithms, two different network topologies are used. The first (left side in Figure 5.1) represents a general network, where the nodes and links are relatively evenly distributed.

The second topology is a 3-hop connected 1-hop dominating set, a possible result of the clustering algorithm mentioned in Section 2.4. Here, the orange nodes are

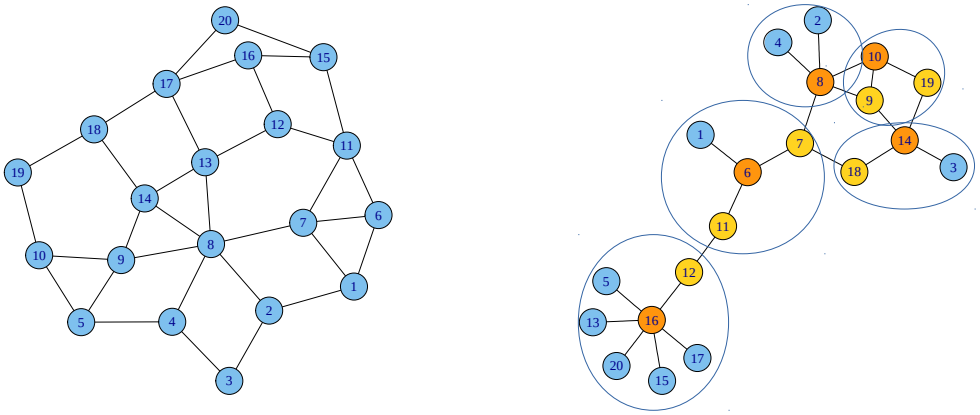


Figure 5.1: The two topologies used for the evaluation. A general network with out a special topology (l.) and a second topology that forms a 3-hop connected 1-hop dominating set (r.)

cluster heads, the yellow nodes are gateways and the rest of the nodes followers. From a functional point of view, the blue nodes could represent a set of Reduced Functional Nodes. In this topology, all RFNs are connected to a single other node, i.e., they can only be part of a route, if they are the source or a destination. This could be useful in some application contexts, e.g., if the RFNs are battery powered, because they will use less energy if they are not part of other routes.

In the following experiments, it is always assumed that the interference range of nodes contains all nodes in the communication range of their neighbors. This means that, especially in the first topology, the interference between nodes is high, i.e., nodes have a lot of other nodes in their interference range. For example, if node 8 sends to node 13 in network topology 1 (and node 13 replies with an ACK), all nodes except nodes 15 and 19 will be blocked in the used slot.

The first topology was chosen to evaluate the algorithm for general networks with evenly distributed nodes. In the second topology, the interference between nodes is generally lower and the average distances between nodes is higher, e.g., the network diameter is five in topology 1 and eight in topology 2. The second topology was chosen to evaluate the algorithms on networks fulfilling the clustering properties from the problem description in Chapter 2.

In addition to the stationary nodes, mobile nodes will be added to the network in some experiments, which means that some nodes will be chosen as AccessNodes and one node as the DistributorNode.

5.2 Evaluation Scenarios

To evaluate the general performance of the presented algorithms, 50 randomly generated scenarios for stationary trees were created. Every scenario contains three

stationary multicast trees with one source and three different destinations. The different trees in one scenario can contain the same nodes. These scenarios are used to compare the general performance of the different route discovery and scheduling algorithms (see Section 5.3) to each other in completely stationary networks. Both network topologies use the same scenarios.

Another important parameter for the evaluation is the number of micro slots in every super slot. If more slots are available, finding a feasible schedule is easier, e.g., if there are more completely free slots available than transmissions needed in a tree, finding a schedule is guaranteed. As the objective of some of the functions is to increase the chance of finding a feasible schedule, the number of micro slots is variable and depends on the experiment.

In addition to those general performance evaluations, some experiments (e.g., the evaluation of the AccessNodes) will only contain a single scenario and the results of the route discovery and scheduling of this scenario will be examined in detail.

5.3 General Performance

The different algorithms for the route discovery and scheduling phase can be grouped into algorithms that minimize the delay and algorithms that increase the utilization and chance of finding a feasible schedule. Therefore, two setups of route discovery algorithms combined with a scheduling algorithm are used to evaluate the general performance.

- **minDelay**: This setup aims at minimizing the delay of the individual paths of each tree as much as possible. To achieve this, the tree is created with *createNewTreeDegree* (Section 3.5.1). New destinations are added with *addDestToTree* (Section 3.5.2), which chooses the path with the highest degree, if there are multiple shortest paths to the tree with the same distance to the root. Finally, the slots are allocated with *assignSlotsDelay* (Section 4.5.2.1). All of these functions aim at creating a tree with short paths and allocate slots, so that the resulting delay is minimal. The scheduling of new trees always starts in slot 0 for the first two experiments. The alternative, to start in the slost after the last slot used in the previous tree, will be tested too (see Section 4.5.3.3).

- **maxUtil**: In this setup, the utilization and the chance of finding a feasible schedule is maximized. For the tree creation *createNewTree* (Section 3.5.1) is used. The threshold for the *getMinNumberOfFreeSlots* function is set to three, as a compromise between increasing the chance of finding a schedule and minimizing the utilization of it. To add a destination, *addDestToTree* (Section 3.5.2) is used (using the number of free slots to choose one of the shortest paths with minimal distance to the root). For the slot allocation, *assignSlotsUtil* (Section 4.5.2.2) is used to further increase the chance of finding a schedule.

The first set of the experiments will evaluate the performance of the algorithms on stationary multicast trees. The algorithms of both setups are executed for all 50

scenarios for both network topologies and the average utilization of the schedule and the average delay from source to each destination are used to evaluate the quality of the results of both setups. In addition to that, the number of scenarios in which the algorithm finds a feasible schedule is examined.

As a starting point in Section 5.3.1 the number of micro slots is set to 15. In Section 5.3.2 the same experiment is done with a lower number of micro slots (12 and 10) and Section 5.3.3 examines the effect of the starting slot for *assignSlotsDelay* with 15 and 12 micro slots.

5.3.1 Delay and Utilization

For the first evaluation, the number of dynamic micro slots per super slot is set to 15, so that both setups find a feasible schedule in all scenarios. Table 5.1 shows the results of this experiment for network topology 1 and Table 5.2 shows the results for topology 2.

Network Topology 1	avg. util	avg. distance	avg. delay	# found Schedules
minDelay	0.21 ± 0.02	2.76 ± 1.23	3.6 ± 2.25	50
maxUtil	0.22 ± 0.02	2.74 ± 1.24	6.06 ± 6.39	50

Table 5.1: The results of the first experiment with **15 micro slots** for network topology 1.

Network Topology 2	avg. util	avg. distance	avg. delay	# found Schedules
minDelay	0.29 ± 0.01	3.99 ± 2.21	5.7 ± 4.34	50
maxUtil	0.31 ± 0.01	3.99 ± 2.21	11.22 ± 11.15	50

Table 5.2: The results of the first experiment with **15 micro slots** for network topology 2.

The value **avg. util** shows the arithmetic mean and standard deviation of the utilization of the found schedules. The value **avg. distance** shows the average distance (in links) from the source to a destination in a tree (this is not necessarily the same as the distance in the network).The value **avg. delay** shows the mean and standard deviation of the delay to each individual destination. The value # **found Schedules** gives the number of scenarios in which the setup found a feasible schedule.

Both, average utilization and delay are higher in network topology 2. This is caused by the longer paths and – generally – less nodes in the interference range of each node in topology 2. The longer paths are also confirmed by the greater average distance from source to destination in topology 2.

There is a large difference in the average delay between the two setups. In the *maxUtil* setup, the average delay of the paths is roughly two times as long as in the *minDelay* setup. In both topologies, the standard deviation in the *maxUtil* setup is nearly three times as big as in the *minDelay* setup. The rather large difference in

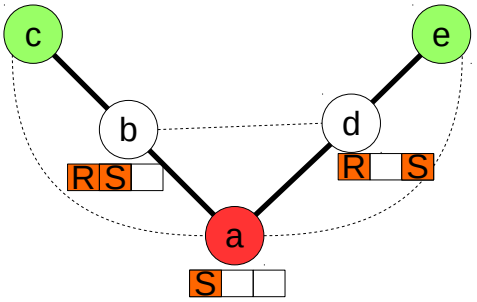


Figure 5.2: An example for the minimum delay of a tree.

delay is not surprising, because *assignSlotsDelay* is aimed at minimizing the delay, while *assignsSlotsUtil* ignores the delay completely. The high standard deviation of the delay for the *maxUtil* setup shows that the delay in this setup is not only high, but also hard to predict. Therefore, for time-critical tasks the *allocateSlotsDelay* function may be the better choice.

In the *minDelay* setup, the average delay is about one or two slots higher than the average distance. In Section 4.4, it was explained that the minimum delay of a path is its number of links, which is the same as the distance from the source to the destination. The fact that the delay is higher than the distance can be explained as follows.

Consider the following example of a tree $\langle c, b, a, d, e \rangle$, also shown in Figure 5.2. Here, it is assumed that node a is the source, and nodes c and e are destinations of a tree. In the first free slot node a can send to nodes b and d (local multicast). But in the second free slot, both nodes b and d can not send together, because they are in each others interference range (which was assumed to contain the 2-hop communication range). So, the delay is 2 for one of the paths and 3 for the other path in the best case. This situation will obviously occur often in trees. Another increase in delay is probably caused by the starting slot for the scheduling with *assignSlotsDelay*, which is set to slot 0 in this experiment, i.e., it is possible that the first few links in a path are scheduled in a different slot region than the later slots, as explained in Section 4.5.3.3. It is expected, that the delay is shorter if the scheduling starts in the slot after the last slot used for the previous trees. This will be confirmed in Section 5.3.3.

There is no large difference in the average utilization of the schedules for the different setups in both topologies. The mean utilization of the schedules differs by less then 10% and the standard deviation is nearly the same in both setups. This indicates that both strategies block more or less the same number of slots with their transmissions. A likely reason for this is that the trees from the different route discovery algorithms are usually very similar to each other. All route discovery algorithms aim at reducing the number of links in the added paths, and they only produce different results if there are multiple shortest paths that also have the same distance to the source of the tree. The nearly equal average distance in both setups

also indicates that the created trees are very similar. This can be further confirmed by examining the average number of links (taking local multicast into account) or the average length of the paths added to a tree, which are nearly the same in both setups (e.g., the average number of links in trees in topology 1 is 4.45 and 4.38 for the two setups). As the created trees are so similar in both setups, these values have been omitted from the tables. Another reason for the low difference in the utilization is that during the slot allocation, maximizing the utilization is only one of the applied policies of the *assignSlotsUtil* function. The other policies are mostly used to increase the chance of finding a feasible schedule.

Furthermore, the example networks are relatively small and the interference between nodes is relatively high. Therefore, a transmission in a slot blocks this slot in a lot of other nodes. So, the number of slots that can be reused is generally low. This may be a reason for the bigger difference in utilization in topology 2, which has less interference. However, as the difference is still small this not enough to draw a general conclusion.

5.3.2 Lowering the Number of Micro Slots

To further assess the effectiveness of the presented algorithm, the experiment was repeated with a lower number of micro slots per super slots, which was set to 12 for the next experiment. Tables 5.3 and 5.4 show the the results of the same experiment with a total of 12 micro slots.

Network Topology 1	avg. util	avg. distance	avg. delay	# found Schedule
minDelay	0.21 ± 0.02	2.74 ± 1.24	3.63 ± 2.5	46
maxUtil	0.22 ± 0.02	2.73 ± 1.23	5.35 ± 5.08	48

Table 5.3: The results of the experiment with **12 micro slots** slots for topology 1.

Network Topology 2	avg. util	avg. distance	avg. delay	# found Schedule
minDelay	0.3 ± 0.01	3.63 ± 2.06	5.36 ± 5.14	26
maxUtil	0.31 ± 0.01	3.96 ± 2.22	9.08 ± 8.34	45

Table 5.4: The results of the experiment with **12 micro slots** slots for topology 2.

The difference in delay in the two setups (both mean and standard deviation) is slightly smaller in this case, but still there. Scheduling a transmission before the slot used in the previous link does not increase the delay as much as in the previous experiment, if there are less micro slots per super slot. This is more noticeable in the *maxUtil* setup, because used slots are determined without taking preceding links into account. An outlier in this experiment is the average distance in the *minDelay* setup in network topology 2. This is probably because of the low number of found schedules. Here, *assignSlotsDelay* can only find a schedule in small trees, and if only small trees are examined, the average distance will be lower. If the average distance would be calculated for all trees, even those in which no schedule was found, it would

be the same as in the previous experiment, because the trees do not depend on the total number of micro slots available (in this setup).

Here, the main point of interest is the number of found schedules. In both topologies, the *assignSlotsUtil* function finds more feasible schedules. In fact, it finds a schedule in 48 (45) of the 50 scenarios, compared to the 46 (26) scenarios in which the *assignSlotsDelay* function finds a feasible schedule in topology 1 (2). In the *assignSlotsDelay* function, no effort to increase the chance of finding a feasible schedule is made. Essentially, every link is assigned the first slot that is free in both sender and receiver. Compared to this, *assignsSlotsUtil* has two policies to increase the chance of finding a feasible schedule for a route, which were explained in Section 4.5.2.2. The results show that the policies actually increase the chance of finding a schedule, as expected.

This trend continues when the number of slots is lowered further. For a total of 10 micro slots per super slot, *assignSlotsUtil* finds a feasible schedule in 27 scenarios in network topology 1, while *assignSlotsDelay* finds a feasible schedule in only 22 of the scenarios.

A conclusion so far is, that the *minDelay* setup is better for networks with low traffic (or a large number of free micro slots), because *maxUtil* yields only a small increase in utilization, but results in a higher delay and higher variance in delay. For networks with a small number of micro slots – or a lot of traffic – the *maxUtil* setup is better, because the chance of finding a feasible schedule is increased compared to the *minDelay* setup.

5.3.3 Scheduling Multiple Trees

For the *assignSlotsDelay* function, several (different) options for the starting slot of each tree were given. One option, which was used in the previous examples, is to start every new tree in slot 0. Another option is to start scheduling the new tree in the slot directly after the last slot used in the previous tree. The different options will be evaluated in the following.

Tables 5.5 and 5.6 show the results of *minDelay*, starting the slot allocation of a new tree directly after the last slot of the previous tree, with a total number of 15 resp. 12 micro slots.

minDelay	avg. util	avg. distance	avg. delay	# found Schedule
topology 1	0.2 ± 0.01	2.73 ± 1.21	3.02 ± 1.39	48
topology 2	0.29 ± 0.01	3.88 ± 2.18	5.03 ± 4.57	43

Table 5.5: Results of *assignSlotsDelay*, when starting the scheduling in the slot after the last slot used in the previous tree and a total of **15 micro slots**.

minDelay	avg. util	avg. distance	avg. delay	# found Schedule
topology 1	0.2 ± 0.02	2.66 ± 1.21	3.05 ± 1.74	38
topology 2	0.3 ± 0.01	3.41 ± 2.17	4.71 ± 4.56	11

Table 5.6: Results of *assignSlotsDelay*, when starting the scheduling in the slot after the last slot used in the previous tree and a total of **12 micro slots**.

Compared to the previous results (e.g., Table 5.1), the average delay is slightly lower. If the total number of micro slots is 15, the avg. delay for network topology 1 is lowered from 3.6 to 3.02. However, the number of scenarios in which a suitable schedule is found is lowered, too (from all 50 to 48). The same is true for topology 2, where the delay is lowered from 5.7 to 5.03, while a schedule is only found in 43 instead of 50 scenarios. The reduced chance of finding a schedule is more noticeable if there are less micro slots in general. If there are only 12 slots, the number of scenarios in which a schedule was found is reduced from 46 (26) to 38 (11) for topology 1 (2). The results are as expected. By starting the scheduling in a later slot, transmissions that would normally fit in earlier slots are now scheduled in a completely free slot. As long as the rest of the nodes on the paths still fit in the schedule, the delay will often be lower, but by ignoring some free slots (which may not be free in other links of the route) that are in the already used part of the schedule, the chance of finding a schedule is lowered.

5.4 Mobile Nodes as Destination

The mobile nodes communicate with the rest of the network through the AccessNodes (see Section 3.5.3).
A rather simple approach to reach the mobile node would be to create a route and schedule to all other nodes in the network (i.e., set all nodes as AccessNodes). To assess the general effectiveness of the AccessNodes, the tree from a source to all other nodes is compared to the tree from the same source to all AccessNodes.

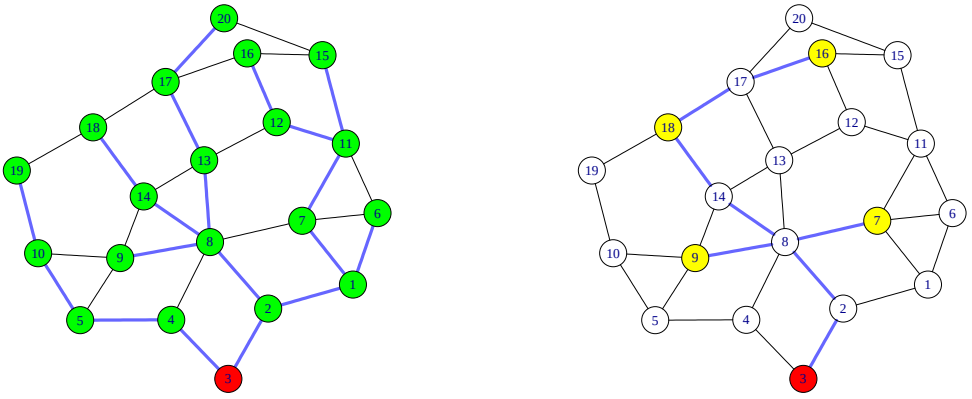


Figure 5.3: A multicast tree to all other nodes (l.) and a multicast tree to all AccessNodes (r.).

Figure 5.3 shows the multicast tree from a source to all other nodes on the left side and the tree from the same source to all AccessNodes on the right side. The trees, and following schedules, were created with the setup *minDelay*, explained in the previous sections. Table 5.7 shows the schedule for sending to all other nodes, while Table 5.8 shows the schedule to reach all AccessNodes. As expected, the difference in the schedules is huge. Significantly less slots are needed to reach the AccessNodes, compared to reaching all other nodes. Even moving the AccessNodes further away from the source would not lead to a large change in the schedule. E.g., node 16 could transmit in slot 0 and 1 or node 7 could send to node 6 in slot 4 or 5, to reach AccessNodes that are further away from the source, without many additional blockings. The maximum delay to reach all other nodes is 11 micro slots for the path from node 3 to node 20 and a total of 128 slots are blocked in this case. On the other hand, when using the AccessNodes the maximum delay is 6 micro slots for the path from node 3 to node 16 and only a total of 69 slots are blocked.

As the difference in used slots and delay is so large, doing more experiments for this case is unnecessary. It is obvious that using AccessNodes leads to significant advantages for mobile multicast trees.
However, the schedule to reach all nodes (Table 5.7) can be used to point out the effectiveness of local multicasting. Local multicasting is used in four nodes, saving a total of five transmissions in every super slot. Without local multicasting, no schedule could be found in this example, because, e.g., node 8, which sends to three receivers in slot 4 has no other free slot left.

Another property of local multicasting can be seen in node 1. Node 1 sends to nodes 6 and 7 in this example, but can not use local multicasting, because node 7 is blocked in slot 2 by the transmission from node 4 to node 5 in the same slot.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x	R_2	S_6	S_7	x	x	x			
Node 2:	R_3	$S_{1\ 8}$	x	x	x	x			x	x
Node 3:	$S_{2\ 4}$	x	x	x	x					
Node 4:	R_3	x	S_5	x	x	x			x	x
Node 5:	x	x	R_4	S_{10}	x	x				x
Node 6:	x	x	R_1	x	x	x	x	x		
Node 7:	x	x	x	R_1	x	S_{11}	x	x	x	x
Node 8:	x	R_2	x	x	$S_{9\ 13\ 14}$	x	x	x	x	x
Node 9:	x	x	x	x	R_8	x			x	x
Node10:	x	x	x	R_5	x	S_{19}				x
Node11:	x	x	x	x	x	R_7	$S_{12\ 15}$	x	x	
Node12:	x	x	x	x	x	x	R_{11}	S_{16}	x	x
Node13:	x	x	x	x	R_8	x	x	x	S_{17}	x
Node14:	x	x	x	x	R_8	x	x	x	x	S_{18}
Node15:	x		x	x		x	R_{11}	x	x	
Node16:	x				x	x	x	R_{12}	x	x
Node17:	S_{20}	x			x	x	x	x	R_{13}	x
Node18:	x	x		x	x	x		x	x	R_{14}
Node19:	x		x	x	x	R_{10}			x	x
Node20:	R_{17}				x	x	x	x	x	x

Table 5.7: The schedule for the multicast tree from one source (3) to all other nodes.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x	x	x							
Node 2:	R_3	S_8	x	x						
Node 3:	S_2	x	x							
Node 4:	x	x	x	x						
Node 5:	x	x	x	x						
Node 6:	x	x	x							
Node 7:	x	x	R_8	x						
Node 8:	x	R_2	$S_{7\ 9\ 14}$	x	x	x				
Node 9:	x	x	R_8	x	x					
Node10:		x	x	x	x					
Node11:		x	x			x				
Node12:		x	x	x	x	x				
Node13:	x	x	x	x	x	x				
Node14:	x	x	R_8	S_{18}	x	x				
Node15:			x		x	x				
Node16:				x	x	R_{17}				
Node17:		x	x	x	R_{18}	S_{16}				
Node18:		x	x	R_{14}	S_{17}	x				
Node19:			x	x	x	x				
Node20:				x	x	x				

Table 5.8: The schedule for the multicast tree from one source (3) to all AccessNodes.

5.5 Mobile Nodes as Source of a Tree

Another important case are trees with the mobile node as the source of a transmission. In this case, the data is sent from the AccessNodes to the DistributorNode, with a specialized scheduling strategy, in which the different branches of this tree can be scheduled in the same slots. The different branches of the tree are never active at the same time as explained in Section 3.5.3.2. To show the advantages of this scheduling strategy, a tree from the AccessNodes to the DistributorNode is first scheduled with the normal strategy (although without multicasting) and then with the algorithm that was proposed for this case.

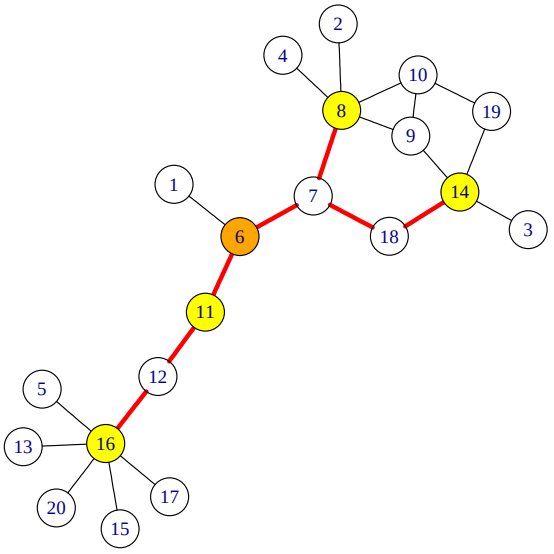


Figure 5.4: A multicast tree from all AccessNodes to one DistributorNode (6).

Figure 5.4 shows the multicast tree from the AccessNodes (yellow) to the DistributorNode 6. The DistributorNode was chosen with the *getDistributorNode* function, i.e., node 6 has the lowest average distance to all other nodes. As this node is responsible to forward all traffic from the mobile nodes to the rest of the network, this will – on average – lead to shorter paths from one AccessNode to any other node of the network.

The right schedule in Table 5.5 was created by the specialized algorithm developed for this case. In this schedule, nodes from the different branches of the tree are allowed to send / receive in the same slots, because they will never be active at the same time. For example, nodes 8 and 14 are both scheduled in slot 0, even though they, and their receivers, are in each other's interference range. Also, node 6 is scheduled to receive from nodes 7 and 11 in slot 2. Again, this is possible because these nodes will never send concurrently.

Slot:	0	1	2	3	4	5	6
Node 6:	R_{11}	x	R_7	x	x	R_7	R_{11}
Node 7:	x	R_8	S_6	x	R_{18}	S_6	x
Node 8:	x	S_7	x	x	x	x	x
Node 9:		x	x	x	x	x	
Node10:		x	x	x	x	x	
Node11:	S_6	x	x	R_{12}	x	x	S_6
Node12:	x	R_{16}	x	S_{11}		x	x
Node13:		x		x			
Node14:		x	x	S_{18}	x	x	
Node15:		x		x			
Node16:	x	S_{12}		x			x
Node17:		x		x			
Node18:	x	x	x	R_{14}	S_7	x	x

Slot:	0	1	2
Node 6:	R_{11}	R_7	$R_{7\ 11}$
Node 7:	R_8	R_{18}	S_6
Node 8:	S_7	x	x
Node 9:	x	x	x
Node10:	x	x	x
Node11:	S_6	R_{12}	S_6
Node12:	R_{16}	S_{11}	x
Node13:	x	x	
Node14:	S_{18}	x	x
Node15:	x	x	
Node16:	S_{12}	x	x
Node17:	x	x	
Node18:	R_{14}	S_7	x

Figure 5.5: The resulting schedules for the tree from the AccessNodes to the DistributorNode.

The left schedule is scheduled with the normal scheduling strategy, in this case *assignSlotsDelay*, without local multicasting. It shows clearly that more slots are needed and blocked in this case. As the difference is large and the different branches of the tree from the AccessNodes to the DistributorNode are at least partially in each other's interference range (because they end in the same node), this is enough to conclude that the specialized scheduling strategy leads to a better schedule in most cases.

However, it is possible to construct examples where the resulting schedules are the same in both scheduling strategies, but even in the worst case (if no slot can be shared by the different branches), the specialized scheduling strategy does not produce a worse schedule. Figure 5.6 shows an example concast tree, for which all scheduling strategies would find the same schedule, because f and e can send at the same time anyway (under the assumption that the interference range is the 2-hop communication range). However, in practice this should rarely be the case.

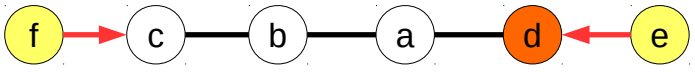


Figure 5.6: An example concast tree in which all scheduling strategies find the same schedule.

5.6 Examining an Example Tree

For a better understanding of the route discovery and scheduling process, a step by step example of the complete process will be shown in the following. This example uses network topology 2 (Figure 5.1) with 10 micro slots. Two mobile nodes M1 and M2 have been added to the network. Nodes 6, 9 and 16 are specified as the AccessNodes for M1 and are shown with a yellow border in all figures. Nodes 7, 12 and 2 are specified as the AccessNodes for M2, shown with a purple border in all figures. Sources of trees are shown in red and destinations in green. The DistributorNode is shown in orange.
For a better understanding, the trees are depicted with arrows pointing in the direction that data is send, even though the links are undirected. Also, some links are shown as multiple, differently colored arrows, which means that the link is part of two different trees. The mobile nodes are each connected to one of their AccessNodes in all figures. Of course, they can be connected to other AccessNodes at different times. In all tables, changed reservations are colored. Reservations that did not change since the previous step are shown in black.

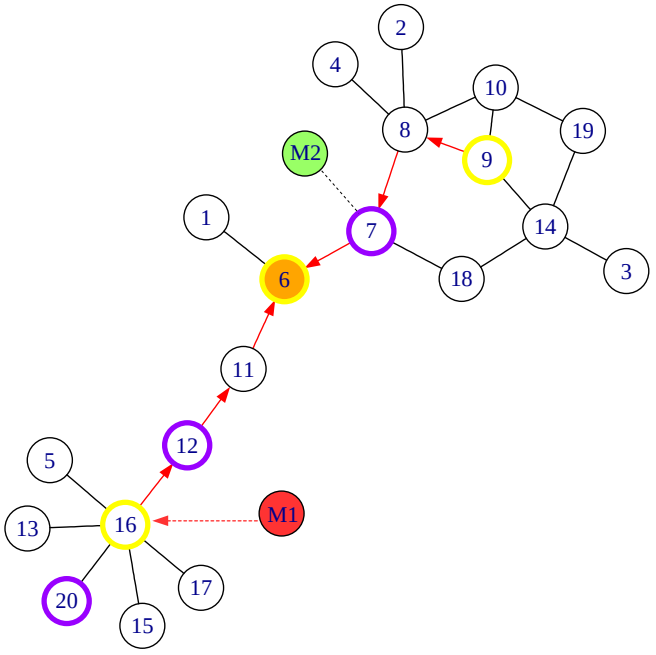


Figure 5.7: The tree from the AccessNodes of M1 to the DistributorNode 6.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x		x	x						
Node 2:	x	x	x	x						
Node 3:	x	x								
Node 4:	x	x	x	x						
Node 5:	x	x	x							
Node 6:	R_{M1}	x	x	$R_{11\ 7}$						
Node 7:	x	x	R_8	S_6						
Node 8:	x	R_9	S_7	x						
Node 9:	R_{M1}	S_8	x	x						
Node10:	x	x	x	x						
Node11:	x	x	R_{12}	S_6						
Node12:	x	R_{16}	S_{11}	x						
Node13:	x	x	x							
Node14:	x	x	x	x						
Node15:	x	x	x							
Node16:	R_{M1}	S_{12}	x	x						
Node17:	x	x	x							
Node18:	x	x	x	x						
Node19:	x	x	x							
Node20:	x	x	x							
NodeM1:	$S_{6\ 9\ 16}$	x	x	x						
NodeM2:	x	x	x	x						

Table 5.9: The schedule for the tree from the AccessNodes of M1 to the DistributorNode 6.

The example starts with a time-critical tree from the mobile node M1 to the mobile node M2. The first step is to determine the DistributorNode and find the concast tree from the AccessNodes of M1 to the DistributorNode. The resulting tree is shown in Figure 5.7 and the schedule in Table 5.9.
Node 6 was chosen as the DistributorNode, because it has the smallest average distance to all other nodes. As there is only one shortest path from each AccessNode to the DistributorNode, all tree creation strategies produce the same tree.
The data will first be sent from the mobile node to the AccessNode currently in range (slot 0), and then from the AccessNodes to the DistributorNode. For the path from the AccessNodes to the DistributorNode, the specialized scheduling strategy from Section 4.5.3.2 was chosen. It can be seen that the transmissions from node 8 to 7 and from 12 to 11 are scheduled in the same slot (2 and 3), even though their resp. receivers are in each others interference range (this is normally not allowed according to Property 2), because only one of the branches of this tree will be active at any time.

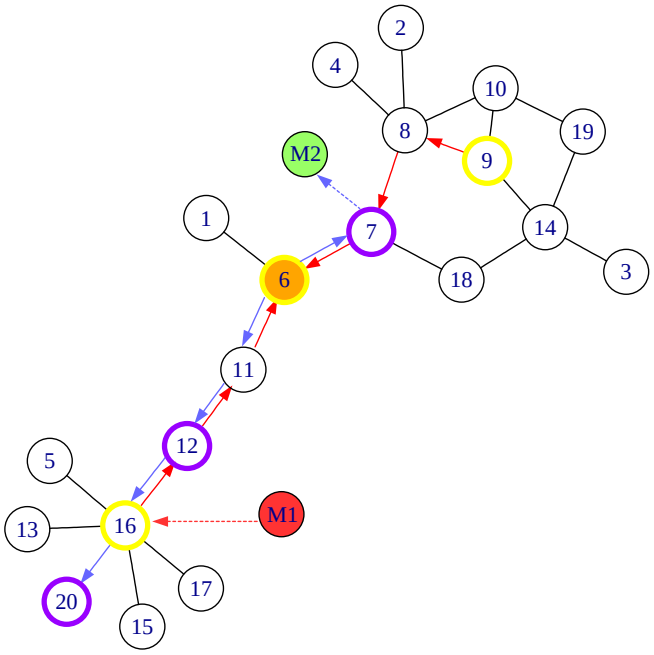


Figure 5.8: The tree from the DN to the AccessNodes of M2 has been added.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x		x	x	x	x			x	
Node 2:	x	x	x	x	x				x	
Node 3:	x	x							x	
Node 4:	x	x	x	x	x				x	
Node 5:	x	x	x			x	x	x	x	
Node 6:	R_{M1}	x	x	R_{117}	$S_{7\ 11}$	x	x		x	
Node 7:	x	x	R_8	S_6	R_6	x			S_{M2}	
Node 8:	x	R_9	S_7	x	x				x	
Node 9:	R_{M1}	S_8	x	x	x				x	
Node10:	x	x	x	x	x				x	
Node11:	x	x	R_{12}	S_6	R_6	S_{12}	x	x	x	
Node12:	x	R_{16}	S_{11}	x	x	R_{11}	S_{16}	x	S_{M2}	
Node13:	x	x	x			x	x	x	x	
Node14:	x	x	x	x	x				x	
Node15:	x	x	x			x	x	x	x	
Node16:	R_{M1}	S_{12}	x	x	x	x	R_{12}	S_{20}	x	
Node17:	x	x	x			x	x	x	x	
Node18:	x	x	x	x	x				x	
Node19:	x	x	x						x	
Node20:	x	x	x			x	x	R_{16}	S_{M2}	
NodeM1:	$S_{6\ 16\ 9}$	x	x	x	x	x	x	x	x	
NodeM2:	x	x	x	x	x	x	x	x	$R_{7\ 12\ 20}$	

Table 5.10: The schedule for the tree from the DistributorNode to the AccessNodes of M2.

The second step is to add the mobile node M2 as a destination of this transmission. This is achieved by creating a new tree from the DistributorNode 6 to all AccessNodes assigned to M2. Again, there is only one shortest path from the DistributorNode to each of the AccessNodes of M2, so all strategies produce the same tree. As this is a time-critical tree, *assignSlotsDelay* will be used for the scheduling and the scheduling starts in slot 4 (the slot after the last slot in which the DistributorNode receives from the AccessNodes of M1) to minimize the delay from M1 to M2.
The resulting tree and schedule are shown in Figure 5.8 and Table 5.10. Local multicasting is used in slot 4 from node 6 to nodes 7 and 11. The rest of the links are all scheduled directly after their preceding link as discussed in Section 4.5.2.1. Slot 8 is reserved to send from the AccessNodes to M2 and therefore blocked in all other nodes (as discussed in Section 4.5.3.2).
This completes the tree from mobile node M1 to mobile node M2. The maximal delay of a transmission from M1 to M2 is 9 micro slots. In Sections 5.4 and 5.5 it was shown how the trees and schedules involving mobiles nodes would look like without AccessNodes or the specialized scheduling strategy for the tree from AccessNodes to DistributorNode. It is obvious, that without these two improvements, no schedule would be found in this example.

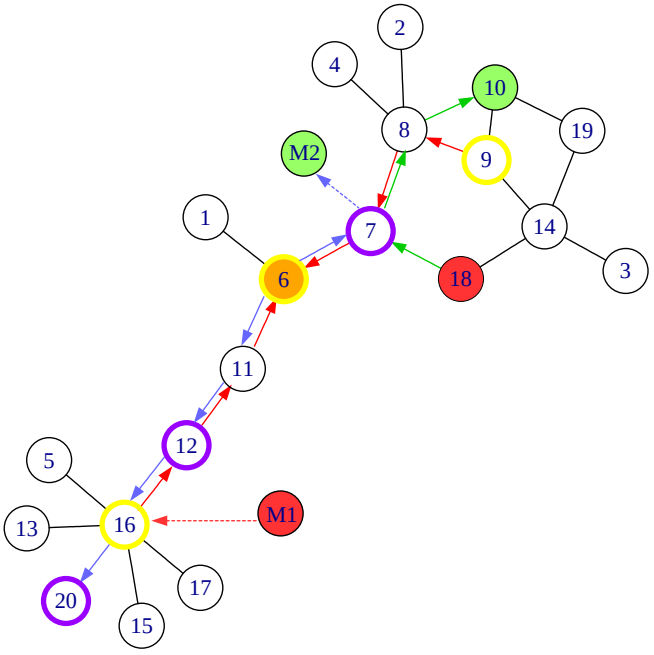


Figure 5.9: A stationary tree from node 18 to 10 has been added to the network.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x		x	x	x	x	x	x	x	
Node 2:	x	x	x	x	x	x	x	x	x	
Node 3:	x	x					x		x	
Node 4:	x	x	x	x	x	x	x	x	x	
Node 5:	x	x	x			x	x	x	x	
Node 6:	R_{M1}	x	x	R_{117}	$S_{7\ 11}$	x	x	x	x	
Node 7:	x	x	R_8	S_6	R_6	x	R_{18}	S_8	S_{M2}	
Node 8:	x	R_9	S_7	x	x	S_{10}	x	R_7	x	
Node 9:	R_{M1}	S_8	x	x	x	x	x	x	x	
Node10:	x	x	x	x	x	R_8	x	x	x	
Node11:	x	x	R_{12}	S_6	R_6	S_{12}	x	x	x	
Node12:	x	R_{16}	S_{11}	x	x	R_{11}	S_{16}	x	S_{M2}	
Node13:	x	x	x			x	x	x	x	
Node14:	x	x	x	x	x	x	x	x	x	
Node15:	x	x	x			x	x	x	x	
Node16:	R_{M1}	S_{12}	x	x	x	x	R_{12}	S_{20}	x	
Node17:	x	x	x			x	x	x	x	
Node18:	x	x	x	x	x	x	S_7	x	x	
Node19:	x	x	x			x	x	x	x	
Node20:	x	x	x			x	x	R_{16}	S_{M2}	
NodeM1:	$S_{6\ 16\ 9}$	x	x	x	x	x	x	x	x	
NodeM2:	x	x	x	x	x	x	x	x	$R_{7\ 12\ 20}$	

Table 5.11: The updated scheduled after adding the tree from 18 to 10.

The example is continued by adding a stationary tree from node 18 to 10 to the network (see Figure 5.9 and Table 5.11). As there are multiple shortest paths from 18 to 10, several algorithms for the tree creation can be used, as discussed in Chapter 3. One option examines the degree of the nodes on the paths. Here, nodes 7 and 8 have a higher degree than the nodes on the other paths (nodes 14 and 9 resp. 19), i.e., *createNewTreeDegree* would choose the shown path. The path chosen by *createNewTree* depends on the threshold set for the *getMinNumberOfFreeSlots* function. Table 5.10 shows that link [7,8] has three free slots left, while all other links on the possible paths have four free slots left. Therefore, if the threshold is three or less, the path shown in the figure would be used. Otherwise, one of the other paths would be chosen. Section 3.5.1 explained that a threshold equal to the length of the path is a safe option. Therefore, the path shown in green in Figure 5.9 was chosen in this example.

As there are not many free slots left, *assignSlotsUtil* is used. The effect that choosing *assingSlotsDelay* for this path has on the rest of the schedule will be examined in Section 5.6.

Links [18,7] and [7,8] are scheduled first and second, because they have less free slots (3) than link [8,10]. For link [18,7], slot 6 is chosen because it is blocked in more interference neighbors than slots 7 and 9 (e.g., it is blocked in node 6 which is free in slots 7 and 9) and free in the same number of links on the route as the

other slots. For link [7,8], slot 7 is chosen for similar reasons. Finally, link [10,8] is scheduled for slot 5, because it is blocked in more interference neighbors than slot 9. This leads to a delay of 10 micro slots for this tree. The resulting tree and schedule is shown in Figure 5.9 and Table 5.11.

In the last step, four destinations (5, 13, 15 and 17) are added to the first tree (from node M1 to M2), i.e., they want to receive the transmission from M1. The resulting tree and schedule is shown in Figure 5.10 and Table 5.12. Since, the destinations are added to an existing tree, local multicasting can be used. All destinations are in one-hop distance to node 16, which is already scheduled to send to node 20 in this tree.

Therefore, the first two destinations (5 and 13) are scheduled to receive in the slot in which node 16 sends (slot 7). Now, node 16 is transmitting to 3 other nodes, which means, according to the assumptions (Section 2.2.2), that another slot has to be chosen for the rest of the destinations. The only slot that is still free in node 16 (and the next destination 15) is slot 9, so node 16 is scheduled to send in slot 9 and node 15 is scheduled to receive in the same slot. The next destination (17) is also scheduled to receive in this slot (again using local multicasting).

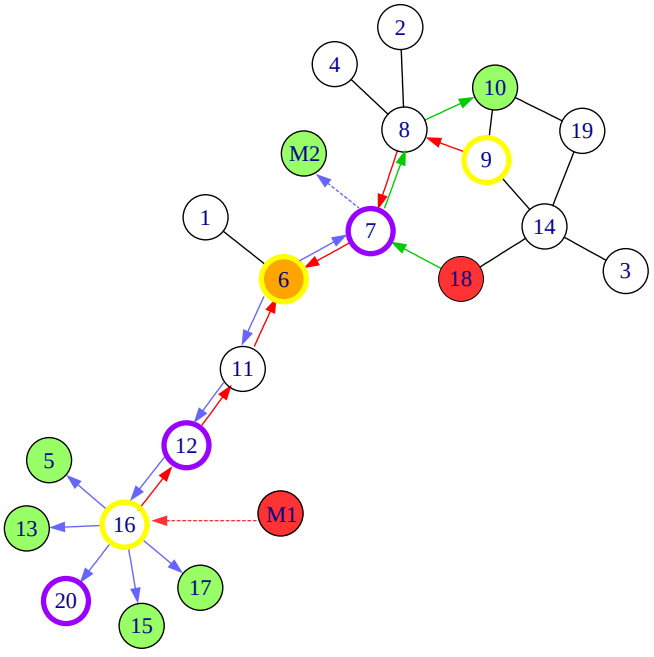


Figure 5.10: Four new destinations (5, 13, 15, 17) have been added to the tree from M1 to M2.

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x		x	x	x	x	x	x	x	
Node 2:	x	x	x	x	x	x	x	x	x	
Node 3:	x	x					x		x	
Node 4:	x	x	x	x	x	x	x	x	x	
Node 5:	x	x	x			x	x	R_{16}	x	x
Node 6:	R_{M1}	x	x	R_{117}	$S_{7\ 11}$	x	x	x	x	
Node 7:	x	x	R_8	S_6	R_6	x	R_{18}	S_8	S_{M2}	
Node 8:	x	R_9	S_7	x	x	S_{10}	x	R_7	x	
Node 9:	R_{M1}	S_8	x	x	x	x	x	x	x	
Node10:	x	x	x	x	x	R_8	x	x	x	
Node11:	x	x	R_{12}	S_6	R_6	S_{12}	x	x	x	x
Node12:	x	R_{16}	S_{11}	x	x	R_{11}	S_{16}	x	S_{M2}	x
Node13:	x	x	x			x	x	R_{16}	x	x
Node14:	x	x	x	x	x	x	x	x	x	
Node15:	x	x	x			x	x	x	x	R_{16}
Node16:	R_{M1}	S_{12}	x	x	x	x	R_{12}	$S_{20\ 5\ 13}$	x	$S_{15\ 17}$
Node17:	x	x	x			x	x	x	x	R_{16}
Node18:	x	x	x	x	x	x	S_7	x	x	
Node19:	x	x	x			x	x	x	x	
Node20:	x	x	x			x	x	R_{16}	S_{M2}	x
NodeM1:	$S_{6\ 16\ 9}$	x	x	x	x	x	x	x	x	x
NodeM2:	x	x	x	x	x	x	x	x	$R_{7\ 12\ 20}$	x

Table 5.12: The updated schedule after adding the new destinations to the tree from M1 to M2.

A Slight Variation of this Example

Slot:	0	1	2	3	4	5	6	7	8	9
Node 1:	x		x	x	x	x	x	x	x	
Node 2:	x	x	x	x	x		x	x	x	x
Node 3:	x	x					x		x	
Node 4:	x	x	x	x	x		x	x	x	x
Node 5:	x	x	x			x	x	R_{16}	x	x
Node 6:	R_{M1}	x	x	R_{117}	$S_{7\ 11}$	x	x	x	x	x
Node 7:	x	x	R_8	S_6	R_6	x	R_{18}	S_8	S_{M2}	x
Node 8:	x	R_9	S_7	x	x		x	R_7	x	S_{10}
Node 9:	R_{M1}	S_8	x	x	x		x	x	x	x
Node10:	x	x	x	x	x		x	x	x	R_8
Node11:	x	x	R_{12}	S_6	R_6	S_{12}	x	x	x	x
Node12:	x	R_{16}	S_{11}	x	x	R_{11}	S_{16}	x	S_{M2}	x
Node13:	x	x	x			x	x	R_{16}	x	x
Node14:	x	x	x	x	x		x	x	x	x
Node15:	x	x	x			x	x	x	x	R_{16}
Node16:	R_{M1}	S_{12}	x	x	x	x	R_{12}	$S_{20\ 5\ 13}$	x	$S_{15\ 17}$
Node17:	x	x	x			x	x	x	x	R_{16}
Node18:	x	x	x	x	x		S_7	x	x	x
Node19:	x	x	x				x	x	x	x
Node20:	x	x	x			x	x	R_{16}	S_{M2}	x
NodeM1:	$S_{6\ 16\ 9}$	x	x	x	x	x	x	x	x	
NodeM2:	x	x	x	x	x	x	x	x	$R_{7\ 12\ 20}$	

Table 5.13: The schedule for the same network when the tree from 18 to 10 was scheduled with *assignSlotsDelay*.

In the previous example, a new tree from 18 to 10 was scheduled with the *assingSlotsUtil* function. To show the difference between the scheduling strategies, the same example is now examined with the tree from 18 to 10 scheduled with *assignSlotsDelay*. The resulting schedule is shown in Table 5.13. The first two links of the tree ([18,7] and [7,8]) are scheduled in the same slots as in the previous example, but the last link from 8 to 10 is scheduled for slot 9 in this case. This leads to a decreased delay for this tree (from 10 microslots to 4 microslots). Therefore, if no other routes have to be added, this schedule is better. However, the previous example has more free slots left. For example, as seen in Table 5.12 it would be possible to add a new tree from node 1 to node 6 in slot 9, or add node 6 to the tree from node 18 to 10 (also slot 9). In the modified example (Table 5.13), node 6 is blocked in all slots, therefore these possibilities no longer exist. This shows the differences between the scheduling strategies and the reason why *assignSlotsUtil* generally finds a feasible schedule in more scenarios, as shown in Section 5.3.

Chapter 6

Conclusion and Future Work

In this thesis, a routing algorithm for the scenario described in Chapters 1 and 2 was developed. The original scenario is a clustered, TDMA-based network containing a topology of stationary nodes and a few mobile nodes. The algorithm is able to discover routes between different types of nodes and creates a conflict-free time slot schedule. Several different functions for the route discovery and scheduling phase were developed and discussed. To increase the number of application scenarios in which the algorithm can be used, the possibility to either minimize the delay or maximize the chance of finding a route for the current and future route requests is given.

The requirement to find routes from one source to multiple destinations is satisfied. Multicast trees are created in the route discovery phase to avoid redundant transmissions and local multicasting is used in the scheduling algorithms to save bandwidth and further reduce the delay.

The algorithm can handle mobile nodes as source and destinations of trees and offers a specialized scheduling strategy to find an efficient schedule without wasted time slots if the mobile node acts as source of a tree. To include mobile nodes a new node type has been introduced. Stationary AccessNodes have to be specified by the user and ideas to determine suitable AccessNodes were given. In the given application context, with its fixed stationary topology, this is not a problem and only requires a bit of additional work.

The algorithm works on a relatively realistic network model, with the interference range of nodes taken into account. Possible packet loss is handled by an acknowledgement mechanism. Other requirements, such as the synchronization and global knowledge of the stationary network can be fulfilled by previous work [13, 18].

The routing and scheduling of management traffic was also discussed and a solution was presented.

In the original scenario, the network was clustered and had several different clustering and functional node types. The algorithm works without problems for this network type, even though the clustering and functional node types are not necessary for the algorithm.

As a proof of concept, the route discovery and scheduling phase of the algorithm were simulated and the simulation confirmed the effectiveness of the created routes and schedules. An actual implementation of the algorithm on wireless nodes was

not part of the thesis.

For future work, several possible optimizations for the algorithm were discussed in the main part of this thesis. For example, in Section 3.4 it was discussed how the current route confirm phase can be optimized for energy constrained networks.

In addition to these, several possibilities to further improve the algorithm exist.

Currently, the algorithm is designed to fulfill route requests with one micro slot per link and super slot. Due to the configurable length of the micro slots, this is enough for many scenarios. The length can be adjusted, so that all packets and their acknowledgements fit into a single slot. A generalization of the algorithm, so that a variable number of slots can be reserved, would be an improvement and should be possible without changing the basic principles.

A variable number of slot reservations would also allow the inclusion of multipathing, i.e., splitting data into parts and sending them along different paths. In some network topologies, this can lead to an improvement in throughput and increase the chance of finding a feasible route in bandwidth constrained networks. However, research suggests that multipathing generally does not lead to a significant improvement in end-to-end delay and network capacity in general networks [23].

Currently, the algorithm selects a single route and schedule for a route request according to the criteria of the used functions. A possibility to further increase the success rate of the scheduling could be the use of alternative paths and schedules. So, instead of selecting a single route / schedule, several alternatives could be saved. If at some point no route or schedule for a new request is found, this would also allow to try different combinations of solutions for previous requests, in order to fulfill the current request. An additional challenge of such an approach is to decide which and how many alternatives should be tried, before the current request is deemed impossible to fulfill.

A similar, but simpler, idea would be to redo the route discovery and scheduling process at specified points of time, for example, after the number of available slots is reduced below some threshold. The routes and schedules currently depend on the order in which they were created. E.g., if a destination is added to an earlier tree, after several other trees were created, the resulting schedule may not be very efficient. So, in that case redoing the whole process in a different order (e.g finishing the route discovery and scheduling for one tree, before creating a new tree), could improve the result.

Bibliography

[1] igraph R package. `http://igraph.org/r/`. Accessed: 15.03.2015.

[2] Latex. `http://www.latex-project.org`. Accessed: 15.03.2015.

[3] R. `http://www.r-project.org/`. Accessed: 15.03.2015.

[4] Jamal N Al-Karaki and Ahmed E Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless communications, IEEE*, 11(6):6–28, 2004.

[5] Shigang Chen and Klara Nahrstedt. Distributed quality-of-service routing in ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1488–1505, 1999.

[6] Shigang Chen, Klara Nahrstedt, and Yuval Shavitt. A qos-aware multicast routing protocol. In *INFOCOM*, pages 1594–1603, 2000.

[7] Yuh-Shyan Chen, Tzung-Shi Chen, and Ching-Jang Huang. Som: Spiral-fat-tree-based on-demand multicast protocol in a wireless ad-hoc network. In *ICOIN*, pages 17–24, 2001.

[8] Yuh-Shyan Chen, Yun-Wen Ko, and Ting-Lung Lin. A lantern-tree-based qos multicast protocol for wireless ad-hoc networks. In Ronald P. Luijten, Eric Wong, Kia Makki, and E. K. Park, editors, *ICCCN*, pages 242–247. IEEE, 2002.

[9] Yuh-Shyan Chen, Tsung-Hung Lin, and Yun-Wei Lin. A hexagonal-tree tdma-based qos multicasting protocol for wireless mobile ad hoc networks. *Telecommunication Systems*, 35(1-2):1–20, 2007.

[10] Imrich Chlamtac, Marco Conti, and Jennifer J-N Liu. Mobile ad hoc networking: imperatives and challenges. *Ad hoc networks*, 1(1):13–64, 2003.

[11] J. J. Garcia-Luna-Aceves and Ewerton L. Madruga. The core-assisted mesh protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1380–1394, 1999.

[12] Johann Gebhardt. TDMA-based Multicast-QoS-Routing-Approaches for Mobile Ad Hoc Networks. Seminar thesis, TU Kaiserslautern, 2014.

[13] Reinhard Gotzhein and Thomas Kuhn. Decentralized tick synchronization for multi-hop medium slotting in wireless ad hoc networks using black bursts. In *SECON*, pages 422–431. IEEE, 2008.

[14] Anuschka Igel and Reinhard Gotzhein. An analysis of the interference problem in wireless tdma networks. In *ICWMC 2012, The Eighth International Conference on Wireless and Mobile Communications*, pages 187–194, 2012.

[15] Kamal Jain, Jitendra Padhye, Venkata N Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. *Wireless networks*, 11(4):471–487, 2005.

[16] Xiaohua Jia. A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks. *IEEE/ACM Transactions on Networking (TON)*, 6(6):828–837, 1998.

[17] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth Computer Science, July 2003.

[18] Christopher Kramer. Ermittlung des Netzzustands von Funk-Netzwerken. Technical report, TU Kaiserslautern, 2013.

[19] Christopher Kramer. Drahtlose Kommunikationssysteme für den Produktionsbereich. Master's thesis, TU Kaiserslautern, 2014.

[20] Baochun Li. Qos-aware adaptive services in mobile ad-hoc networks. In *Quality of Service - IWQoS 2001*, pages 251–265. Springer, 2001.

[21] Chunhung Richard Lin. On-demand qos routing in multihop mobile networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1735–1744. IEEE, 2001.

[22] Chunhung Richard Lin and Jain-Shing Liu. Qos routing in ad hoc wireless networks. *IEEE J.Sel. A. Commun.*, 17(8):1426–1438, September 2006.

[23] Mattias Nissler and Reinhard Gotzhein. Performance evaluation of multi-path routing in reservation-based wireless networks. In *Proceedings of the 12th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, pages 268–273. ACM, 2009.

[24] S Ramanathan and Martha Steenstrup. A survey of routing techniques for mobile communications networks. *Mobile Networks and Applications*, 1(2):89–104, 1996.

[25] E. Royer and C. Perkins. Multicast Ad hoc On- Demand Distance Vector (MAODV) Routing, 2000.

[26] Jochen Schiller. *Mobile Communications*. Addison-Wesley, Boston, second edition, May 2003.

[27] Jian Shen, Wenying Zheng, Jin Wang, Zhihua Xia, and Zhangjie Fu. Routing protocols using directional antennas in ad hoc networks: A comparative review. *International Journal of Grid & Distributed Computing*, 6(5), 2013.

[28] Kuei-Ping Shih, Chih-Yung Chang, Yen-Da Chen, and Tsung-Han Chuang. Dynamic bandwidth allocation for qos routing on tdma-based mobile ad hoc networks. *Computer Communications*, 29(9):1316 – 1329, 2006. {ICON} 2004 12th {IEEE} International Conference on Network 2004.

[29] Kannan Srinivasan and Philip Levis. Rssi is under appreciated. In *In Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets*, 2006.

[30] Jian Tang, Guoliang Xue, and Christopher Chandler. Interference-aware routing and bandwidth allocation for qos provisioning in multihop wireless networks. *Wireless Communications and Mobile Computing*, 5(8):933–943, 2005.

[31] Xu Zhen and Zhou Long. Bandwidth constrained multicast routing for tdma-based mobile ad hoc networks. *Journal of Communications Systems*, 8(3):161–167, 2013.