Strategies for Increasing Maximum Throughput and Reducing Latency in Tree-Based WSNs

Von der Fakultät 1 - MINT - Mathematik, Informatik, Physik, Elektro- und Informationstechnik der Brandenburgischen Technischen Universität Cottbus-Senftenberg genehmigte Dissertation zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von

M.Sc. Aleksandar Ilić

geboren am 7. Mai 1989 in Niš (Jugoslawien)

Vorsitzender: Dr.-Ing. Marc Reichenbach

Gutachter: Prof. Dr. Peter Langendörfer

Gutachter: Prof. Dr.-Ing. Jörg Nolte

Gutachter: Prof. Dr. Zoran Prijić

Tag der mündlichen Prüfung: 03.05.2023

DOI: https://doi.org/10.26127/BTUOpen-6498

Zusammenfassung

Diese Promotion befasst sich mit Strategien zur Erzielung eines hohen Datendurchsatzes in drahtlosen Sensornetzwerken, die ein Zeitmultiplexverfahren (TDMA) verwenden, um den Medienzugriff zu regeln. Die Arbeit verwendet einen komplexen Ansatz, der sich nicht nur mit dem Scheduling-Algorithmus, sondern auch mit der Netzwerkschicht und dem Interferenzmodell befasst. Die Arbeit schlägt vier Lösungen vor, die den Datendurchsatz, die Fairness und die Latenz im betrachteten Szenario signifikant verbessern.

Die Promotion beginnt mit einem Überblick über aktuelle MAC-Protokolle (Medium Access Control) mit Schwerpunkt auf TDMA. Basierend auf dieser Übersicht wird der Schluss gezogen, dass im Bereich der Berechnungen durch Schedule Algorithmen nicht viel Raum für Verbesserungen verbleibt. Viele solcher Algorithmen werden bis heute vorgeschlagen, und sie können Schedule-Längen nahe dem theoretischen Minimum erreichen. Die Forschung zeigt jedoch auch einen Mangel an detaillierter Bewertung und Vergleich dieser Algorithmen; dies macht die Auswahl des am besten geeigneten Algorithmus für eine bestimmte Anwendung schwierig und die Leistungsschätzung ungenau. Daher wurde eine umfassende Bewertung von TDMA-Protokollen nach dem Stand der Technik unter Verwendung von Simulationen und über 200 zufällig generierten Netzwerken durchgeführt, um dieses Problem anzugehen. Die Ergebnisse ermöglichen die Auswahl eines geeigneten Algorithmus und die Abschätzung der Leistung für jede spezifische Anwendung.

Als nächstes wird das Problem mehrerer Paketübertragungen während eines einzelnen Zeitschlitzes analysiert. TDMA-Protokolle nach dem Stand der Technik gehen davon aus, dass in jedem Schlitz ein Paket übertragen werden kann, und optimieren die Anzahl von Schlitzen, die jeder Knoten unter dieser Annahme erhält. Wenn Knoten jedoch mehr als ein Paket übertragen können, wird die Leistung eines solchen Zeitplans beeinträchtigt. Um dies zu lösen, wird das M-TreeMAC-Protokoll vorgeschlagen. Dieses Protokoll berücksichtigt die tatsächliche Anzahl der in einem Zeitschlitz übertragenen Pakete und optimiert den Zeitplan entsprechend. Darüber hinaus wird beobachtet, dass die Routing-Topologie die mit diesem Algorithmus erstellte Zeitplanlänge stark beeinflusst. Ein Algorithmus, der die Topologie optimiert, um bei Verwendung von M-TreeMAC den kürzesten Zeitplan zu erzielen, wird vorgeschlagen, wodurch die Vorteile noch weiter gesteigert werden.

Schließlich wird die Genauigkeit des 2-Hop-Interferenzmodells, das üblicherweise von modernen TDMA-Scheduling-Algorithmen verwendet wird, untersucht und unter Verwendung eines realistischen Funkmodells basierend auf Messergebnissen simuliert. Die Ergebnisse zeigen hohe Paketverlustquoten für Pakete, die eine große Anzahl von

Sprüngen durchlaufen, um die Senke zu erreichen. Das adaptive Interferenzmodell wird vorgeschlagen, um das 2-Hop-Interferenzmodell zu verbessern. Das vorgeschlagene Modell kann den Durchsatz in Netzwerken mit zehn oder mehr Hops erheblich steigern.

Abstract

This thesis deals with strategies for achieving high data throughput in wireless sensor networks that use a time division multiple access (TDMA) scheme to resolve medium access. The thesis uses a multi-sided approach that deals not only with the scheduling algorithm but also with the network layer and the interference model. The thesis proposes four solutions that significantly improve data throughput, fairness, and latency in the considered scenario.

The thesis starts with an overview of state-of-the-art medium access control (MAC) protocols, emphasizing TDMA. Based on this overview, it is concluded that not much space for improvement is left in the field of schedule calculation algorithms; many such algorithms are proposed up to date, and they can achieve schedule lengths close to the theoretical minimum. However, the research also reveals a lack of in-detail evaluation and comparison of these algorithms; this makes choosing the most suitable algorithm for a particular application hard and performance estimation inaccurate. Therefore, an extensive evaluation of state-of-the-art TDMA protocols using simulations and over 200 randomly generated networks was performed to tackle this issue. The results allow choosing an appropriate algorithm and estimating performance for each specific application.

Next, the problem of multiple packet transmissions during a single time slot is analyzed. State-of-the-art TDMA protocols assume that one packet can be transmitted in each slot and optimize the number of slots each node gets under this assumption. However, when nodes can transmit more than one packet, the performance of such a schedule is impaired. To solve this, the M-TreeMAC protocol is proposed; this protocol considers the actual number of packets transmitted in a time slot and optimizes the schedule accordingly. Furthermore, it is observed that the routing topology heavily impacts the schedule length created using this algorithm; an algorithm that optimizes the topology to result in the shortest schedule when M-TreeMAC is used is proposed, increasing benefits even further.

Finally, the accuracy of the 2-hop interference model, commonly used by state-of-the-art TDMA scheduling algorithms, is studied and simulated using a realistic radio model based on measurement results. The results show high packet loss ratios for packets traveling a large number of hops to reach the sink. The adaptive interference model is proposed to improve the 2-hop interference model. The proposed model can increase throughput significantly in networks with a height of ten or more hops.

Contents

1 Introduction

A Wireless sensor network (WSN) is a network of radio-enabled sensor nodes. Sensor nodes gather data by measuring and monitoring one or more physical quantities in their environment. The data collected by the sensors is then delivered to the destination nodes. A destination node may be a node from which a user may use this data or a node that can take action based on this data. In modern sensor networks, the destination node is often a node connected to the internet, through which the data is transferred to a server to be processed further; the connection to the internet is commonly achieved wirelessly, using a GSM module.

With the increased technical development in recent years, the cost of radio modules and microprocessors used to control the radio and sensors in a node has decreased. At the same time, faster and more energy-efficient processors became available. These developments increased the scope of wireless sensor network applications significantly. Moreover, lower device cost has brought them to the fields for which they were too expensive in the past. Intelligent home systems are one example; they include sensor nodes in various home appliances and furniture [1]. In addition, reduced costs also allowed for having wireless sensor nodes in personal gadgets (smart watches, sleep monitoring devices) and even on pets (tracing pet's location, pulse, body temperature [2]).

Lower device cost also affects the usage of wireless sensor networks in the fields in which they were present traditionally. Traffic monitoring systems used to rely mainly on wired sensors in the past. However, wireless solutions are being applied in this field more frequently. Furthermore, reduced cost allows for wireless sensor networks in less critical applications in this field, such as traffic data gathering for research. Another example is the use of agricultural monitoring networks, which are becoming available to smallholdings.

Protocol development has been focused on high-performance protocols in recent years. Unfortunately, such protocols require powerful and high-cost devices; at the same time, the energy consumption increases. Examples of such high-cost protocols are time slotted channel hopping protocols (TSCH) [3–5] or complex and extensive industrial protocol standards (ISA 100.11a [6] and WirelessHART [7]). However, with the increased application scope of WSNs, there is a rising requirement for effective and fast communication protocols, which are, at the same time, simple enough to be implemented on low-cost hardware.

This thesis studies the problem of achieving maximal possible throughput using low-cost and energy-efficient hardware. The motivation arose while designing a wireless sensor network for a traffic monitoring system using 16-bit MSP430 series microcon-

trollers. The network was powered using solar cells, putting additional limits on power consumption. As existing complex and energy-hungry solutions (like ZigBee [8], IEEE 802.15.4 [9], and Bluetooth low energy [10]) were not an option, in this case, the research on what can be achieved with available limitations was conducted. The products of this research are improvements, models, and protocols proposed in this thesis.

This thesis studies the problem of designing a protocol stack for a convergecast wireless sensor network; the goal is to achieve the highest possible throughput using low-cost devices. A convergecast sensor network has one sink node, which collects all data produced by other sensor nodes. In modern sensor networks, data gathered is commonly transferred to the user using a connection to the internet. Often only one node will have internet access, reducing the costs. Therefore, all data will be collected at that node, meaning that such a network is a convergecast network. Widespread usage of such networks justifies the focus of this thesis on this kind of network.

The first step in solving this problem is an extensive study of state-of-the-art link- and network-layer protocols. The most relevant protocols are revised in detail in section 2.4 of this thesis. Based on this study, it is concluded that the link-layer protocol is the crucial element in achieving high data throughput. More specifically, its medium access controls (MAC) part. This part has the role of organizing the nodes to communicate without their packets interfering with each other. The maximal throughput that can be achieved depends on how well the interference is avoided and how many nodes can transmit simultaneously. This thesis studies these problems in detail and proposes improvements in both fields.

The detailed study of state-of-the-art MAC protocols covers three types of such protocols, each having a very different approach. The applicability of protocols of each type in the studied problem is considered. However, the analysis identifies time division multiple access protocols as the irreplaceable choice for the considered scenario. Other options are low-duty cycle protocols, which heavily sacrifice throughput to benefit energy consumption, and random access protocols, which experience throughput drop at high packet generation rate values. Therefore, the TDMA is selected as the approach of choice. Next, it is studied how to design and optimize a TDMA MAC algorithm to maximize the throughput.

A TDMA MAC protocol avoids collisions by allocating one or more time windows (called a time slot) to each node; a node may only transmit during one of the allocated slots. A collection of time slots is called the schedule. To achieve the highest possible throughput, the scheduling algorithm should find the shortest possible schedule that assigns the appropriate number of time slots to each node. This problem is proven to be NP-Complete [11]. Therefore, finding the optimal solution in a polynomial time is

impossible. Hence, approximative methods should be used to solve this problem.

As this thesis aims to maximize the throughput, the first efforts were to improve the existing scheduling algorithms. However, the literature analysis reveals this to be a well-studied problem. Therefore, the focus of the research is put into maximizing throughput while using the existing scheduling algorithms. The thesis proposes four different improvements, allowing to achieve this. The first is a guideline for selecting the most suitable scheduling algorithm based on the network properties. The second and third improvements consider multiple packet transmissions during a single time slot, a fairly common scenario in WSNs. Scheduling and network topology optimization algorithms are proposed to improve performance under these conditions. The scheduling algorithm is based on one of the existing approaches; it differs in how the number of slots each node gets is calculated. This number is optimized for multiple packet transmissions. The fourth improvement is a novel interference model, the adaptive interference model. This model reduces interference, increasing throughput subsequently.

As mentioned before, many scheduling algorithms for optimizing throughput when TDMA MAC protocols are used are proposed up to date. However, detailed comparison and evaluation of these protocols are lacking. Different schedule calculation algorithms are characterized by the lowest bond of schedule length the algorithm can achieve. These values are similar for many protocols. Furthermore, this value only gives information about the best possible performance, but not on how often it is achieved and how much actual performance will differ on average. Based on all this, the choices approach is to define a guideline for selecting the most appropriate protocols for a given application, rather than designing another algorithm similar to many existing ones.

Section 3.1 provides an overview of the most effective state-of-the-art schedule calculation algorithms. To create a guideline for choosing one of them, they were implemented in the OMNeT++ simulator. Their performance was simulated using more than 200 different randomly generated sensor networks. Based on these simulations, a guideline is provided to pick the protocols that deliver the best performance for the given application. The results lead to the classification of these algorithms in two groups: the trade-off group and the high-performance group. Protocols of the first group are simple to implement and offer a very fast network setup. The trade-off is lower throughput compared with the high-performance group; the offered throughput value is still considerably high. Results show that in the case of narrow networks (where there are not many parallel scheduling possibilities), trade-off protocols perform almost as well as the high-performance ones. The results show that the protocol by Park *et. all.* [12] has a slight advantage over the other high-performance protocols for larger networks and that Lai's [13] algorithms performed the best for networks with smaller sizes.

Section 4 further research possibilities to increase throughput in convergecast sensor networks. It considers multiple packet transmissions during a single time slot. Namely, slot duration and packet transmission time can rarely be equal. Furthermore, time slot duration is often a few larger than packet transmission time, allowing for a couple of packets to be transmitted in a time slot. State-of-the-art schedule calculation algorithms do not consider this, resulting in a schedule that is not optimized for the traffic and impaired performance. This section proposes a scheduling algorithm that considers multiple transmissions when calculating the schedule. Furthermore, it is shown that the length of such a schedule strongly depends on the routing topology, and an algorithm for optimizing this topology is proposed; the author published this algorithm in [14].

Multiple transmissions during a single time slot change the total number of slots each node requires. Not considering this leads to not only an unnecessarily longer schedule but also to a lower data delivery fairness in the network. This thesis proposes a novel scheduling algorithm, M-TreeMAC, which solves this issue. The proposed algorithm belongs to the group of trade-off algorithms; it is inspired by one algorithm from this group, TreeMAC. The proposed algorithm significantly improves over TreeMAC, as shown using extensive simulations presented in Section 4.2. In the case of linear networks, improvement is significant in terms of both throughput and latency. However, in general networks, the improvement is only significant in reducing packet latency; throughput remains similar due to complex traffic flow in such networks.

In Section 4.3 a topology optimization that further improves M-TreeMAC is proposed. Namely, unlike TreeMAC, the schedule length of M-TreeMAC depends heavily on the routing topology used. In this section, this dependency is explained, and an optimization criterion is formally defined. Based on this criterion, an algorithm that finds the optimal topology is proposed. Finally, the proposed algorithm is implemented in the OMNeT++ simulator and evaluated using an extensive set of randomly generated networks. The results show that the proposed algorithm can reduce the schedule length by as much as 30%.

Finally, in Section 5, the fourth strategy for increasing throughput is considered. This time, the interference model is improved rather than the schedule calculation algorithm. Namely, when a schedule is calculated, the scheduler tries to schedule as many nodes as possible to share a slot. When this is done, an interference model is used to determine whether two nodes can share a time slot. The most commonly used model is the 2-hop interference model. This model is very simple, but it is not very accurate. Therefore, its usage leads to some amount of packet loss due to collisions, even though TDMA MAC is theoretically collisionless. Collisions can be completely avoided if more complex physical models are used. However, their usage requires additional hardware

modules and introduces additional complexity complexity as well as increased network setup time.

Section 5 proposes the adaptive interference model. This model provides a compromise between the 2-hop model and physical interference models. It offers significantly improved accuracy over the 2-hop model. At the same time, it keeps the simplicity of the 2-hop model, allowing its usage instead of the 2-hop model with no or little modifications to the other protocol components and layers. The adaptive interference model uses hop count to define interference range, the same way the 2-hop interference model does. However, instead of this range being fixed to 2-hops, it is adapted for each application scenario. The adaptation is performed using a simple algorithm, which considers network configuration, radio properties, and radio environment. The author previously published a simpler version of this algorithm in [15]; this thesis proposes an improved version.

2 Background and Related Work

This chapter introduces wireless sensor networks and the problems and challenges associated with their design and implementation. The emphasis is placed on the most relevant issues for the topic of this thesis, maximizing data throughput. The decisive factor in achieving high throughput is the physical layer; the radio's bit rate defines how long the transmission of each packet lasts. The physical layer is, however, not the topic of this thesis. On the contrary, this thesis deals with selecting, designing, and optimizing upper-layer protocols to achieve the highest possible throughput using the provided physical layer.

The crucial element of the protocol stack (excluding the physical layer) for achieving high throughput is Medium Access Control (MAC). The role of this layer is to ensure packet transmission without collisions (a collision occurs when a radio receives two packets simultaneously). There are various approaches to achieve this. These approaches can be divided into contention-based, contention-free, and low-duty cycle protocols. The first goal is selecting the correct protocol type for the application. The second one is optimizing it to suit the particular application's requirements. Section 2.4 describes these three protocol types and gives an overview of some state-of-the-art protocols from each group. Based on this overview, a group of protocols suitable for high-throughput applications is selected. Finally, the protocols of this group are simulated in Section 3.3, and guidelines for choosing the most suitable one for each application are provided. The metrics used for this evaluation are described in Section 2.3.

The second most important layer for high throughput is the network layer. This layer finds and maintains paths (routes) that data packets take to reach their destination node in the network. These routes can be chosen based on different criteria; one of them can be throughput maximization. Section 2.6.2 describes how these paths are formed in converge-cast sensor networks and which algorithms are commonly used in them. Section 2.6.3 explains the relation between the routing topology and data throughput in the network.

2.1 Wireless Sensor Networks

A wireless sensor network is a collection of sensor nodes capable of wireless communication. Each wireless sensor node consists of sensors, a microprocessor, a power source, and a radio module. A block diagram of a generalized sensor node is depicted in Fig. 1. The application running on the microcontroller controls the sensors and generates data packets based on measurements. These packets are then transmitted until they reach their destination node. The destination node is usually a node with a user interface or

connection to the internet, allowing a user or a data-gathering application to process the data.

Wireless sensor nodes are often designed to operate autonomously, without connection with a power source. They contain a battery and, optionally, a power source for that purpose. This power source is usually an energy harvesting module. It may be used to prolong battery life or allow indefinite battery life, making the sensor fully autonomous. Common energy harvesting modules use solar, wind, or thermal energy.

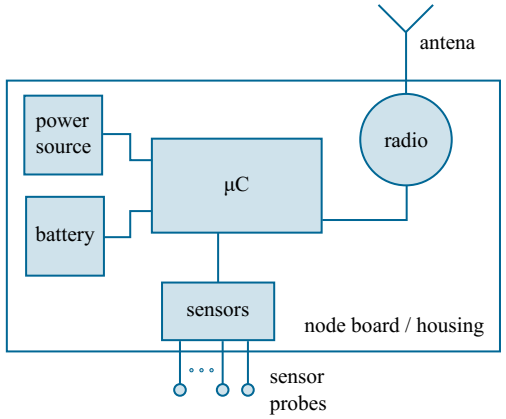


Figure 1: Block diagram of a wireless sensor node.

A wireless sensor network is comprised of multiple wireless sensor nodes. Each of them might have a different role. Role definitions depend on the protocol used for data transfer and the application. One common type of wireless sensor network is a converge-cast sensor network. Converge-cast networks have one or more sink nodes, which collect data from the whole network and transfer it to the user. The other nodes are called sensor nodes, and they generate data by performing measurements. Converge-cast sensor networks can be single-hop or multi-hop. In single-hop sensor networks, all sensor nodes are within the radio range of the network sink; when they have a data packet, nodes transmit it to the sink directly. In multi-hop sensor networks, nodes must use other nodes to forward data packets to the sink. Such networks are more elaborate and have a complex network layer.

More complex protocols, usually intended for multiple different applications and deployment scenarios, define a larger number of roles the nodes can take. For example, ISA 100.11A [6] standard defines the following roles: routing device, IO device, portable device, backbone router, gateway, system manager, and security manager. IEEE 802.15.4 [9] defines four different roles, Super PAN coordinator, PAN coordinator, Full functioning device (FFD), Reduced function Device (RDF). In this case, this is needed because of the unique network structure. The network using this protocol

is divided into clusters, each having a star topology with a PAN coordinator in the middle. Super PAN coordinator is predefined, and it initiates the network formation by sending beacons, which other nodes use to join the network. Then, the super PAN coordinator instructs some of the nodes to take to the role of the PAN coordinator and add additional nodes to the network in that way.

One multi-hop converge-cast sensor network is depicted in Fig. 2. This network has two sinks, seven sensor nodes (blue node with green sensor module), and two routing nodes. Solid lines represent links used for routing the packets, while dashed lines represent links between nodes that are not a part of the routing network. The protocol running in each node discovers these links and selects routes. Besides that, some other tasks of this protocol are to avoid collisions and ensure that packets have reached their destination. Because a protocol must accomplish various very different tasks, it is divided into layers. The collection of these layers is called a network stack. This type of organization allows easier implementation and adaptation of the protocol to a certain application. For example, this allows changing just one protocol layer while keeping the others.

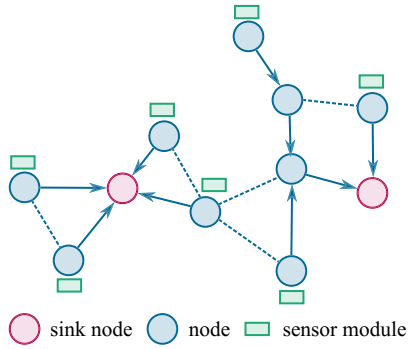


Figure 2: Example converge-cast multi-hop sensor network.

Wireless sensor networks have a wide range of applications. The design of the nodes and communication protocols used by them vary heavily based on the application. For example, a DC power supply is available in some applications, and sensors can be powered using an external power source. Low energy consumption is not a priority in such cases, and MAC protocol can be optimized for maximum throughput. A DC power supply is readily available in many application scenarios. For example, sensor nodes can be powered from the power line in sensor networks used for building management systems [16]. Or, in some traffic monitoring systems, sensor nodes can be powered from the network used for road lighting [17].

The main challenge in many other applications in energy efficiency since nodes are

powered either from a pre-charged battery or using an energy harvester. One such application example is medical body area networks [18–20]. These networks are used for measuring vital parameters from one or more patients. Commonly measured parameters are heart rate, temperature, blood oxygen saturation level, and electrocardiogram (ECG) signals. To achieve that, sensors are attached to the patient's body and connected with the monitoring device using a suitable wireless communication protocol. The number of sensors can be high; for example, precise EKG monitoring requires at least six probes. Therefore, it is inconvenient to power all the sensors with a wire. For these reasons, battery-powered sensors are used. This introduces the need to find a trade-off between Quality of Service (QoS) and energy efficiency. Higher QoS comes with incensed energy consumption, requiring a larger battery, which is heavier and more expensive.

Some application scenarios require many nodes scattered on a vast and hard-to-access space. For example, monitoring perma-frost requires positioning sensor nodes at very isolated places and high altitudes in the mountains [21]. On the other hand, forest fire monitoring networks require coverage of vast forest areas with a large amount of nodes [22–24]. Locating and accessing nodes in such a network is very demanding and less cost-efficient than deploying a new sensor network. Therefore, energy-efficient protocols that sacrifice the quality of service are needed to meet these requirements. This can be achieved using very low duty cycle protocols, allowing the nodes to keep radios in sleep for most of the time, turning it on as infrequent as once in a few days to transfer the accumulated data.

The advancements in energy harvesting in recent years allowed for its more common application in the design of wireless sensor networks. Energy harvesting enables sensor nodes to recharge their battery by using the energy available in the environment. A typical energy source for harvesting is solar energy [25], which can be easily and cost-efficiently harvested using solar cells. However, harvesting solar energy might be an issue in an urban environment due to the lack of direct sunlight. One solution for this problem is harvesting RF energy [26], which is becoming more available recently due to technical development. Other energy sources include mechanical energy, harvested using piezoelectric devices [27], and thermal energy [28].

Energy harvesting can be used in two ways in wireless sensor networks, to increase battery life or to make self-powered sensor nodes. Due to the energy limitations that can be harvested, the latter is only possible when low-power protocols and devices are used. In sensor networks with higher throughput and performance, the benefits of energy harvesting remain limited to increased battery life. Special protocols are designed for such networks to maximize the benefits of energy harvesting. These protocols take

into consideration current energy availability and adjust protocol parameters accordingly [29].

2.2 Communication Protocol Stack

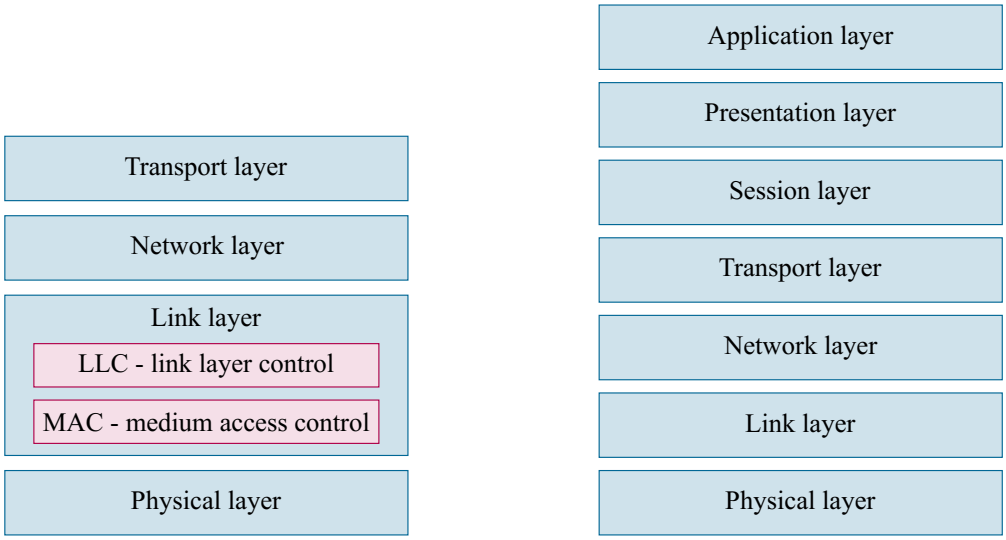
Communication protocols used by the nodes in a wireless sensor network need to perform many tasks necessary for the network operation, such as: network discovery, network joining, network re-discovery, collision-less data transfer, packet acknowledgment, packet splitting, etc. These functions are separated into different protocol layers to simplify the implementation process and allow interchangeability and re-usability. Protocol layers are stacked on top of each other to create a protocol stack.

Protocol functions can be divided into layers in different ways. A few standards that specify how this division is performed are available in the literature. The most known such standard is the OSI reference model. This model divides protocol into seven layers. The OSI model is often used in complex computer networks, where a large amount of data needs to be transferred efficiently and where the protocol is interfaced with computer software.

On the other hand, in wireless sensor networks, the application layer is implemented on the same micro-controller as the other layers. Furthermore, cost reduction and energy efficiency are among the most critical factors in wireless sensor networks, whereas computer networks usually value reliability more. Therefore, wireless sensor network stacks generally follow the OSI model but only define and implement the lower four layers. Figure 3 shows the comparison between a wireless sensor network stack and a network stack following the OSI model.

The lowest layer in the stack, the physical layer, has two main components; radio hardware specification and radio interface. Radio hardware specification defines what functionalities the used hardware must fulfill to be suitable for usage with the protocol stack. For example, it specifies parameters like carrier frequency, modulation, minimum packet length, etc. The radio interface defines a set of functions through which the upper layers interact with the radio hardware. This allows using the same link layer (the layer above the physical layer) with different hardware. Typical interface functions a physical layer defines are functions for transmitting a packet, changing radio parameters, performing clear channel assessment, etc. In addition to functions, the physical layer defines indications used to notify the link layer about events, such as a packet reception or a finished transmission.

The role of the link layer is to provide a reliable transfer of data between two nodes that are direct neighbors, i.e., can communicate directly using a radio link. It is divided into two sub-layers, Medium Access Control (MAC) and Link Layer Control (LLC). The



(a) WSN network stack

(b) OSI reference network stack

Figure 3: Comparisson of common WSN network stack organization with OSI reference model.

MAC layer has the task of transmitting packets without causing collisions with other transmissions. On the other hand, the Link Layer Control sub-layer's job is to ensure that the transmitted packet was successfully received. In the case of unsuccessful transmission, LLC initiates a retransmission of the packet. In addition, LLC is responsible for monitoring and reporting link quality to the higher layers. When the reliable data transfer is unnecessary, this part of the link layer can be omitted to lower the cost and complexity and increase energy efficiency.

The network layer is responsible for finding a route between two nodes that are not within each other's radio range. Besides that, the network layer performs network discovery and allows new nodes to join the network. However, unlike on the internet, in WSNs, this layer does not have to provide routing between any of the two nodes. Because low cost, complexity, and high energy efficiency are the most critical requirements for WSNs, this type of routing can be omitted. For example, in converge-cast networks, it is sometimes enough to provide routing from data-gathering nodes to the network sink.

The transport layer has a similar role as the LLC but on the network level. Its job is to ensure that packets reach their destination nodes and initiate retransmission otherwise. In addition, the transport layer allows for the transmission of packets whose size is larger than the link layer's limit. It achieves this by splitting large packets into chunks and resembling them again at the destination node. Because many wireless

sensor networks can tolerate some level of packet loss, it is not unusual to exclude the transport layer to reduce cost and increase energy efficiency.

2.3 Performance Evaluation Metrics

Performance evaluation of a communication protocol for wireless sensor networks is not a straightforward task because different protocols have different optimization goals. For example, some protocols are designed to be power efficient; This is achieved by sacrificing reliability and throughput to lower the power consumption. On the other hand, other protocols are designed to have high throughput and reliability at the cost of higher power consumption. Therefore, each protocol must be examined individually, using appropriate performance metrics. Standard metrics are throughput reliability, packet delay, energy consumption, and fairness.

Throughput is one of the most important parameters for evaluating network protocol performance. It represents the amount of data that can be transferred in the unit of time at a specific place in the network. It is expressed either in bit/s, or as a normalized value ranging from zero to one, with one representing continuous packet transmission [30]. Throughput can be defined for a link, a path, or the whole network. Definition of the network throughput depends on the network topology. For example, in a mesh network, it is defined as traffic received on all destination nodes in a unit of time; in converge-cast networks, it is defined as the amount of data received by the sink in the unit of time [31].

When optimizing the network throughput is the goal, it is also essential to consider whether the throughput is equally distributed among the nodes. The parameter that quantifies equality of throughput distribution is called data delivery fairness or just fairness. The most critical layer for data delivery fairness is the MAC layer. The MAC layer regulates access to the radio medium, and it is its job to ensure that all nodes can access the radio medium as often as needed. This is a challenging task, as some parts of the network experience higher contention for the medium access than the others. As a result, nodes in these areas might need to wait longer to access the radio medium. This can lead to high throughput in some parts of the network and very low in others. In total, network throughput might be high, even though fairness is low because nodes in highly congested areas are getting very little medium access time.

Fairness is especially important in converge-cast networks. In converge-cast networks, the sink node gathers all data collected by the other sensor nodes. Some of these nodes are multiple hops away from the sink. If all nodes have an equal chance to access the medium, packets from such nodes will reach the sink less often, resulting in low fairness. Therefore, protocols for converge-cast networks must ensure that packets

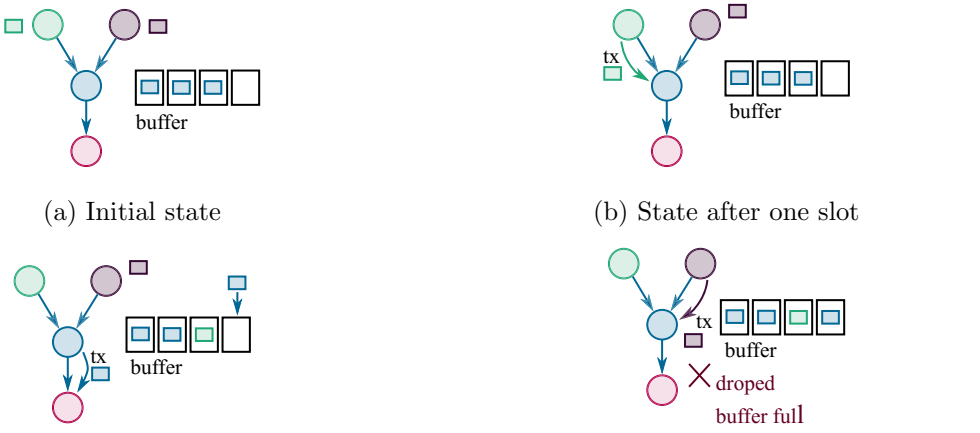
originating from all nodes have a similar chance of reaching the sink.

Multiple metrics can be used to evaluate data delivery fairness in wireless sensor networks. One of the most commonly used metrics is proposed in [32]. It is calculated using the following equation,

$$\phi = \frac{(\sum_{i=1}^{N} n_i)^2}{N \cdot \sum_{i=1}^{N} (n_i)^2} \tag{1}$$

where N is the total number of sensor nodes excluding the sink in the network, and n_i is the number of packets received from the node with index i. Fairness defined in this way ranges from 0 to 1. Fairness equal to one corresponds to a fair throughput distribution, meaning that the sink will receive the same number of packets from all sensor nodes in the network.

A common cause of reduced fairness in converge-cast networks is buffer congestion. When data generation rates are high, buffers of some nodes get full. This causes the packets from the nodes above to be discarded, while packets from the particular node experiencing high traffic are delivered, as illustrated in fig 4. As seen in the example, the blue node will always have a packet to transmit to the sink, and the throughput observed at the sink will not be reduced. However, packets from the purple and green nodes will often be dropped. Dropping packets from specific nodes and delivering one from the other node instead causes low fairness, while the throughput remains the same. If only throughput is used to evaluate performance, such a problem can remain undetected. This is why throughput and fairness must be examined together to obtain a relevant evaluation, especially when converge-cast sensor networks are evaluated.



(a) Initial state

(b) State after one slot

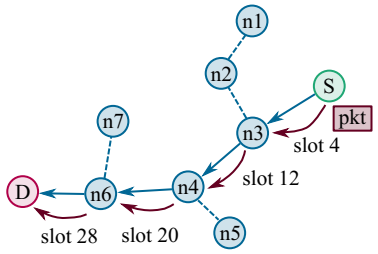
(c) State after two slots

(d) State after three slots and packet dropped in the third slot

Figure 4: Example of reduced network fairness due to buffer overflow.

In some applications, like, for example, event detection or traffic monitoring, packet delay can be of higher importance than throughput. Considerable packet delays at high throughput values are common in multi-hop TDMA networks. In such networks, nodes are in time sync whit each other, and they are allocated specific time windows (time slots) in which they are allowed to transmit. Packets often travel over multiple hops to reach the sink. Delays of such packets depend significantly on how slots of the nodes on the path are aligned.

The worst-case slot alignment is illustrated in Fig 5. Each node in the path receives a packet just after transmitting its own packet. At each hop, the packet stays in the buffer during the whole cycle before it gets transmitted to the next node on the path. This results in significant delay, especially when the number of hops is large.



(a) network

rx node:					n3								n4				
tx node:	D	n6	n4	n3	S	n5	n2	n1	n7	D	n6	n4	n3	S	n5	n2	n1	n7
slot no:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
				cycle 1									cycle 2					
rx node:			n6								D							
tx node:	D	n6	n4	n3	S	n5	n2	n1	n7	D	n6	n4	n3	S	n5	n2	n1	n7
slot no:	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
				cycle 3									cycle 4					

(b) time schedule

Figure 5: Example increased packet delay due to slot misalignment on a routing path.

Some protocols achieve high throughput values at the cost of considerable latency due to a lack of slot alignment. Therefore, besides throughput and fairness, packet delivery latency should also be considered. Packet delivery latency can be evaluated with various metrics. Common are average and maximum packet delay over all paths in the network. Another helpful metric is the maximum average delay over a path. In the case of converge-cast networks, each node communicates over one or more paths, all having the sink as a destination. Therefore, the maximum average delay is usually defined for each source node.

Another typical performance metric is the packet delivery ratio. It is defined as the

percentage of successfully delivered out of a total number of attempted packet transmissions by the network layer. Transmission of a single packet from sender to destination node, including all forwarding on the way, counts as one attempted transmission. The packet delivery ratio can be increased at the link and network layers. The network layer can improve the delivery ratio, for example, by employing network-level acknowledgments or by finding alternative routing paths. An alternative routing path can be used when the main path is experiencing congestion, which can occur for various reasons [33]. On the other hand, the link layer can improve the packet delivery ratio by better management of contention or by using link-level packet acknowledgments.

Another important performance metric is energy consumption. For evaluation of low-power communication protocol, this is the critical parameter. For example, energy harvesting nodes can provide a very limited amount of energy. However, the protocol used should be able to provide continuous operation using the available energy. Another example are battery-powered sensor networks. In this case, the network lifetime depends on the power consumption of the implemented protocol. However, energy consumption is of importance for all types of protocols. It is relevant even in the case of grid-powered sensor nodes, as low-energy consumption requires a less complex power supply, reducing the size and cost of the device.

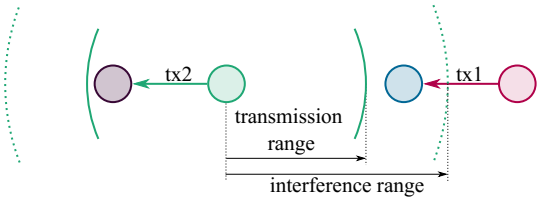
2.4 Medium Access Control - MAC

The role of the MAC protocol is to coordinate transmissions of different sensor nodes so that they do not interfere with each other. This seemingly simple task is connected with many difficulties. There are three different classes of MAC protocols based on how this problem is solved. These are contention-based, contention-free, and low-power.

When one transmission interferes with another in such a way that it causes it to fail, it is said that a collision has occurred. Based on the nature of the interference, there are two types of collisions, called type one and type two. A type one collision happens when a transmission reaches the destination node while the node is transmitting Fig. 6a. A type two collision occurs when another signal reaches the node that is receiving a packet, causing the reception to fail Fig. 6b. The node whose transmission caused the interference may be out of the transmission range, as the interference range is larger than the transmission range Fig. 6b.

There are three groups of MAC protocols based on collision avoidance strategy: contention-based, contention-free, and low-power. A contention-based protocol uses carrier sensing procedures, allowing the node to detect whether the radio medium is free or not. If it is busy, randomized delays are utilized to ensure that nodes competing for medium access retry at different times. Contention-free protocols are more complex

(a) Type one collision; node receives a packet while transmitting.



(b) type two collision; one transmission interferes with the other.

Figure 6: Type one and two collisions.

but more efficient as well. They arrange a way for radios to communicate in advance. This is achieved by creating a schedule for each node, which specifies when and at which channel the node may transmit. Contention-free protocols allow for higher throughput and higher energy savings than contention-based protocols. However, this comes at the cost of increased complexity. Namely, a lengthy and complex process involving communication with neighbors or discovery of the whole network must be performed to create a schedule. This makes various issues such as compatibility with other layers, periodical network down-times, node solicitation, etc. On the other hand, low-power protocols are designed for minimal power consumption and low traffic. These protocols keep nodes in sleep mode as long as possible; Nodes wake up periodically to transmit and receive data.

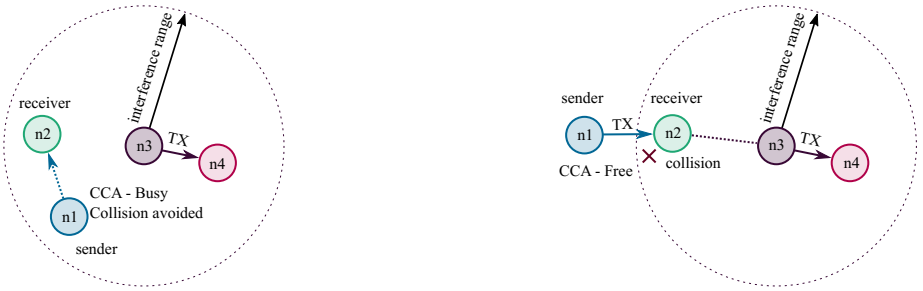
2.4.1 Contention-Based MAC Protocols

Contention-based protocols use acknowledgment packets (ACK) to detect if the transmission was successful. When a node successfully receives a packet, it sends an ACK to the sender. On the other hand, after transmitting, the sender waits for an ACK. If such a packet is not received within a predefined time window, the sender concludes that a collision has occurred. This means that other nodes have tried transmitting at the same time and that their transmissions have probably failed as well. If transmissions are repeated immediately, another collision will occur. Instead, the conflict is resolved using a protocol-specific randomized back-off scheme, which schedules contesting nodes to transmit at different times.

Most commonly used contention-based protocols are based on CSMA (Carrier Sense Multiple Access) principles. These protocols combine random back-offs with a carrier sense operation to estimate if the radio medium is busy at the moment. Carrier sense

operation is based on estimating the signal-to-noise ratio on the radio channel. This operation detects ongoing transmissions inside the receiver's interference range, as illustrated in Fig. 7a. In the figure, node $n1$ detects a transmission between nodes $n3$ and $n4$, and postpones transmission intended for node $n2$.

However, Carrier sense operation can not avoid all collisions. For example, if an ongoing transmission is not in the transmitter's range but is in the receiver's range, it will not be detected. The source of such a transmission is called the hidden terminal. The hidden terminal problem is illustrated in Fig. 7b. In this example, node $n3$ is a hidden terminal for node $n1$.



(a) CCA detects interference and prevents a collision

(b) Hidden terminal scenario

Figure 7: Usage of Clear Channel Assessment to reduce the number of collisions in CSMA and CSMA-CA.

There are many variants of the CSMA scheme. The main difference is in action taken after the medium is assessed to be busy. The persistent CSMA continues to sense the medium until it becomes free, in this case. Then, a randomized back-off is performed to resolve contention between multiple nodes contending simultaneously. On the other hand, the non-persistent variant performs the back-off immediately on a busy medium.

CSMA-CA (Carrier Sense Multiple Access - Collision Avoidance) tries to avoid initial collision and perform back-off even before the first transmission attempt. This results in fewer collisions and better performance in high contention scenarios. However, in the case of low contention, delays are increased and throughput reduced. The slotted version of CSMA further improves the performance by introducing transmission slots. When the medium is detected as free, a random time slot is chosen. If the medium is still free in the selected slot, the packet is transmitted; otherwise, a random back-off is performed.

Fig. 8 illustrates non-persistent slotted CSMA-CA, as defined in IEEE 802.15.4 standard. Nodes are synchronized and can compete for medium access during a con-

tention access period (CAP). As illustrated in the figure, this period is divided into slots when slotted CSMA is used. If a node has a packet to transmit, it performs a random back-off at the beginning of the contention period. When the back-off expires, the node performs a CCA operation on the slot boundary. If the channel is clear, it starts the transmission, like node 1 in the figure. If not, it performs another back-off, like node 2 in the figure.

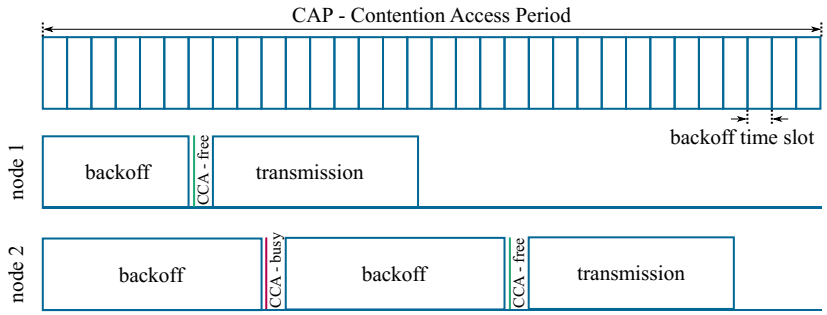


Figure 8: Slotted nonpersistant CSMA in IEEE 802.15.4.

In IEEE 802.11 standards, the RTS/CTS handshake mechanism is introduced to prevent collisions due to the hidden terminal and decrease power consumption. Before transmitting a packet, a node sends an RTS (Ready To Send) packet. An RTS contains the destination address and the length of the packet. Then, it waits for a CTS (Clear To Send) packet as a response. If there is no response, transmission is aborted. An RTS packet has two functions. The first one is to put neighbor nodes (except the destination) in sleep mode and save energy; these nodes will not be able to transmit or receive during the transmission announced by the RTS packet. The second role is preventing a collision due to the hidden terminal. If there is an active transmitter within the receiver's range, which would act as a hidden terminal to the sender, the receiver will be in sleep mode and won't respond to the RTS.

The RTS/CTS handshake mechanism reduces the number of collisions, increases maximal throughput, and reduces energy consumption. Nevertheless, the energy consumption remains relatively high, as the nodes have to be active and waiting reception whenever the radio medium around them is not busy. Therefore, this protocol is still not suitable for energy-constrained sensor networks.

As stand-alone protocols, random access protocols are not suitable for wireless sensor networks because of high energy consumption, both due to the possibly high number of re-transmissions and the inability to achieve low power operation. However, they are often used as a part of contention-free or low-power protocol. Even established protocol standards, such as IEEE 802.11 [34] and IEEE 802.15.4 [9] use some variant of CSMA

as a part of the more complex MAC protocol. In addition, most of the contention-free protocols include a random access protocol as well. For example, many TDMA (Time Division Multiple Access) protocols use CSMA at the beginning of each time slot. This prevents packet loss due to external interference or due to unexpected interference. The second may occur due to an imprecise interference range model (used to calculate the schedule) or changes in the network (which may arise after calculating the schedule).

2.4.2 Low-Power MAC Protocols

The main goal of low power protocols is to reduce power consumption as much as possible. This is achieved at the expense of packet latency and network throughput. These protocols keep nodes in sleep mode for prolonged periods to achieve low power consumption. Communication is performed in short time windows when nodes wake up. There are two classes of low-power MAC protocols, asynchronous and synchronous. In asynchronous protocols, nodes do not have clocks synchronized; therefore, they do not have information about the wake-up schedules of their neighbors. When a node wants to communicate with a neighbor, it wakes up and waits for the neighbor to signalize its wake up. Synchronous protocols eliminate this unnecessary waiting time by synchronizing nodes and informing them about the wake-up schedules of their neighbors. This comes at the price of additional overhead. The introduced overhead may benefit or increase power consumption (consuming more than saved with reduced waiting times).

The basic concept of asynchronous low-power MAC protocols was introduced in STEM (Sparse Topology and Energy Management). This protocol keeps nodes in low power listening mode when they do not have data available for transmission. Nodes wake up periodically and perform a low-power listening operation, which detects if a preamble is transmitted. When data is available, the node wakes up and transmits a preamble to indicate that it has data pending. The duration of the preamble is equal to the wake-up period of the nodes. Hence, all neighbors detect the preamble. When a node detects a preamble, it wakes up and waits for the transmission of the pending packet. This concept is illustrated in Fig. 9.

After STEM emerged, many similar protocols were proposed, which utilize the same idea while providing additional improvements. For example, the XMAX [35] introduces an array of shorter preambles instead of a long one to save energy on the transmitter side. The DPS-MAC [36] combines this approach with a low-power listening concept to reduce energy consumption on the receiver side. TICER [37] uses a short wake-up packet containing the destination address sent repeatedly instead of a preamble. When the intended receiver receives the wake-up packet, it responds with a CTS. This saves energy on the receiver side because shorter packets are transmitted instead of a

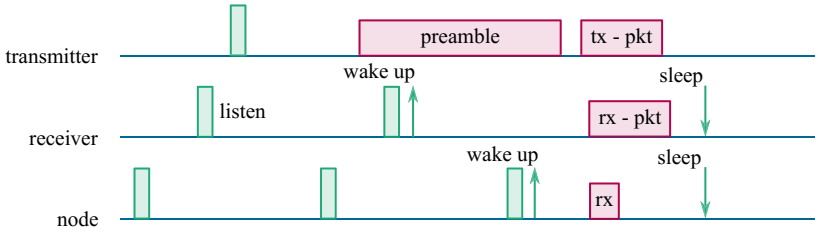


Figure 9: Waking up a receiver along the neighbor nodes using a preamble to transmit a packet in STEM.

long continuous preamble. Additionally, a CTS will be overheard by the neighboring nodes that woke up to transmit, preventing collisions due to hidden terminal. Finally, CMAC [38] introduces converge-cast routing to the low-power MAC protocols, and WiseMAC [39] utilizes schedule learning to reduce energy consumption.

RICER [37] introduces a new concept; receiver-initiated wake-up, suitable for scenarios with high traffic and dense networks. When RICER is used, all nodes wake up periodically and pull potential transmitters by sending beacons. If a node has a pending packet for a neighbor, it waits for the neighbor to transmit a beacon. Then, it competes with other potential transmitters to transmit the packet; the contention is solved using a random delay. Further improvement is achieved in RI-MAC [40] by utilizing a CSMA with a flexible contention window size. The contention window size is announced in the wake-up beacon.

A somewhat unique protocol called RC-MAC [41] is designed especially for converge-cast networks with a tree-shaped routing topology. It uses a similar approach as RICER and RI-MAC. However, it solves the contention using the parent-children relations. Additionally, it recognizes the different traffic requirements of each node based on its position in the routing tree. Namely, nodes close to the sink have more traffic because they forward packets from all nodes located above them. This is taken into consideration when wake-up schedules are created and contention resolved.

Synchronous low-power protocols synchronize either neighboring or all nodes in the network (local or global synchronization). Furthermore, during the synchronization process, nodes learn the wake-up schedules of their neighbors. This allows the nodes that want to transmit or receive to wake up when the desired node does and interact with it. However, the synchronization increases overhead because it requires periodic exchange of messages to keep the clocks synced. The benefit is that when a node wants to transmit a packet, it does not have to wait for the destination node to wake up. Instead, it remains in the low-power sleep state until the destination node wakes up and wakes up at the same time. This saves power used in the waiting state on the one

hand but uses more energy for transmitting packets necessary to keep nodes synched. Whether this leads to power savings depends on the network traffic. If traffic is very low, asynchronous protocols perform better. In most of the other cases, synchronous have a clear advantage. In addition to potential power savings, node synchronization allows more clever protocol design, increasing the maximal supported throughput.

Synchronous protocols divide time into cycles as illustrated on Fig. 10. Each cycle consists of the active period and sleep period. At the beginning of the active period, nodes wake up and exchange information about packet availability with neighbors. If a node or neighbor wants to transmit a packet, it stays awake until packet exchange is done. Otherwise, it goes back to the sleep state immediately. A typical active cycle of a few nodes running asynchronous low-power MAC is illustrated in Fig. 10. In the figure, node A announces a packet for node B. Node B stays awake to receive the packet, while node C goes to the sleep state since there are no packets destined to it announced.

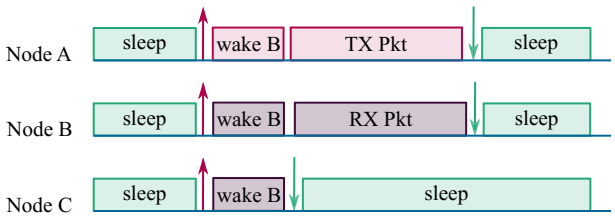


Figure 10: Cycle of a low-power synchronous protocol.

One of the first synchronous low-power protocols was S-MAC [42]. It lowers the cost of the synchronization process by clustering nodes and synchronizing nodes within a cluster. Nodes located at the cluster border keep track of the timing of both clusters. This allows fast and low-overhead synchronization of nodes in large networks. Nodes running SMAC wake up periodically; on wake-up, they inform neighbors they wish to communicate with to stay awake (as in Fig. 10). The main drawback of this approach is that packets can propagate a maximum of one-hop each cycle. After a node transmits a packet, the receiver can not forward it in the same cycle because the next hop will be in the sleep mode.

Multiple protocols in the literature improve SMAC by allowing packets to traverse more than one hop within a cycle. For example, the DW-MAC [43] protocols use an SCH packet instead of CTS and RTS to keep the receiver's neighbors awake; this allows a packet to cover two hops per cycle. Another example is the T-MAC [44] that achieves the same result with a bit different approach, which involves a packet called "future request to send."

A synchronous low-power protocol must solve three problems to achieve high through-

put and low power consumption. The first one is effective contention upon wake-up since, in a high traffic scenario, more than one node will have a packet in this case. The second is packet propagation over multiple hops per cycle. Finally, the third is the transmission of more than one packet by the same node per cycle; this is not necessarily solved by allowing the propagation over multiple hops per cycle.

A protocol called the SCP-MAC [45] solves all three problems. It introduces a contention window for sending a wake-up tone at the beginning of each cycle. Nodes that have a packet pending will compete to transmit. If they lose the contention, they remain active and ready to receive. Nodes without a packet for transmission will listen for a wake-up tone. If none is received, they immediately go back to sleep to preserve power.

The SCP-MAC introduces the adaptive channel polling to allow a node to transmit multiple packets in one slot and traversal over multiple hops per cycle; it is illustrated in Fig. 11. The adaptive channel polling is performed using adaptive contention windows; three such windows follow the main one at the beginning of each active cycle Fig. 11. These additional contention windows are used to sense traffic in the network. If there is a reception during at least one of them, the protocol concludes that the network is experiencing increased traffic. As a result, additional contention windows are added to accommodate increased traffic; they follow the adaptive ones Fig. 11. The dynamic windows remain part of the cycle as long as there is a reception during one of the adaptive windows.

The adaptive channel polling allows the transmission of multiple packets per cycle and their propagation over multiple hops. For example, in Fig. 11, node A has three packets to transmit to node B. The first one is transmitted in the regular contention window and the second one during the adaptive period. Since node B received one packet in the adaptive window, it introduces three dynamic contention windows. In the first dynamic window, it receives the third packet from node A. Dynamic slots are kept in all flowing cycles until traffic is reduced, and none of the dynamic or adaptive slots are used. On the other hand, Node C does not receive any packets during the adaptive period. It transmits one packet to node D (not in the figure) in the third adaptive contention window. Then, it goes to sleep while nodes A and B continue to communicate.

Asynchronous low-power MAC protocols are easy to implement and highly energy efficient. However, they are suitable for applications with low to medium throughput requirements. Additionally, they introduce high packet delivery delays because a node has to wait at each hop for the receiver to wake up. Therefore, those protocols are unsuitable for high throughput scenarios studied in this thesis. On the other hand,

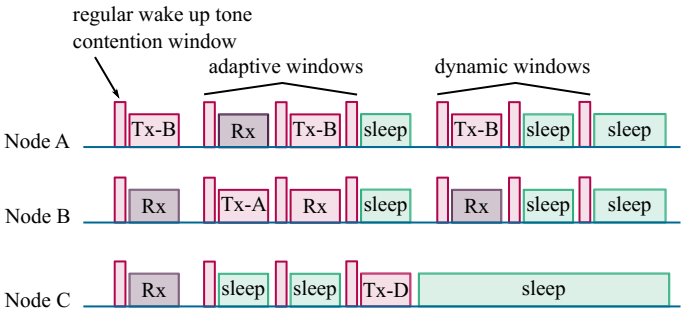


Figure 11: Dynamic pooling in SCP-MAC

synchronous low-power protocols can adapt to higher traffic loads and provide low latency and high throughput. They are, therefore, the best choice when high throughput needs to be combined with very high energy efficiency.

2.4.3 Contention-Free MAC Protocols

Contention-free MAC Protocols use network topology information to minimize or completely eliminate interference between parallel transmissions, allowing high and predictable data throughput. The first one allocates a specific time interval for transmitting to each node or link, called a time slot. A node may transmit only during this assigned interval. A few nodes that do not interfere with each other may share one time slot. This approach is called time division multiplexing. On the other hand, frequency division multiplexing assigns different radio channels to nodes instead of time slots. This approach requires more complex implementation and hardware. Therefore, it is mainly used as part of complex protocols designed for sensor nodes equipped with very capable processors. This thesis focuses on less complex, low-cost solutions that can quickly be developed and customized for a specific application. Therefore, the focus is on protocols and algorithms that use the time division multiplexing approach.

Time division multiplexing protocols are significantly more complex than random access and low-power protocols. These protocols operate in several phases. The main phase is the contention-free phase, during which nodes exchange data based on the schedule. The calculation of this schedule requires message exchange between nodes. The calculation can be performed in two ways, by negotiation between nodes or based on network topology. In the second case, network discovery is conducted to gather the required information on the network topology. A random-access protocol is used to collect data necessary for the schedule calculation in both cases.

TDMA protocols prevent collisions by assigning time slots to different transmissions. Transmissions that do not interfere with each other may be given the same slot. The

time slots can be assigned either to nodes or links. If a time slot is assigned to a node, it can use it to communicate with any of its neighbors. The collection of slot numbers and identifiers of nodes or links assigned in each time slot is called a time schedule. The number of time slots in a time schedule is called schedule length. All slots in a time schedule comprise a cycle. Cycles are repeated one after another during the contention-free period.

Before the contention-free period is started, each node must obtain its schedule. The time schedule can be calculated in a distributed or centralized manner. After the schedule calculation and distribution, each node receives its part of the schedule; a part of the schedule belonging to a specific node is called the node schedule. A node schedule contains information about slots in which the node transmits, stays awake to receive, and goes to sleep. Schedules of different nodes in a network are using an example network, illustrated in Fig. 12. The example network has three nodes and one sink. Node $n1$ is assigned three slots, one to transmit its packet and two for packets from the other two nodes. Node $n1$ receives a packet in the first two time slots and transmits it in the following three time slots. Node $n2$ transmits its packet in the first time slot to node $n1$. Then, it goes to sleep for the rest of the cycle.

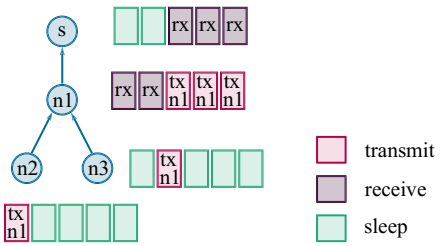


Figure 12: Example showing schedules of nodes in a network.

The difference between node-based and link-based scheduling is illustrated using two examples in Fig. 13. The figure shows two different networks and their schedules. For the network in Fig. 13a, a node-based schedule is calculated, while the network in Fig. 13b uses a link-based schedule. The benefit of link scheduling is a better adaptation to traffic and a higher number of possible parallel transmissions. A higher number of parallel transmissions is possible because each link is considered separately. For example, on Fig. 13b, in the second slot, node $n2$ transmits a packet to node $n1$; in the same slot, node $n4$ transmits to node $n5$. A node-based schedule could not schedule these two transmissions simultaneously.

Link scheduling has a few significant drawbacks. The most critical ones are adaptability to changes in traffic and long schedule calculation time. For example, if a node

(a) Node scheduling

(b) Link scheduling

Figure 13: Examples of node and link scheduling.

does not have a packet for the scheduled link but has a packet for another, the slot remains unused. Most importantly, link-scheduling does not provide benefits when used for scheduling converge-cast networks. This is because most of the traffic flows in one direction, towards the sink. On the other hand, link-based schedules can schedule two links from one node in the same time slot if the other sides of the links are in different directions (a parent and a child). Since converge cast networks have links only in one direction, a link-based approach would not benefit such a network. Increasing throughput in converge-cast wireless sensor networks is the main topic of this thesis; therefore, the emphasis is put on algorithms that use node-based schedules.

TDMA protocols can calculate schedules locally or globally. When the local calculation approach is used, each node calculates its schedule by negotiating with surrounding nodes. The main problem with this approach is determining cycle length. For a schedule to work, each node must use the same cycle length. To do so, all nodes must agree on the same schedule length. However, each node has information about its schedule length and hose belonging to nodes in its neighborhood (it can be a one-hop or larger neighborhood). This is illustrated on Fig. 14. Node $N1$ has a schedule length equal to three and node $N4$ to four. Node $N3$ does not know the schedule length of node $N4$, and if it starts repeating the schedule after the third slot, it will cause collisions. This problem can be solved by presetting the cycle length to a specific value, line in DRAND [46] or Z-MAC [47]. However, this always leaves some slots unused, resulting in decreased throughput.

Another issue most local scheduling algorithms have is the lack of adaptation to the traffic. This problem is partly solved by TRAMA [48]. To adapt schedules to the traffic, TRAMA introduces moving time cycles. Each node announces cycle length and the percentage of the cycle it uses to the neighbors. Thanks to this announcement, each

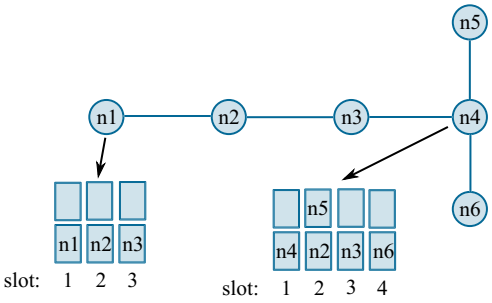


Figure 14: Phases of a central scheduling TDMA MAC.

node has information about the slot usage of its neighbors. Then, a node calculates priority for each slot using a hash function to determine which slots to use. For each slot, a node calculates its priority and priorities of all nodes in the two-hop neighborhood. The drawback with this approach is that the number of slots each node gets can not be controlled precisely, allowing only a rough adaptation to the traffic. Namely, the number of slots each node receives is pseudo-random. The range in which this random number value lies is controlled by adjusting the active portion of each node's cycle.

Local scheduling provides good performance while keeping the implementation cost low. However, the number of slots assigned to each node can not be efficiently adapted to the traffic requirements. Converge-cast networks, which are the main interest of this thesis, use tree routing topology to deliver the gathered data to the sink. Traffic requirements of different nodes in a tree-based network differ dramatically among the nodes. For example, leaf nodes only need a few slots for their data. On the other hand, nodes close to the sink will need many slots to forward the data from nodes above them. Therefore, local scheduling TDMA protocols are not suitable for usage in converge-cast networks.

Centralized scheduling algorithms are more complex to implement and have longer network setup times; however, an exact number of slots can be assigned to each node, allowing precise adaptation of protocol to the traffic requirements of each node. The centralized approach also allows for better parallel transmission optimization, significantly reducing cycle length. Therefore, these protocols are the most suitable for converge-cast networks with high data packet generation. The drawbacks are longer network setup time and complex algorithms for adding new nodes to the network, which require network discovery and the schedule calculation to be performed again. However, this is not an issue because nodes are fixed in most application scenarios. Such applications are, among many others, traffic monitoring systems, industrial monitoring systems, and medical body area networks.

Centralized scheduling algorithms are executed in three or four phases; network

discovery phase, schedule dissipation phase, scheduled access phase, and optionally random access phase. There are a few reasons to include the random access phase. For example, a node can discover new nodes or send protocol-specific packets during this phase. Using the random phase for protocol control packets is beneficial because time slots are optimized for data packets destined for the sink. Control packets may experience high delays and packet loss in the contention-free phase.

Fig. 15 illustrated typical change of phases in a centralized TDMA MAC protocol. When the protocol is started, it is in the network discovery phase. During this phase, the network is discovered, and schedules are calculated. The schedules are then distributed to the nodes during the schedule distribution phase. Then, data transfer is started, and scheduled access and random access phases are cycled periodically. After some time, data transfer is temporarily stooped to rediscover the network and calculate a new schedule. A rediscovery is performed to account for changes in network topology, radio medium, and traffic requirements of individual nodes.

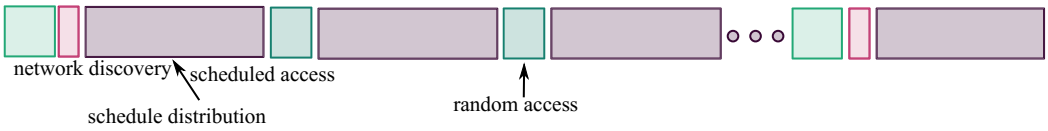


Figure 15: Phases of a central scheduling TDMA MAC.

The sink node discovers the network during the network discovery phase, using protocol-specific packets. During this phase, the sink learns the topology of the whole network. This includes a list of all nodes, neighbors of each node, and optionally, link quality information. Based on this information, it represents the network with a graph. Vertices of this graph are sensor nodes, and edges are links between nodes. Additionally, the sink creates an interference graph to represent interference relations in the network. This graph is created either based solely on the network graph or using additional information gathered during the network construction phase. Vertices of the interference graph are sensor nodes, and an edge between two vertices will exist if the corresponding nodes can not transmit simultaneously.

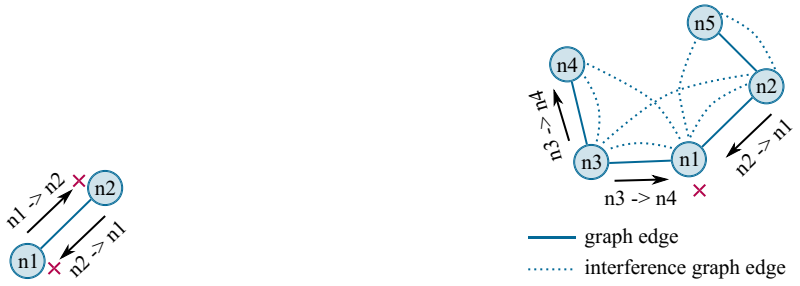
The problem of finding a schedule with minimal length is equivalent to finding a minimal graph coloring of the interference graph. However, such coloring will only assign one color to each node, i.e., time slot. Therefore, additional adjustments are needed to consider traffic flow and adapt the number of colors assigned to each node based on its traffic requirement.

An interference model is used to construct an interference graph. This model allows calculating which nodes interfere with each other. Most commonly used interference model is the two-hop interference model [31, 49–52]. According to this model, an in-

terference edge exists between two nodes if the distance between them in the network graph is one or two hops. This is based on the assumption that the transmission and interference ranges are equal. Using this model, an interference graph can be constructed based only on the network graph, which is the main advantage of this model.

According to the two-hop interference model, there are two occasions in which a node may not transmit, illustrated in Fig. 16. The first one is when its neighbor is transmitting; if the node transmits in this case, it can not receive the neighbor's packet. In the figure, node $n1$ can not transmit when node $n2$ is transmitting. The second occasion is when a two-hop neighbor transmits. In this case, a transmission would cause a collision at the node between (one-hop neighbor). For example, if nodes $n3$ and $n2$ in the example (Fig. 16) transmit at the same time, collisions occurs at the node between them, node $n1$.

Besides the two-hop interference model, a node-based model, there are link-based interference models based on the same assumptions. In the example (Fig. 16), the usage of such a model would allow $n3$ to transmit to $n4$ at the same time when node $n1$ is transmitting to node $n2$ (called back to back transmission). However, such benefits will be only possible for transmission in the opposite directions in regards to the sink. And since traffic in converge-cast networks flows towards the sink, such models are not relevant for the problem studied in this thesis.



(a) Type one collision, caused by immediate neighbors

(b) Type two collision, occurs when nodes two hops apart transmit at the same time

Figure 16: Two types of collisions that can occur according to the two-hop interference model.

The problem of finding a schedule with the minimal length for converge-cast sensor networks is NP-complete, as proven by Choi et al. [11]. Many algorithms that find an approximate solution have been proposed so far [31,50,51]. Some of them use standard graph coloring algorithms. Others find different simplified solutions for solving this problem. For example, TreeMAC [31] schedules one linear sub-network at a time. This approach simplifies schedule calculation without reducing schedule length drastically.

The result is a very efficient yet simple protocol with a very short network setup time (in comparison with other protocols of this type).

Centralized TDMA scheduling protocols achieve high data throughput in converge-cast sensor networks with data gathering tree topology while keeping complexity low. More complex, multi-channel, or slotted channel hopping protocols allow even higher throughput but at the cost of significantly increased complexity and hardware requirements. Using multiple channels requires complex node discovery and a network joining process. Furthermore, nodes can always get out of synchronization and lose contact with the rest of the network. To solve this problem, complex routines for checking for such nodes and enabling them to rejoin the network are necessary. This all requires strict standards and a long development time. Such protocols are not practical for custom applications and simple hardware. Instead, they are suitable to be used as part of standards like Wireless Hart or IEEE 18.15.4. These protocols use complex radios with parts of the protocols implemented in hardware. These are predefined standards and can not be wholly adapted to the custom application. Therefore, this thesis focuses on single-channel protocols, which can provide easy and efficient customized solutions with low cost and high energy efficiency thanks to the simpler hardware.

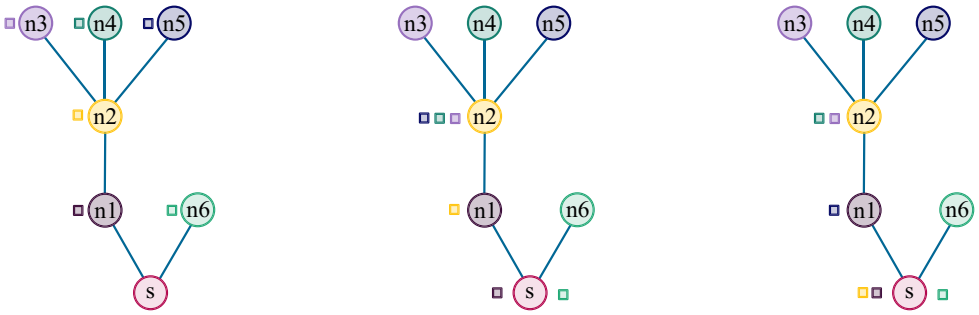
2.5 Challenges in Designing a TDMA MAC Protocol for Converge-Cast Networks

For a TDMA MAC to perform well in a converge-cast network, it should create a sufficiently short schedule, and, at the same time, adapt that schedule to the traffic. Some protocols disregard schedule adaptation; the result is a short schedule that performs poorly despite its length. A schedule should be adapted to two properties of the traffic. The first one is the unsymmetric traffic requirement among the nodes; to adapt to this property, a schedule allocates an adequate number of slots to each node. The second one is the traffic flow. An adapted schedule considers at which point in the schedule a node receives a packet and assigns it a slot for forwarding that packet afterward (not before it received the packet).

Converge-cast networks use a tree-based routing topology. In a tree-based topology, nodes at lower levels (close to the sink) are forwarding packets from nodes above them; the number of such packets can be quite high. Therefore, traffic requirements vary drastically among the nodes in such networks, especially if the number of nodes is large. A TDMA protocol that does not consider this will perform poorly.

An example in Fig. 17 illustrates the importance of adapting the number of slots assigned to a node to its traffic requirement, even in small networks. One time slot is assigned to each node in the example, regardless of its position in the routing tree. The

figure illustrates the traffic flow in the network. In the second cycle, four nodes ($n3$, $n4$, $n5$, $n6$) do not use their time slot; they do not have a packet to transmit. At the same time, node $n2$ has four packets in the second cycle, but it can transmit only one.



(a) Initial packet distribution

(b) Packet distribution after the second slot

(c) Packet distribution after the third slot

Figure 17: Illustration of large number of unused slot in the case when equal number of slots is allocated to each node in a converge-cast network.

After adapting the number of slots to the traffic requirement of each node, adapting the schedule to the traffic flow is the second vital task of a scheduling algorithm. In converge-cast networks, the traffic flow is predictable, as all data is directed in the same direction, towards the sink. Therefore, a schedule can be adjusted to the traffic flow based on the network topology. However, this property is commonly disregarded by scheduling algorithms. As a result, packet latency increases because packets need multiple cycles to reach the sink. Furthermore, packets can be accumulated at buffers while waiting for a transmission slot, resulting in data loss.

An example in Fig. 18 illustrates the effects of a scheduling algorithm not considering traffic flow. In the example, each node first transmits and then receives a packet from its child. This results in high latency because packets traverse one hop in each cycle. Furthermore, this reduces the chance that a node uses its transmission slot. Namely, in reality, packets are not generated at all nodes simultaneously. Therefore, it can happen that a node does not use its slot (due to the lack of packets) and then receives a packet and generates its own. This can cause packet accumulation at buffers and potentially their loss.

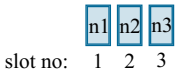
Another essential factor for good performance is eliminating interference between the nodes transmitting in the same time slot. The commonly used two-hop interference model is simple but does not remove interference completely. Some interference models use the highest radio power to detect interference sources; this approach results in precise interference graphs. However, when this model is used, the highest radio power

(a) Initial packet distribution

(b) Packet distribution after the first cycle

(c) Packet distribution after the second cycle

(d) Packet distribution after the third cycle



(e) Time schedule

Figure 18: Time slot synchronization where parent is scheduled ahead of its children.

can not be used for transmissions. As a consequence, the effective radio range is reduced. Besides the obvious downsides of reduced radio range, throughput is lower because packets need to travel over a larger number of hops to reach the sink. In this thesis, an adaptive N-hop interference model is proposed, which keeps the simplicity of the two-hop model while improving performance significantly.

2.6 Network Layer

Sensor networks usually cover an area larger than the radio range of individual nodes. Many nodes in such networks are not within each other's radio range. Nodes outside radio range exchange packets using nodes between them to forward a packet. The task of forwarding packets is performed by the network layer. The network layer uses protocol-specific packets to discover leading to the desired destination node in the network. Each node stores information about routes in a routing table. A routing table contains a list of known reachable destination nodes; for each destination address, the table stores the address of the next hop.

For each destination node, there may be different routes available. In convergecast networks, the destination is always the sink node. Nonetheless, there are often multiple ways to reach the sink from each node. There are many different algorithms for choosing and maintaining routing paths. Routing algorithms can select routes using different optimization criteria. For example, if high throughput is desired, the shortest possible routes are chosen. On the other hand, if nodes are energy constrained, routes are chosen to spread data evenly among the nodes; this results in some packets taking

longer routes to reach the sink.

TDMA MAC protocols for tree-based sensor networks create schedules based on the routing topology. Therefore, there is a strong connection between the MAC and the link layer. This chapter first describes the relationship between these two layers in such protocols. Then, it gives an overview of state-of-the-art routing algorithms for convergent networks. Finally, a connection between the routing topology and TDMA performance is commented on in Section 1.6.3.

2.6.1 Network Layer Organization

Because schedules for converge-cast networks are calculated based on the network topology, a schedule is unavailable when the protocols start. Instead, another protocol, usually a random access, is used initially. Information necessary to calculate the schedule is gathered using that auxiliary protocol. Information gathering involves network discovery in most cases; it can be performed in a centralized or distributed manner. If it is centralized, the sink node acts as a master, discovers the whole network, and gathers all the necessary information to calculate the schedule. Otherwise, individual nodes communicate with other nodes in the network to exchange the necessary data. In both cases, there is a need to communicate with nodes more than one hop away.

Schedule calculation needs to exchange messages with nodes in the network other than immediate neighbors. However, the network layer, located above the link, is responsible for that task. Furthermore, in most cases, schedule calculation involves network discovery; the network layer does this to create the routing anyway. Due to all this, the schedule calculation part of a TDMA MAC is usually implemented inside the network layer. Otherwise, network discovery and routing would have to be implemented two times.

The organization of the network and link layer of a TDMA protocol is illustrated in Fig. 19. The schedule calculation is one sub-unit of the network layer. This part of the TDMA calculates the schedule and sends them to the MAC layer in the form of a protocol-specific message. Network discovery is implemented as a separate sub-unit. At the network startup, network discovery discovers the network first; it does so using the random access part of the MAC layer. During the network discovery, the routing tables are created as well. Once the network is discovered, the schedule calculation algorithm can use the information gathered to calculate the schedule. If necessary, the schedule calculation algorithm can collect additional information beforehand by transmitting protocol-specific packets; routing tables are available by this point, allowing communication with other nodes.

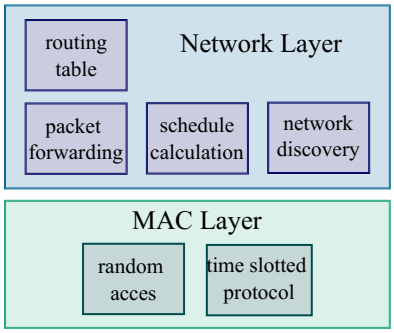


Figure 19: Organization of the network and MAC layer.

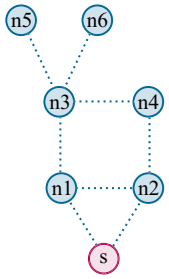
2.6.2 Network Discovery

The network layer is responsible for network discovery and creating and maintaining routing tables. These two are interconnected because network discovery requires exchanging data with nodes more than one hop away. Similarly, route discovery includes discovering new nodes and links. For these reasons, network discovery and route selection are usually integrated when tree-based sensor networks are the case. Usually, a neighbor discovery algorithm is run first, allowing each node to discover immediate neighbors. Then, a tree formation algorithm is run. This algorithm assigns a level to each node and a parent node to each node except the sink; the parent assignment defines routing paths.

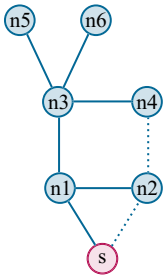
The majority of protocols use a shortest-path spanning tree for routing. In such a tree, the distance from any node to the sink is equal to the shortest distance between these two nodes in the graph. The difference between a spanning tree and a shortest path spanning tree is illustrated in Fig. 20. Using a shortest-path tree for routing ensures that each packet reaches the sink with the shortest possible number of transmissions. It also simplifies the application of the two-hop interference model. In such a tree, the distance between two nodes is equal to the distance between the same two nodes in the graph. Therefore, knowing a node's two-hop environment is not necessary to prevent collisions.

Protocols designed for energy-constrained nodes do not use shortest-path trees; these protocols deliberately choose longer paths to distribute the traffic evenly among the nodes. On the other hand, most high throughput use shortest-path spanning trees. Only one exception is known to the author, the protocol by Lai. *et. all.* [13]. This protocol starts with a shortest-path spanning tree. Then it changes some of the paths to allow more slot sharing.

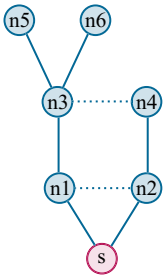
Algorithms for creating a shortest-path tree are well known and studied. Commonly



(a) Graph



(b) Spanning tree



(c) Shortest path spanning tree

Figure 20: Difference between spanning tree and shortest path spanning tree.

used approaches are the centralized Dijkstra's algorithm and distributed update-based algorithm. These algorithms require that each node has information about its neighbors. Then, they assign a parent node to each node in the network, excluding last-level nodes; this assignment is performed in a way that the path from any node to the sink is a shortest path. These algorithms do not collect topology information. Therefore, if they are used without modification, the sink node needs to run an additional algorithm afterward to discover the network. This thesis used a modified Dijkstra's algorithm; it uses modified packets to provide topology information to the sink without increased complexity. This chapter describes the original Dijkstra's algorithm, and the modification is described in Section 3.2.2.

Dijkstra's algorithm performs path selection (equivalent to parent assignment) level by level. Tree construction also discovers level, in the sense that nodes learn at which level they are. Path selection at each level corresponds to one construction phase. The sink initiates every phase p by broadcasting a $start_p$ message. At the beginning of phase p, level $p-1$ is discovered (from the previous phase); the last level discovered is called the leaf level. When a node receives a $start_p$ message, it retransmits the message unless the node is at level $p-1$. As a result, a $start_p$ message is retransmitted until it reaches level $p-1$ (the leaf level).

When a leaf level node receives a $start_p$, it transmits a $join_{p+1}$ message to its quiet neighbors instead of forwarding it. Quiet neighbors are nodes that have not joined the tree yet (did not get a parent assigned). When a node receives a $join_{p+1}$ message, it responds with an ACK message. The node's neighbors hear this ACK, and they remove it from the list of quiet neighbors. After sending a $join_{p+1}$ message to all quiet neighbors, the leaf level node sends an *echo* message to the sink. This message is used to synchronize the construction process. A node waits for an *echo* message from all its children before forwarding it to the sink. Therefore, the sink receives an *echo* message

only when all leaf nodes finish the construction of the next level.

The message complexity of Dijkstra's algorithm is $\mathcal{O}(m + n \cdot diam)$ and time complexity is $\mathcal{O}(D^2)$, where m is the number of edges, n the number of nodes and $diam$ is the graph diameter. It is an efficient and well-studied algorithm that is simple to use and debug. It is also easy to modify to gather additional information, a property often required in wireless sensor networks. Most protocols use this algorithm for these reasons, even though the update-based algorithm offers improved performance.

The update-based algorithm is a distributed algorithm that improves over Dijkstra's in terms of time and message complexity. The message complexity achieved with this algorithm is $\mathcal{O}(m \cdot diam)$ and time complexity $\mathcal{O}(D)$. However, this algorithm does not send acknowledgments to the sink. Acknowledgments are often used to provide the sink with topology information necessary for some of the protocols it is running (for example, TDMA schedule calculation). Therefore, the Dijkstra algorithm is used in this thesis because it allows for more flexibility.

2.6.3 Routing Topology Optimization

The routing tree optimization algorithms available in the literature can be divided into three groups. The algorithms of the first group aim to create load-balanced trees; this can provide various benefits, including shorter TMDA schedules and more balanced energy consumption. The second optimization is designed for data aggregation sensor networks. These algorithms minimize the number of packets required to aggregate all data. Finally, the last group is comprised of algorithms for prolonging the network lifetime. They try to distribute the traffic as evenly as possible.

There are two types of load-balanced trees, a top load-balanced tree, and a fully load-balanced tree. These two types are illustrated in Fig. 21. The term fully load-balanced Tree was coined by P. Hsiao *et. al.* [53]. It refers to a tree in which all branches at all nodes carry the same loads. For a tree to be fully-load-balanced when all nodes have one packet (one load), every node must have the same number of children. Unfortunately, such trees are almost impossible to construct in real-world sensor networks. Therefore, a more practical type of load-balancing, top load-balancing, is introduced. In a top load-balanced tree, all top subtrees carry the same load. A top subtree is a subtree rotted at a direct neighbor of the sink.

One fully load-balanced tree is illustrated in Fig. 24a. The number of packets each node receives from every child in that Tree is the same. For example, node $n3$ receives one packet from both children. In the same way, the sink gets four packets from both of its children. Fig. 21b shows a top load-balanced tree. It is easy to see that this Tree is not fully load-balanced. For example, node $n1$ receives one packet from node

(a) A fully load-balanced tree

(b) A top load-balanced tree

Figure 21: Difference between spanning tree and shortest path spanning tree.

$n4$ and two packets from node $n3$. However, both sub-trees rooted at the sink receive four packets from each child. Therefore, this Tree is top-load balanced but not fully load-balanced.

There are many load-balancing algorithms in the literature. Some of them are focused on prolonging network lifetime [54], [53]. They spread routing paths evenly among the nodes to balance the energy consumption. The paths chosen are often deliberately longer than necessary; this increases the total number of transmissions but lessens the burden from the busies routing paths, prolonging the network lifetime. From the perspective of this thesis, these algorithms are not beneficial as they are not aimed at increasing data throughput.

Load-balancing algorithms that keep the shortest-path tree topology can lead to a shorter TDMA schedule and increased network throughput. Two such protocols that achive very good results are considered in this thesis. The first one is proposed by O. D. Incel *et. al.* [50]. This algorithm assigns children to parents level by level. It uses search sets when choosing a child for each parent. Search sets allow the algorithm to choose a child that will leave more options open in the future (when assigning children to upper levels). Another state-of-the-art algorithm considered here is Energy driven Tree Construction (ETC) [55]. This algorithm tries to create a tree close to a fully load-balanced tree. The algorithm calls such a tree a near-balanced tree. The algorithm calculates the number of children each node would have in a fully load-balanced tree. Then, the nodes negotiate with the neighbors and try to adjust the number of children to be as close to that number as possible.

This thesis considers a problem that was not analyzed so far to the best of the author's knowledge; optimizing a TDMA schedule for multiple packet transmissions during a single time slot. This thesis proposes a TDMA protocol that optimizes its schedule for multiple packet transmissions in the same time slot, called M-TreeMAC. As discussed in setion 4.1, a schedule optimized in such a way is heavily dependent on

the routing topology. It is also observed that such a schedule is generally shorter when the routing tree is load-balanced. An algorithm for optimizing the topology to produce the shortest possible schedule when M-TreeMC is used is proposed in this thesis. As it is the first algorithm of that kind, its performance is assessed by comparing it against state-of-the-art load-balancing algorithms.

2.7 Conclusion

This chapter gives an overview of the state-of-the-art efforts to increase data throughput in wireless sensor networks. In Section 2.1, the concept of sensor networks is introduced and explained which parts of the protocol stack can be improved to increase throughput. Because the MAC layer was identified as a critical point, and in detail overview of state-of-the-art MAC protocols are given in Section 2.4. Based on this overview, it is concluded that contention-free protocols are the most suitable choice. As the research in this field is already developed, the proposed strategy is to choose the most appropriate protocols for the application and introduce small modifications to adapt it even further. Section 2.5 deals with challenges faced when designing a TDMA protocol for a specific type of application. Finally, Section 2.6 is explaining the relationship between the MAC layer and the network layer and how routing topology can affect data throughput.

3 Selecting Schedule Calculation Algorithm

The problem of finding the optimal schedule for converge-cast networks is well studied. This problem is NP-Complete, and the optimal solution can not be found in polynomial time. Therefore, efforts have been put into developing algorithms that find a near-optimal solution. A significant number of such algorithms have been proposed up to date. Some of these algorithms use a graph coloring approach. They modify the existing graph-coloring algorithms to assign time slots to nodes while taking care of the traffic flow and the difference in the number of slots each node requires. Other algorithms consider slot assignment as a separate problem and use more intuitive approaches to solve it.

Even though multiple publications that propose different algorithms for solving this problem exist, no comprehensive comparative evaluation of these algorithms has been published up to date, to the best of the author's knowledge. For example, authors of TreeMAC [31] compare TreeMAC with CSMA-CA and Funneling-MAC. The first one is a random-access protocol, while the second one is a hybrid design of low-power and TDMA protocol. Park *et. all.* compare their protocol with two much simpler and less efficient slot assignment algorithms, one by [50] and the other used in TRASA [56]. However, a comparison with complex and efficient algorithms does not exist. Furthermore, all existing performance evaluations are performed using either one single or only a few different networks.

Due to this lack of data, choosing the most effective TDMA MAC protocol for a specific high throughput application is not a straightforward task. The only information available for most of the protocols is the theoretical lowest bond on the schedule length, and this bond is similar for many protocols. There is no actual information on what throughput ranges can a protocol achieve and under what circumstances (including network density, topology, and traffic nature).

This thesis chapter provides a guideline for selecting a slot assignment algorithm. This guideline is created based on an extensive evaluation of the state-of-the-art slot assignment algorithms. The evaluation is performed using the OMNeT++ simulator and over 200 networks of different sizes and topologies. An advanced randomized traffic generation model was used to test the existing algorithms at different packet generation rates. The results are analyzed, and the guideline for choosing a protocol based on the network topology and size is proposed.

This chapter is organized as follows. Section 3.1 gives an in-detail overview of the state-of-the-art slot assignment algorithms. Each of them is explained in detail, together with design logic and expected weaknesses and strengths; this helps provide

insights into which one is suitable for a specific application. Next, the evaluation of the described algorithms is given in Section 3.3. First, the implementation details are explained. This includes the implementation of protocols that use these algorithms in OMNeT++ and random network generation algorithms used. Guidelines for choosing an appropriate algorithm are provided in this section as well. Finally, the results and findings from the whole chapter are summarized in Section 3.4.

3.1 State-of-the-Art Schedule Calculation Algorithms

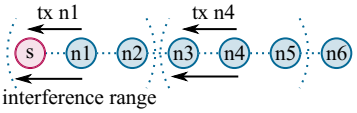
3.1.1 TreeMAC

TreeMAC [31] is a simple yet very effective scheduling algorithm. It is based on the observation that the bottleneck in every converge-cast network is located at the sink. Due to this, parallel scheduling of different top sub-trees does not benefit performance significantly because the roots of top subtrees have to be scheduled separately. When scheduled in parallel, the packets can reach the roots of top subtrees, but they will accumulate there. Based on these observations, TreeMAC authors propose to schedule one linear sub-network at a time. They find a way to construct a schedule in a distributed manner without the need to discover the whole network. This approach significantly decreases the network construction time and reduces complexity; the cost is increased schedule length. However, the increase in schedule length is minor in many cases; the simulations performed in this thesis identify these cases.

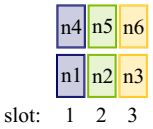
TreeMAC uses the two-hop interference model. The algorithm calculates a schedule without knowing the topology of the whole network; this is possible when the two-hop interference model is combined with scheduling one linear subnetwork at a time. Namely, in a linear network, whether two nodes interfere or not can be determined based on their level solely. If the difference between their levels is one or two, they will interfere; otherwise, they can transmit in parallel.

The example in Fig. 22 shows the schedule created by TreeMAC for a linear network. In the first slot, nodes at levels one and four are scheduled. Then, nodes at distances two and five in the second. Similarly, nodes at levels three and six in the third. In this way, all nodes have been scheduled once within three time slots, and the schedule was calculated based on the level of each node.

Implementation of such a schedule calculation is simple. Every node needs to know its distance from the sink, which is normally obtained during the network construction phase. Then, it can calculate its time slot using the expression $slot_{no} = (d-1) mod 3$. In this expression, d represents the distance from the sink in hops (equal to the node level). Such a schedule assigns only one time slot to each node.



(a) Linear network and two nodes outside each others two hop range transmitting at the same time



(b) Schedule allowing parallel transmission

Figure 22: Time schedule for a linear network created using two-hop interference model

To assign a number of slots proportional to the traffic demand, each node must know the number of nodes above it and the cycle length. A node can obtain the first one in the network construction phase (by minor modification of the construction algorithm). But the cycle length is only known by the sink. However, the sink can distribute it to all nodes after network construction without prolonging the construction time considerably. A group of three time slots, which is being repeated, represent one frame.

The organization of a schedule created by TreeMAC is shown in Fig. 23. Tall slots in the schedule comprise a cycle, and a cycle is composed of frames. Each frame has three time slots, as explained earlier. Instead of assigning slots, treeMAC assigns frames to nodes. Each node calculates which slot it may use within the assigned frame based on its level. During one frame, each node passes one packet to its neighbor. In this way, one packet reaches the sink during each frame. The total time frames required to collect packets from all nodes is equal to the number of nodes in the network N, and the total number of time slots is equal to $3N$.

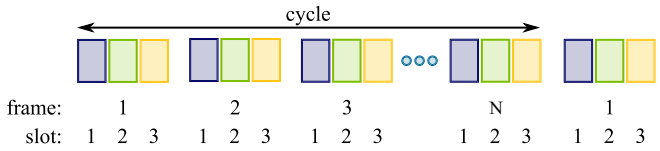
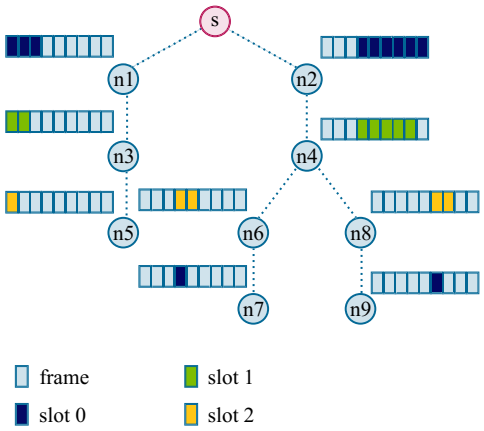


Figure 23: Cycle of TreeMAC composed of frames, each three slots in length.

The number of frames required for a branch to transmit all packets to its root is equal to the number of nodes in the branch, excluding the root. This is used to calculate the schedule in a distributed manner. The only information needed about the topology is the number of nodes in each branch. This information is easy to obtain during the tree construction process with little or no modification based on the tree construction algorithm used. The sink then initiates schedule calculation. Sink calculates cycle length in frames, equal to the total number of nodes in the network and starting frame for each branch. Then, it passes this information to each of its children. This information is then passed further. Each node assigns the starting frame

to its children based on the number of nodes in that child's sub-tree. This is illustrated in Fig. 24. The sub-tree rooted at the node $n1$ contains tree nodes; therefore, it is assigned frames from one to three. The sub-tree rooted at the node $n2$ has six nodes, so it gets assigned the next six frames, four to nine. Similar is repeated at node $n4$, which assigns frames four and five to one and six and seven to the other branch.



(a) TreeMAC frame assignment

Figure 24: Example of schedule generated by TreeMAC with the illustration of frame assignment performed starting from the sink.

Although many algorithms that produce shorter schedule lengths exist, TreeMAC is still important because it offers a very good balance between complexity and performance. It achieves satisfactory performance in terms of throughput while, at the same time, implementation complexity and network setup time remain low. Simulations presented in Section 3.3 show that treeMAC performs almost as well as the more complex algorithms in many scenarios.

3.1.2 Scheduling by Gandham *et. al.*

The protocol proposed by Gandham *et. al.* [51] is a more advanced and complex version of TreeMAC. It is developed based on the same ideas, usage of two-hop interference model and scheduling one linear sub-network at the same time. However, Gandham *et. al.* introduce both more advanced scheduling for a linear network and scheduling of different top-level sub-trees at the same time. The implementation remains distributed, because nodes calculate their own schedule based on a few parameters provided by the network sink.

Gandham *et. al.* first analyze scheduling problem of a linear network. They propose a scheduling algorithm that achieves schedule length of $3N-3$, where N is the number of

nodes excluding the sink. Then, they prove that this is the shortest possible schedule for this problem. TreeMAC achieves schedule length of $3N$ in this case. Another improvement compared with TreeMAC is the slot order. Gandham *et. al.* propose to schedule each node after its parent has transmitted a packet, as opposed to scheduling nodes before their parents in TreeMAC. This increases the chance that a node will have a packet to transmit assigned slot, because if it does not have its own packet, it might have a packet forwarded to it form its parent in the previous slot.

To calculate the schedule in a distributed manner, each node needs to know its distance from the sink and total number of nodes in the network. Nodes then keep track of their state. There are three possible states a node can be in, transmit(T), receive(R) and inactive(I). The initial state is calculated based on the distance from the sink d. If $dmod3$ is equal to 1, it is set to T; if it is equal to 2, it is set to R and if it is equal to 0, it is set to I. The state is then changed at each time slot transition, according to the state transition diagram on the Fig. 25. The nodes follow this state machine for $3(N-2)$ time slots. After that, only the two nodes closest to the sink will have a packet to transmit. For the next three time slots, all nodes are in state I and nodes with distance one and two from the sink are scheduled separately. The resulting schedule for a network with six nodes can be seen on Fig. 26a and the schedule TreeMAC would produce for this network on Fig. 26b. As it can be seen from the figures, TreeMAC results in unused slots at the end of the schedule; Gandham *et. al.* solves this problem by assigning the last three time slots differently.

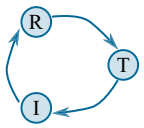
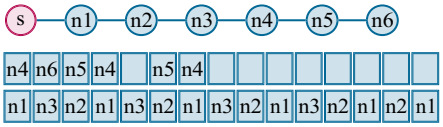
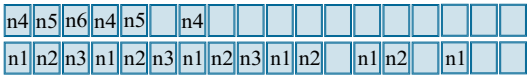


Figure 25: State machine used to determine in which slot to transmit

Further improvements over TreeMAC are achieved when creating a schedule for a tree structured network. Gandham *et. al.* observe that if there are three independent top sub-trees, the schedule can be created in such a way that the sink receives one packet in each time slot. This is achieved by scheduling sub-trees in parallel. In the first time slot, the largest sub-tree is scheduled and it remains active all the time. Then in the second time slot, the second largest sub-tree is scheduled, and set to be active for the next three time slots. When the schedule is calculated, each node keeps track how many packets in total sub-trees rooted at it have remaining. When a sub-tree is assigned three time slots, all nodes in that sub-tree will transmit once and one packet will reach the root. Because of this, the root only tracks total number of packets remaining in each



(a) Linear network with six nodes and schedule created by Gandham *et. al.*



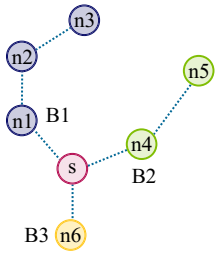
(b) Scheduule created for the same network by TreeMAC

Figure 26: Scheduling using algorithm proposed by Gandham *et. al.*.

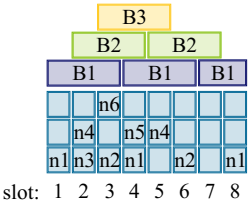
sub-tree, without the need to track packets at each node. This allows the distributed implementation of the algorithm. Details of this implementation are omitted here for brevity; they can be found in the original paper [51]. Minimal schedule length that can be achieved is $max(3n_k - 1, N)$, where n_k is the number of nodes in the largest top sub-tree, and N number of nodes in the network. Weather this minimum will be reached or not, depends on the network topology.

The example schedule calculated using this algorithm is shown on Fig. 27. For the sake of simplicity, this network has only three sub-trees, and there are no edges between the nodes in different sub-trees. The algorithm schedules the sub-tree $B1$ in the first time slot; the sub-tree $B2$ is then scheduled starting from the second time slot and $B3$ from the third. Each sub-tree is considered to be active for three time slots, even though it might finish transmitting earlier. For example, the sub-tree $B2$ is considered active from slots five to seven, even though it only transmits in fifth time slot. This approach results in unused time slots and slightly longer schedule length. However, thanks to these simplifications, the complexity is significantly reduced and the distributed implementations of the protocol is possible.

This protocol offers an improvement over TreeMAC without the need for a centralized implementation and significant increase of implementation complexity. However, when more complex algorithms are used, significantly better performance can be achieved. The improvement achieved by this algorithm compared to TreeMAC is relatively small. This is because this algorithm can only schedule whole sub-trees at the same time, and not individual parts of these sub-trees. For this to be possible, there must be no conflicts between any of the nodes from the two sub-trees. More advanced algorithms schedule individual nodes from different sub-tree at the same time, allowing higher number of parallel transmission. However, this requires knowledge of the topology of the whole network and comes at the cost of significantly increased complexity.



(a) Tree network with three independent sub-trees



(b) Schedule created for the tree network

Figure 27: Scheduling non-conflicted sub-trees in parallel.

3.1.3 Scheduling by Ergen and Varaiya

The scheduling algorithm by Ergen and Varaiya [57] uses a graph coloring algorithm to calculate the schedule. This approach requires a centralized implementation and knowledge about the network topology. However, this allows the algorithm to create a shorter schedule and consequently achieve better performance in terms of throughput and packet delivery latency. Before the schedule is calculated, an approximate minimal vertex coloring of the conflict graph is found; the schedule is then calculated based on this coloring. Ergen and Varaiya propose two versions of the algorithms, a node-based and level-based version. In the node-based version, colors are assigned to individual nodes, while in the level-based, all nodes on the same level get the same color.

The node coloring version starts with finding an approximate minimal graph coloring of the conflict graph; this is done using a heuristic graph coloring algorithm. Then, the schedule is calculated based on this coloring. After coloring the conflict graph, a trivial schedule can be calculated by associating each color with a time slot number. However, such a schedule would assign one time slot to each node, i.e., it is not adjusted to the traffic.

A schedule adapted to the traffic is created phase by phase. Every node that currently has a packet to transmit gets one time slot in each phase. That time slot is most commonly the one associated with the node's color. However, a time slot other than one associated with the node's color may be assigned to it as well when the algorithm detects that this will not cause a collision. Time slots assigned in one phase comprise a superslot. To determine which nodes have a packet in which phase, it is assumed that each node has one packet at the beginning. Then, the packets are followed as time slots are assigned to nodes. Finally, phases are repeated until all packets reach the sink.

The algorithm for creating superslot based on the conflict graph coloring is shown

in Algorithm. 1. To create a superslot, the algorithm goes through all colors (line 3). For each color, it creates a set of nodes that will get the time slot associated with that color (denoted by T). This set is comprised of nodes with current color that have a packet and additional nodes that can transmit with these nodes. Additional eligible nodes can exist when some nodes with the current color do not have a packet. This allows for the nodes with another color to temporarily (only in one phase) transmit with different colored nodes. The for loop in line 6 finds such nodes. This loop examines the set of alternative transmitters. Each node from this set is examined, and it is checked weather there is a conflict between that node and any of the nodes in T. If no conflict is found, the node is added to T (line 8).

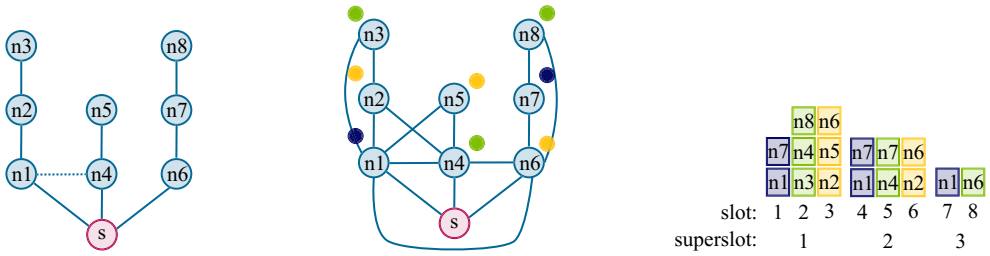
Algorithm 1 Scheduling nodes based on assigned colors

```
colors - set of colors assigned to nodes
slot = 0
for all c in colors do
    T = set of nodes with color c and a packet to transmit
    set_alt = set of nodes with color! = c and a packet to transmit
    for all n_i in set_alt do
        if no interference edge between n_i and nodes in T then
            T = T ∪ {n_i}
    if T ≠ ∅ then
        Assign slot slot to nodes in T
        slot = slot + 1
    if All packets reached the sink then
        break
```

The operation of the algorithm is illustrated using an example network displayed in Fig. 28a. In the figure, solid lines represent routing paths, and dashed lines represent links not used for routing (they are important for creating the interference graph). For creating the interference graph, the 2-hop interference model is used; the resulting conflict graph is shown in Fig. 28b. The results of coloring this graph using the heuristic algorithm used by Ergen and Varaiya are shown in the same figure (Fig. 28b). In this case, the nodes were colored using three colors.

The schedule produced by the algorithm based on this coloring is shown on Fig. 28c. Finding an alternative transmitter that can transmit with different colored nodes (line 6) can be illustrated using this example. For example, in slot five, nodes with green color are considered first. The only such node with a packet is node $n4$. The algorithm identifies node $n7$ (colored blue) as an alternative transmitter; it schedules it in slot five with green-colored nodes. As it can be seen from the interference graph (Fig. 28b), nodes $n4$ and $n7$ have no edges in between them. Therefore, they can transmit in slot

five despite having different colors.



(a) Network

(b) Corresponding interference graph and assigned colors

(c) Time schedule

Figure 28: Interference graph creation and color assignment.

The level-based algorithm first forms the level conflict graph and then calculates the schedule based on the coloring of this graph. In the level conflict graph, each vertex corresponds to all nodes at one network's level. An edge between two vertexes in the level conflict graph exists if there is an edge in the network conflict graph between any two nodes at levels represented by these two vertexes. The schedule is calculated in a similar way as in the node-based version. In each slot, levels corresponding to the current color are scheduled. However, not all the nodes from the same level can be scheduled simultaneously. Instead, the algorithm tries to schedule as many nodes from the same level as possible. The exact algorithm is not given here; it can be found in the original paper by Ergen and Varaiya [57].

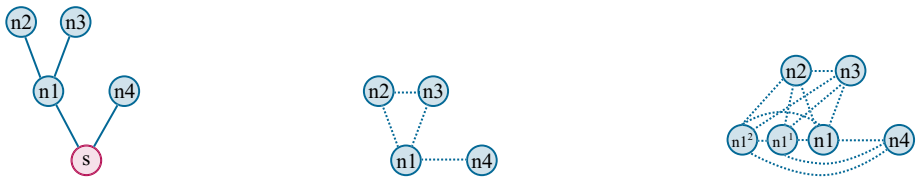
The algorithm by Ergen and Varaiya offers improved performance compared to distributed approaches at the cost of the increased complexity and longer network construction time. The improvement in terms of performance is dependent on the network topology and size. This dependency is analyzed in Section 3.3. In that section, it is compared with other distributed approaches, and a guide for choosing the most suitable algorithm based on the application is proposed.

3.1.4 Scheduling by Lai *et. al.*

The scheduler proposed by Lai *et. al.* [13] uses a graph coloring algorithm that colors each node with an arbitrary number of colors; this number is set to be equal to the number of slots required by the node. Multiple colors are assigned to a node by adding a virtual node for each additional color; the resulting conflict graph is called the extended conflict graph. After the extended conflict graph is colored, the schedule is formed by associating each color with a time slot.

The construction of an extended conflict graph is illustrated using an example network (Fig. 29). To form this graph, first, additional (virtual nodes are added). For example, node $n1$ needs three time slots; therefore, two virtual nodes corresponding to it are added, nodes $n1^1$ and $n1^2$. Then, additional edges are added. Firstly, an edge is added between every virtual node and the node it corresponds to and other virtual nodes corresponding to the same node. In the example (Fig. 29), an edge is added between virtual node $n1^2$ and the node it corresponds to, node $n1$. Additionally, an edge is added between $n1^2$ and the other virtual node corresponding to node $n1$, node $n1^1$.

After connecting virtual nodes with their corresponding node, edges are added between virtual nodes and other nodes. To do so, a list of neighbors of the corresponding node is created. If these neighbors have virtual nodes, they are also added to the list. Then, virtual nodes of the corresponding node are connected with all nodes from the list. In the example (Fig. 29), node $n1$ is connected with nodes $n2$, $n3$ and $n4$; therefore, an edge is added between virtual nodes corresponding to it ($n1^1$ and $n1^2$) and these three nodes.



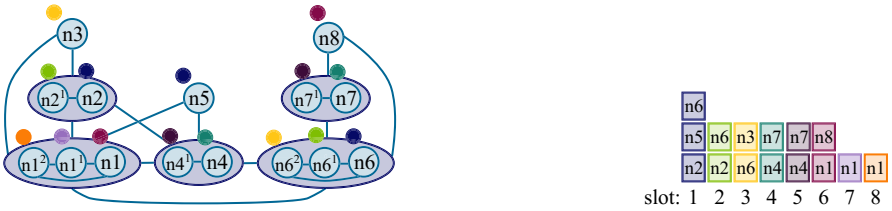
(a) Network (b) Conflict graph (c) Expanded conflict graph

Figure 29: Interference graph creation and color assignment.

Coloring the extended conflict graph is performed using a more complex algorithm than the one used by Ergen and Varaiya. Instead of using a degree-based heuristic, the coloring order is chosen in the following way. In the beginning, the node with the lowest degree is selected and removed from the graph. Then, the degree of each node in this newly formed graph is calculated. Again, the node with the lowest degree is removed. This process is repeated until all nodes are removed. The coloring order is reverse from the order in which the nodes were removed from the graph. This means that the node removed the last will be assigned a color the first. This coloring method colors a graph G using at most $\delta(G) + 1$ colors. The value $\delta(G) + 1$ is based on the sub-graph of G, in which the minimum degree of any vertex is maximal. The value of $\delta(G) + 1$ is equal to the minimal degree any vertex has in that sub-graph. The time complexity of this algorithm is $\mathcal{O}(|V| + |E|)$.

An example schedule created by this algorithm is shown in Fig. 30. This sched-

ule is calculated for the same sensor network used to illustrate the algorithm Ergen and Varaiya; this network is shown in Fig. 28a. The extended conflict graph and its coloring calculated using the algorithm by Lai is shown in Fig. 30a. The illustration of the conflict graph is simplified by grouping nodes with virtual nodes corresponding to them. The schedule based on this coloring is shown in Fig. 30b. It is eight slots long; the same length was obtained using the algorithm by Ergen and Varaiya.



(a) Expanded conflict graph and assigned colors (b) Time schedule

Figure 30: Extended conflict graph and color assignment.

In the case of larger, more complex networks, this algorithm commonly results in a shorter schedule length than the algorithm by Ergen and Varaiya. This can be concluded from simulation results presented in Section 3.3. However, this algorithm does not consider traffic flow; this can result in worse performance despite a shorter schedule. A common problem when a schedule calculated by this algorithm is used is buffer overflow. This can be illustrated using the example in Fig. 30. For example, node $n1$ transmits three times in a row at the beginning of the cycle. Then, until the next cycle, it collects all packets from above; in this case, that is only three packets, and a buffer overflow is unlikely. However, in a larger network, a node might be scheduled in the same way, requiring it to collect a larger number of packets than its buffer size before transmitting them.

The algorithm by Lai *et. al.* uses a more advanced strategy for assigning multiple time slots to nodes, resulting in shorter schedule lengths than the algorithm by Ergen and Varaiya. However, it does not consider the traffic flow, potentially resulting in degraded performance. This can result in a very high packet delivery time or, even worse, a low packet delivery ratio if the schedule unadjusted to the traffic flow causes the buffer of some nodes to overflow. The lack of consideration of the traffic flow is especially an issue in networks with a large height. Therefore, the application of this algorithm should be limited to networks with a low height. In such networks, shorter schedule results in better performance, despite the lack of traffic flow adjustment.

3.1.5 Scheduling by Park *et. al.*

Park *et. al.* approach the scheduling problem directly, rather than transforming it into the graph coloring problem [12]. This approach allows the algorithm to directly consider the traffic flow and the required number of slots for individual nodes. Furthermore, they prioritize the largest top sub-tree when assigning time slots. This priority maximizes the benefits of parallel scheduling, as this sub-tree is the most critical.

The pseudocode describing this algorithm is shown as algorithm. 2. Unlike graph coloring algorithms, where nodes are traversed and colors are assigned to them, this algorithm goes through time slots and tries to schedule as many nodes as possible in each slot. This is done using the while loop in line 2. For each time slot, the algorithm examines all the nodes in the network and tries to assign them to transmit in that time slot. The array $CntData$ is used to track each node's current number of packets. Each element of this array corresponds to the node with the same index as the array element; the sink corresponds to the first element of the array (index value of zero). Hence, the while loop in line 2 is executed until all packets reach the sink.

Algorithm 2 Schedule calculation algorithm

```
slot_no ← 0; CntData ← {0, 1...1}
while CntData[0] < N do
    IF = {0...0}
    assignSlot(slot_no, sink)
    slot_no ← slot_no + 1
function assignSlot(slot_no, v_i)
    if v_i = sink then
        v_i ← neighbor with maximal number of descendants
        assignSlot(slot_no, v)
    else
        if IF[i] = 0 and CntData[i] > 0 then
            assignSlotToNode(slot_no, v_i)
            j ← Index(Par(v))
            CntData[i] ← CntData[i] − 1
            CntData[j] ← CntData[j] + 1
            for all v_j in TwoHop(v) do
                IF[j] ← 1
        if Cnt(C(v)) = 0 then return
        for all v in C(v_i) do
            assignSlot(slot_no, v)
```

Before trying to assign any node in a time slot, the algorithm first initializes array IF to all zeros (line 3). This array tracks which nodes can still be scheduled in the

current time slot without interfering with already scheduled nodes; value zero means that the corresponding node can be scheduled (zero symbolizes no interference). After this array is initialized, function *assignSlot* is called in line 4; this function schedules as many nodes as possible in this time slot.

The function *assignSlot* is given in line 6 of the algorithm. The pseudocode is simplified by representing a node with index i as v_i. Similarly, a collection of node's children is represented as $C(v)$ and node's parent as $Par(v)$. Nodes in list $C(v)$ are sorted in the descending order of the number of descendants. The number of descendants of a node is equal to the number of nodes in the sub-tree rooted at that node.

Function *assignSlot* is called recursively to traverse the whole network. The function tries to schedule the node passed as the first argument first and then selects the next node to be examined. When the sink is passed as the first argument, the sink is not scheduled; instead, the sink's children are ordered in the descending number of descendants. Then, *assignSlot* function is called on each of them, starting from the first one in the list. When the node passed as the first argument is not the sink, it is checked if it can be scheduled in the current slot (line 11). If the node does not interfere with scheduled nodes and has a packet, it is scheduled in the current slot. Afterward, the next node to be traversed is selected, regardless of whether the node was scheduled (line 20). The next node to be traversed is always the child node with the highest number of descendants that was not covered yet. When the leaf node is reached, the function returns (line 18). In this way, the network is traversed in a depth-first manner, covering the larger sub-tree first at each branching.

After a node is scheduled in line 12, nodes in its two-hop neighborhood are marked as not eligible to be scheduled in the current slot. This is done by setting the corresponding element in the IF array to one. Then, the number of packets the current node has is reduced (line 14), and the number its parent has increased (line 15).

The schedule created using this algorithm for the network from Fig. 28a is shown in Fig. 31. As the algorithm first tries to schedule the largest sub-tree, node $n1$, the root of the largest sub-tree is scheduled in the first time slot. The algorithm does not stop at node $n1$. Instead, it goes through the whole network and tries to schedule additional nodes in this slot. It succeeds in scheduling nodes $n5$ and $n6$ to share the first slot with node $n1$. Next, slot two is examined. Node $n1$ is checked first again; however, it does not have a packet to transmit at this point. Therefore, its child, node $n2$, is scheduled together with node $n7$ in the second time slot. The rest of the schedule is calculated in a similar way.

Compared to Ergen and Varaiya and the other similar algorithms, this algorithm considers an additional network property, the sizes of top sub-trees. However, this

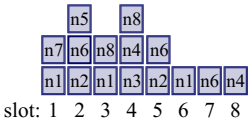


Figure 31: Schedule created using algorithm by Park *et. al.*

comes at the cost of less efficient heuristics since graph coloring allows the usage of some advanced and more efficient heuristics, like the one used by Lai *et. all.* [13]. Nevertheless, as the simulation results presented in Section 3.3 show, this trade-off is justified, and this algorithm performs very well in most cases.

3.1.6 Scheduling by Tsai and Chen

The schedule creation algorithm proposed by Tsai and Chen [58] approaches the scheduling problem directly, similar to the algorithm by Park *et. al.* Unline Park *et. al.*, this algorithm schedules nodes level by level, in contrast to prioritizing the nodes close to the sink. The algorithm by Tsai and Chen starts with the schedule length of three slots. Then it goes through all the nodes and tries to assign one of the three slots to them. Afterward, the schedule is extended by three additional time slots, and the process is repeated. In contrast, Park *et. al.* extend the schedule one slot at a time and schedule as many nodes as possible in the newly added slot.

Tsai and Chen assing states to nodes to keep track of interference, rather than using a conflict graph like most algorithms. These states and their meaning are shown in Table 1. When a node is scheduled, the states of that node, receiver, and neighbors are updated. Neighbors of the receiver are set to state r; this means they can not be scheduled to transmit in the current time slot. They can, however, receive in the current time slot if the transmitter is not in the first receiver's range. Neighbors of the transmitter, other than the receiver, are set to state t. This state allows them to transmit, back to back, with the first transmitter.

Table 1: Node states and state descriptions

State	Description
T	Node is transmitting
R	Node is receiving
t	Node can only transmit
r	Node can only receive
I	Node can neither transmit nor receive
F	Node can both transmit and receive

This algorithm creates the schedule period by period. In the original paper, a period of size three is considered. The algorithm initializes the schedule length to the length of one period, which is three by default. Then, it goes through all levels, starting from the first, and tries to schedule each node in one of the slots from the first period. After examining all nodes, an additional period is added to the schedule, and the process is repeated. The algorithm terminates when all packets have reached the sink.

Time schedule calculation using this algorithm is illustrated in Fig. 32. The figure illustrates the schedule calculation for the first three time slots, i.e., the first period. The calculation was performed for the same sensor network used to demonstrate other algorithms; this network is displayed in Fig. 32. The algorithm traverses the network graph level by level and tries to schedule each node in one of the three time slots. Node $n1$ is traversed the first and scheduled in the first slot. The remaining nodes at level one, nodes $n4$ and $n6$, are examined next and scheduled in slots two and three, respectively. The schedule and node states after processing the first level are shown in Fig. 32a.

slot	s	n1	n2	n3	n4	n5	n6	n7	n8
1	R	T	t		I		r		
2	R	I			T	t			
3	R	r			r		T	t	

(a) Slots allocation after processing first level

slot	s	n1	n2	n3	n4	n5	n6	n7	n8
1	R	T	t		I		R	T	t
2	R	I			T	t			
3	R	R	T		R	T	T	t	

(b) Allocation after processing second level

slot	s	n1	n2	n3	n4	n5	n6	n7	n8
1	R	T	t		I		R	T	
2	R	I	R	T	T	t		R	T
3	R	r	T		R	T	T	t	

(c) Slots allocation after processing third level

Figure 32: Slot allocation using the algorithm by Tsai and Chen. Blank space means that node is in the state F.

Then, the algorithm moves to the second level and starts by scheduling node $n2$. To illustrate how the states are used to prevent collisions, selecting a time slot for this node is explained in detail. Firstly, the algorithm checks if node $n2$ can transmit in the first slot; this is not possible because its receiver, node $n1$ is in state T. Therefore, the algorithm moves to the next slot. As node $n2$ can not be scheduled in the second slot because node $n1$ is in state I, the algorithm finally cheks if node $n2$ can transmit in the third slot. Here, node $n1$ is in state r, meaning it can receive the packet, and node $n2$ is scheduled to transmit.

The remaining nodes are processed in a similar way. Fig. 32c shows the schedule

and the states of nodes after the scheduling of the first period is completed. In this case, every node from the network was scheduled to transmit once during the first period. The complete schedule is shown in Fig. 33b. For this example network, this algorithm achieves the optimal schedule length of eight.

(a) Network graph

(b) Time schedule

Figure 33: Network graph and time schedule calculated using algorithm by Tsai and Chen.

The algorithm by Tsai and Chen is similar to the algorithm by Park *er. al.*. Like other centralized schedule calculation algorithms, it is complex to implement and requires the sink to gather information about network topology, resulting in a long network set-up time. Theoretically, the shortest schedule length this and other centralized algorithms can achieve are similar.

3.2 Implementation of TDMA Protocols in INET Framework for OMNeT++

The INET framework for OMNeT++ provides the structure and libraries necessary to implement and simulate complexly wired and wireless protocols. This framework organizes the network stack in the same way the internet network stack is organized. Therefore, all implementations in this framework must fit into that frame. In this framework, each layer is represented with a module; each model is composed of different sub-modules, each performing a different, predefined function.

TDMA MAC protocols use a schedule calculated based on the network topology. Therefore, implementing such a protocol within a network stack that separates the network and link layer is not straightforward. The approach chosen here is to implement schedule calculation in the network layer and slot tracking in the MAC layer. Such a division allows the implementation of a universal TDMA MAC protocol; the universal MAC waits for the schedule calculated by the network layer before starting the slotted operation phase. The universal MAC also contains a simplified CSMA-CA random access protocol used before the slotted operation is started. When the network layer performs network discovery and schedule dissipation, the universal TDMA MAC is in

random-access mode and uses this CSMA-CA implementation to transmit packets.

The protocols described in Section 3.1 are composed of centralized and distributed protocols. Each of them requires different information about network topology to compute the schedule. Some of them need to construct the complete network graph at the sink. Others do not gather topology information at the sink; instead, they calculate the schedule in a distributed manner, using the information gathered during the network construction phase. Nevertheless, all of them can be implemented in a fully centralized manner. In such an implementation, the sink gathers information about the network topology, calculates the schedule, and distributes it to all nodes. This results in a longer network set-up time and greater message complexity; however, protocol performance remains unchanged during the slotted operation phase.

This comparative evaluation of state-of-the-art TDMA protocols aims to compare their performance during the slotted access phase, i.e., to compare throughput, packet latency, and packet delivery fairness each protocol can achieve in this phase. Hence, a centralized implementation approach will not affect simulation and comparison results. On the other hand, a centralized approach allows using of the same network discovery and schedule dissipation protocol for simulating all schedule calculation algorithms described in Section 3.1. This simplifies both the implementation and simulations, shortens the time required for development, and allows to quickly update the results if a new schedule calculation algorithm appears. Therefore, considering all mentioned advantages, the centralized approach is chosen.

3.2.1 INET Framework Overview

INET Framework provides an extensive library of protocols, physical medium and radio models, and many modules, which can be used to simulate wireless sensor networks in the OMNeT++ simulator. INET contains an implementation of the internet stack protocols, such as TCP, IDP, IPv4, and IPv6, and many standard MAC protocols, such as Ethernet, IEEE 802.11, and CSMA-CA. However, it lacks more complex MAC protocols often used in wireless sensor networks, such as TDMA protocols.

The INET framework divides protocols into layers; each layer is represented by a module. Modules may contain different sub-modules. The organization of the network stack in INET is shown in Fig. 34. The lowest module represents the wireless interface; it contains a radio module, a MAC protocol, and a link-layer control protocol. The radio module contains antenna, modulation, and signal propagation models. The network layer is located above the wireless interface; it contains a routing table, a route calculation protocol, and an address resolution protocol, which are all implemented as separate modules. Finally, a transport protocol and applications are located above

the network layer. INET supports multiple applications running simultaneously, each implemented as a separate module.

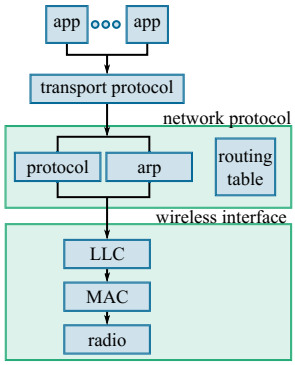


Figure 34: The organization of the network stack in INET framework for OMNeT++.

There are two ways modules can communicate in INET; using packets and messages. Packets represent data that should be transmitted using radios. On the other hand, messages are used for sending commands from one module to the other. Since modules in INET represent protocols, protocols located at different layers in the stack communicate with each other using messages.

The library of the INET framework provides multiple radio signal models. The simplest model available is the unit disc model. This model represents radio transmission with two parameters: the transmission and interference range. All radios within the transmission range will hear the transmission; if there is no interference, they can receive the transmitted packet successfully. On the other hand, a transmission will interfere with receptions at all radios within the transmitter's interference range. Though simple, this model offers sufficient accuracy in many cases.

More complex models take path-loss and radio modulation into consideration to provide results closer to real-world performance, especially in terms of interference. One such model, the scalar radio model, models a radio signal with a scalar value representing its strength. At each receiver, signal to noise ratio is calculated based on received signal strength, the strength of all interfering transmissions, and background noise value. Once the signal-to-noise ratio is calculated, the packet loss ratio is obtained based on its dependence on the signal-to-noise ratio; this dependence is a property of the used modulation scheme and exists within the framework. The most complex model provided, the dimensional radio model, extends this model by considering the signal shape. It calculates SNR at the receiver based on the shapes of the receiving and all interfering signals.

3.2.2 Network Discovery

A modified Dijkstra's algorithm is used for network discovery and routing tree construction. The standard version of Dijkstra's algorithm assumes that nodes have discovered neighbors before running the algorithm. Therefore, a neighbor discovery must be performed before starting the tree construction. The modified version used in this thesis unifies neighborhood discovery with tree construction. Furthermore, Dijkstra's algorithm does not deliver topology information to the sink, which some simulated protocols need. Therefore, the version used modifies protocol-specific packets to include topology information and deliver it to the sink.

The tree construction is performed level by level. The construction of each except the first level is initiated by the sink broadcasting a $Start_p$ packet, where p signifies the level. To discover and construct the first level, the sink broadcasts the $Query_1$ packet. When neighbors of the sink receive this packet, they set their level to 1 and sink as their parent. Then, they respond by sending a *QueryResponce* packet to the sink. Sink receives such a packet from each neighbor, saves the address, and creates a list containing the address of all neighbors.

Next, the sink initiates the discovery of the second level by broadcasting a $Start_2$ packet. When neighbors of the sink receive this packet, they broadcast a $Query_2$ packet as the response. The role of this packet is twofold; nodes at the first level use it to discover nodes at the second level and create a list of their same-level neighbors. Nodes that receive a $Query_2$ packet respond with a *QueryResponce*; this packet contains a $SameLevelNode$ flag. A $Query_2$ packet is always transmitted by nodes at the first level. When a node at level one receives this packet, it responds with a *QueryResponce* packet with the $SameLevelNode$ flag set, as it is located at the same level as the sender; nodes at level two will leave this flag cleared in their response.

A node located at the second level will receive a $Query_2$ packet from all its neighbors located at the first level. When a node at the second level receives the first $Query_2$ packet, it sets its parent to be the transmitter of the packet. Then it responds with a *QueryResponce* packet; this packet has another flag, flag $Child$, to indicate if the responding node has chosen the transmitter as the parent. In this case, the node sets this flag in its response. When the node responds to $Query_2$ packets from other neighbors at the first level, it leaves this flag cleared. At the same time, this flag is used by $Query_2$ transmitters to create the list of their children.

The construction of the second level is illustrated in Fig. 35a. The construction of this level was initiated by the sink transmitting $Start_2$ packet, which was received by nodes $n1$ and $n2$. After receiving this packet, these nodes compete for medium access to transmit a $Query_2$ packet. The figure assumes that node $n1$ won the competition and

transmitted a $Query_2$ packet the first. The figure only shows $QueryResponce$ packets received by nodes $n1$ and $n2$ as resonances to transmitting $Query_2$ packets. The name of this packet was shortened to $QRes$ on the figure, and flags $SameLevelNode$ and $Child$ are shortened to SL and C, respectively.

(a) Level two construction

(b) Level three construction

Figure 35: Query response packets transmitted during construction and discovery of the second and the third tree level. These packets have two data fields, SL (same level node) and C (number of children nodes).

Node $n1$ is the first one to transmit $Query_2$ packet. As a response, it receives two $QueryResponce$ packets. The first one is from node $n3$, located at the same level as node $n1$. This is the first $Query_2$ packet received by node $n3$; therefore, its response to node $n1$ has the $Child$ flag set. The second $QueryResponce$ node $n1$ receives is from node $n2$. Node $n2$ is located at level one, same level as the node $n1$; therefore, this $QueryResponce$ has the $SameLevelNode$ falg set and the $Child$ flag cleared. Ather the node $n1$, node $n2$ transmits its $Query_2$ packet, and gets a $QueryResponce$ from nodes $n1$, $n3$, and $n4$. It is assumed that node $n3$ received a $Query_2$ from node $n1$ first. Therefore, the $QueryResponce$ packet node $n3$ sends to node $n2$ has the $Child$ flag cleared.

After collecting all $QueryResponce$ packets, a node sends a $StartResponce$ packet to the sink. This packet replaces the *echo* packet in the standard Dijkstra's algorithm, used to notify the sink that the current level has been processed. To deliver network topology information to the sink, this packet includes an array containing children and an array containing neighbors nodes. After receiving $StartResponce$ packets from all nodes on the second level, the sink broadcasts a $Start_3$ packet to initiate the construction of the third level.

The third level is discovered in the same way. First, the $Start_3$ packet, broadcast by the sink, is received by nodes at the first level. Then, they forward it to the nodes at the second level, which, in turn, broadcast a $Query_3$ packet. Afterward, nodes at the second level collect and process $QueryResponce$ packets. Finally, after collecting

all responses, they send a *StartResponce* packet to the sink; these packets are shown in Fig. 35b.

The higher levels are discovered in the same way. In general, when a node at level a receives $Start_b$ packet it will forward it to children if $b < a - 1$ or send $Query_b$ if $b = a - 1$. After collecting *StartResponce* packets, *StartResponce* packet is sent to the sink. *StartResponce* packets contain the list of sender node neighbors located at the next level. The algorithms terminate when no new nodes are discovered at a certain level (detected by examining lists of neighbor nodes in *StartResponce* packets).

3.2.3 Packet Routing

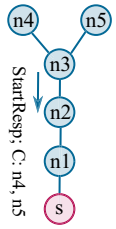
The modified Djikstra's algorithm used for network construction broadcasts packets ($Start_b$ packets) until they reach a certain level. This eliminates the need for upstream (from sink to nodes) routing during this phase. For downstream routing, used to send *DisoveryResponce* packets to the sink, parent-children assignment is used. Even though not needed for tree construction, upstream routing is needed for dissipating schedules from the sink to the nodes and for more advanced routing algorithms, studied later in Section 4. Therefore, the tree construction algorithm is modified to provide upstream routing. The modification only creates routing tables for routing packets from the sink to nodes or from a node to nodes in its sub-tree. This is, however, sufficient for most of the applications for convergecast sensor networks.

StartResponce packets are used for creating and updating routing tables; these packets contain the list of the sender's children. When a node receives a *StartResponce* packet, it marks that the children in this list can be reached through the node that forwarded the *StartResponce* packet. Therefore, when new nodes are discovered, they become children of the node that discovered them. Then, that node sends a *StartResponce* packet to the sink, notifying all nodes on the path to the sink that they can reach these nodes through it.

An example showing how a *StartResponce* packet modifies routing tables on its path is shown in Fig. 36. In the example, node $n3$ discovers node $n4$ and $n5$ and sends a *StartRespnce* packet to the sink, containing these two nodes as its children. When node $n2$ receives this packet, it adds an entry for these two destination nodes and sets $n3$ as the next hop for reaching them. Similarly, node $n1$ adds these two nodes; however, it sets node $n2$ as the next hop, as node $n2$ forwarded the *StartRespnce* packet to it.

destination	next hop
n2	n2
n3	n2
n4	n2
n5	n2

(a) Routing table of node $n1$



(b) Adding routing table entries using a *StartResponce* packet.

Figure 36: An example routing table and the usage of a *StartResponce* packet to add new entries.

3.2.4 Schedule Dissipation and Start of the Slotted Access Phase

Once the network discovery is completed, the sink has gathered network topology information and constructed the network graph based on it; the sink can calculate the schedule based on this information. At the same time, routing tables for upstream routing are created at all nodes in the network, allowing the sink to send packets to sensor nodes. The network is now ready to switch to the slotted operation mode.

When switching to the slotted operation mode, the first step is delivering the schedule to nodes. The sink does this by sending a $TdmaConfigPkt$ to each node. A $TdmaConfigPkt$ packet contains transmission slots of the target node and cycle length; this information is sufficient to configure the TDMA MAC layer. When a node receives this packet, it configures its universal TDMA MAC by sending a $TdmaConfigCmd$ message to it. The definition of this command is given in Fig. 37. The command contains the cycle length and an array with transmission slots numbers ($txSlost$), indicating the slots in which the target node may transmit. Other than that, it contains flag *sink*, informing the MAC if the network layer has the role of the sink, and a few parameters for configuring acknowledgment mode.

```
class TdmaConfigCommand extends TdmaCommand
{
    bool sink = false;

    int cycleLength;
    int txSlots[];

    bool useAck = true;
    bool piggybackAck = true;

    int ackTimeoutMode;
    int ackTimeoutVal;
    int ackSlots[];
}
```

Figure 37: Implementation of $TdmaConfigCmd$ command in INET.

A $TdmaConfigCmd$ delivers schedule information to the MAC layer and configures all parameters necessary to start the slotted operation mode. However, it leaves the node in the random access mode. To start the slotted operation mode, the network layer must first configure the MAC module for this mode and then start it using a $TdmaStartCmd$. The sink initiates the transition from the random access to the slotted operation mode. It first sends $TdmaStart$ command to its own MAC layer, switching it to the slotted operation mode. Then it sends $TdmaStart$ packet to its neighbors. When a neighbor node receives this packet, it first switches its own MAC layer to the slotted operation mode. Then, it responds with a $TdmaStartAck$ packet, as illustrated in Fig. 38.

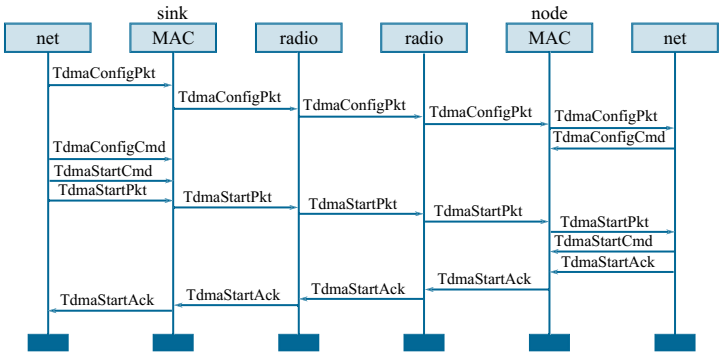


Figure 38: Commands and packets exchanged when starting slotted operation mode.

3.2.5 Random Network Generation

The efficiency of a schedule calculation algorithm is strongly dependent on the network topology. To provide a comparative evaluation of the described schedule calculation algorithms, the algorithm should be tested using an extensive set of networks with different sizes, densities, and deployment area shapes. An application for generating random sensor networks is developed to create such a set of networks. Three deployment area shapes are used; circular, narrow rectangular, and broad rectangular. In the cases of the circular shape, two sink positions were used, in the network center and on edge. For each deployment area, networks containing a different number of nodes are generated.

Before nodes are placed randomly onto the deployment area, anchor nodes are added at the deployment area boundary. The purpose of the anchor nodes is to spread across all parts of the deployment area. The algorithm aims to generate a network consisting of a defined number of nodes. The number of nodes in the network will be referred to as network size. Since nodes are placed randomly, not all placed nodes will be part

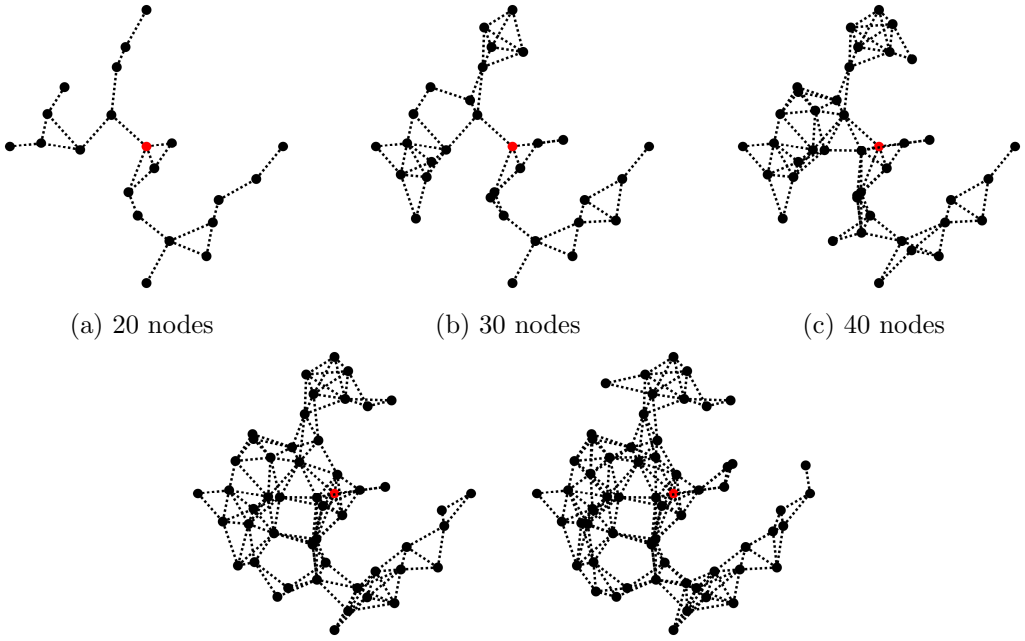
of the network. Some of the nodes will be isolated, ie., they will be no other nodes in their radio range, or no possibility to forward packets to the sink. Such nodes are called unconnected nodes and they are removed after generating the network.

As the number of generated nodes is increased, the size of the network does not increase linearly. Instead, it might remain unchanged, when the newly added node is isolated, or increase for a value larger than one, when the newly added node connects multiple previously isolated nodes with the rest of the network. Therefore, generating a network of desired size is not straightforward. To achieve the desired size, the following algorithm is used. After placing anchor nodes, nodes are placed at random coordinates until the network size reaches value greater or equal to the desired network size. If the network sizes is equal to the desired size at this point, the algorithm terminates. Otherwise, all added nodes are removed, and process is repeated without resting the random number generator.

As a part of protocol evaluation, dependence of average maximum throughput and schedule length is calculated. To reduce random noise from this dependence, networks of larger sizes are generated by starting from a smaller network and adding new nodes randomly to increase the size. This is illustrated on Fig. 39. The figure shows five circular networks, with sink located in the middle. Four anchor nodes were used. The first one has the size of 20 nodes. The second one, consisting of 30 nodes, is generated by adding an additional 10 nodes to the first network. This is repeated with the step of ten, ending at 60 nodes. The dots on the figure represent nodes. Red dot represents the sink, black dots other nodes. Dashed lines connect nodes that are within each others radio range. The radius of this network is 280m and radio range is 100m.

The algorithm evaluation requires the usage of many random networks with the same parameters (size, shape, sink location). The set used in this thesis contains ten random networks for each set of parameters. Since networks of larger sizes are created based on those with a smaller size, the process of increasing network size is repeated. In the case of the circular deployment area, 50 random networks are generated in this way (10 for each number of nodes). Fig. 40 shows nine out of ten networks of size 40 used. The same was repeated for topology with the circular deployment area and sink located at the edge of the network.

Networks with two different rectangular deployment areas are generated. The first one has a width of 250m and a length of 500 meters (referred to as wide rectangular). The second deployment area has a smaller width, equal to 100m, and a length of 800m (refereed to as narrow rectangular). The radio range value used is 100 meters, the same as in the case of circular networks. Fig. 41 shows two of the generated rectangular networks. The one in Fig. 41a is a wide rectangular, while Fig. 41b shows one of the



(a) 20 nodes (b) 30 nodes (c) 40 nodes

(d) 50 nodes (e) 60 nodes

Figure 39: Circular networks of different sizes with sink in the middle.

narrow rectangular networks.

3.3 Simulations and Results

Randomly generated networks described in Section 3.2.5 are used to evaluate and compare schedule calculation algorithms described in Section 3.1. In total, 210 different networks are used, with four different topologies. The simulations are performed in OMNeT++. The simulations start with the sink performing discovery of the network. Then, it calculates the schedule and distributes it to the node. After this, the TDMA operation is started. Traffic generation is paused during the discovery and start phases; it begins once the network switches to the slotted operation mode.

For the simulation results to be relevant, data traffic must be random and resemble traffic that can be expected in a real-world application. This is especially important because scheduling algorithms take care of the traffic flow in the network and the traffic flow is dependent on the availability and number of packets at different nodes. For this purpose, an event-based traffic generation algorithm is used. This model schedules traffic by generating events at a random time and place. Each event has a radius; when the event is triggered, a packet is generated in each node within the radius. The simulations used the event radius value of 100m.

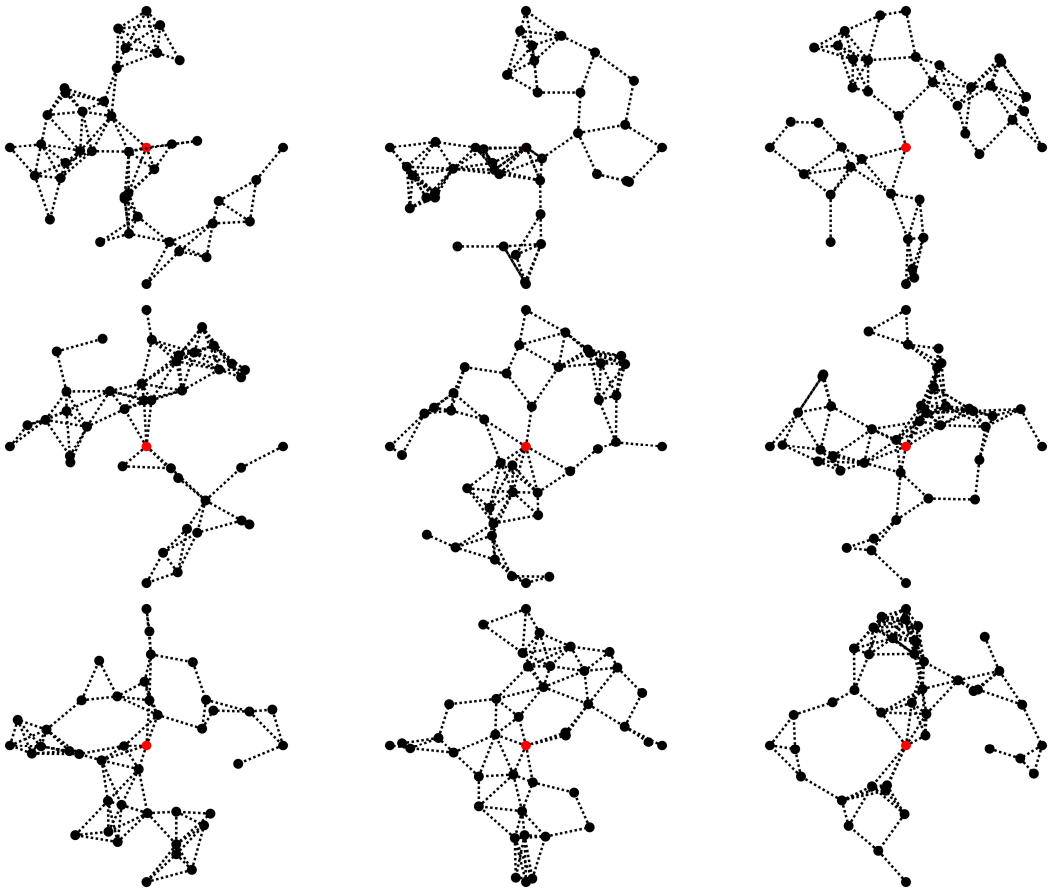
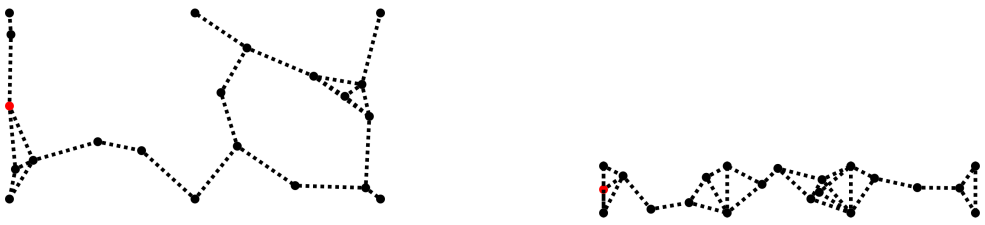


Figure 40: Nine different circular networks comprised of 40 nodes and one sink in the middle.

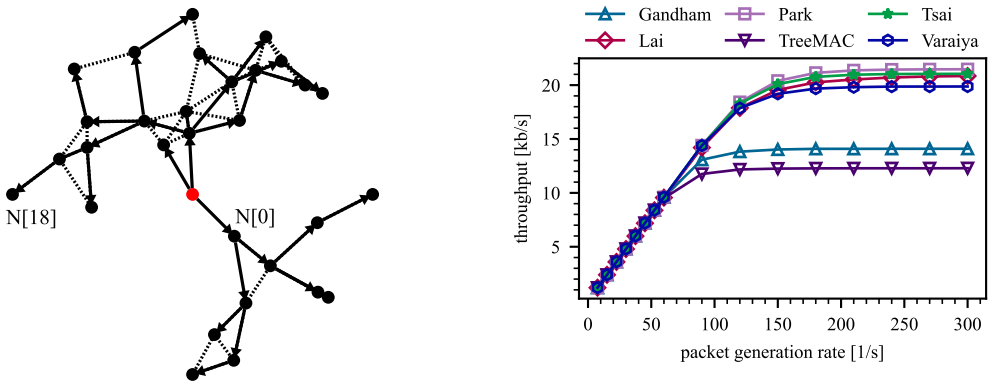


(a) Wide network

(b) Narrow network

Figure 41: One of ten wide and narrow rectangular networks with size 20 used for simulation.

The simulation results are first analyzed and explained using a single circular network that consists of 30 nodes. The radius of this network is 280m, and the sink node is positioned in the center of the network. The network and child allocation are shown in Fig. 42a. Full lines ending with arrows on the figure connect parent nodes with their children. Dashed lines connect nodes that are within each other's radio range but are not connected in the routing tree. To evaluate the performance of different schedule calculation algorithms, the packet generation rate is varied in the range from 20 to 400 packets per second (equivalent to 1 to 20 packets per second per node).



(a) Network (b) Throughput

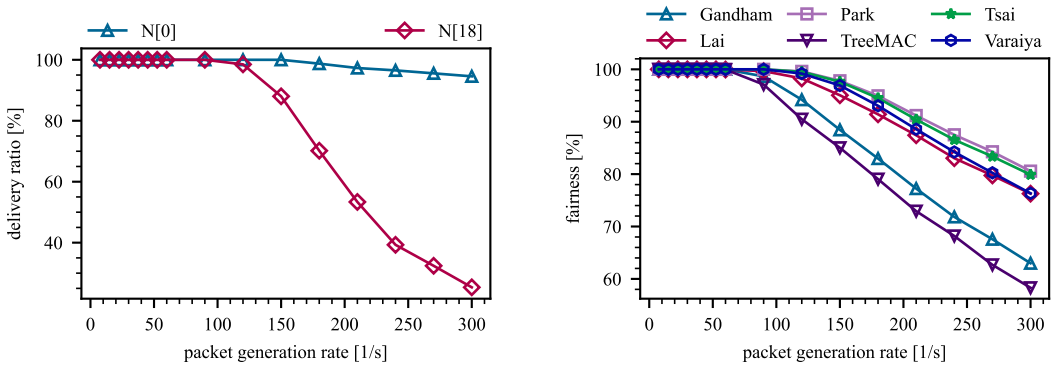
Figure 42: Example circular network comprised of 30 nodes and simulation results.

The dependence of the throughput at the network sink on packet generation rate, when different schedule calculation algorithms are used, is shown in Fig. 42b. When packet generation rate is low, throughput increases linearly with packet generation rate. However, when the packet generation rate is further increased, the throughput increase slows down, and the dependence stops being linear. At one point, the throughput reaches a threshold value. After this point, the throughput value remains constant when the packet generation rate is further increased.

This dependence is significantly different than in the case of random access protocols. When random access protocols are used, the throughput will start dropping with an increased packet generation rate at one point. This happens due to an increased number of collisions. However, when TDMA protocols are used, slotted access prevents collisions. Hence the different behavior at high throughput values. When TDMA MAC is used, the throughput increases until the number of incoming packets allow the usage of all slots. At this point, the maximum throughput value is reached. When the packet generation rate increases beyond this point, the throughput does not drop. However, the network will not be able to transmit all packets, causing their accumulation in

buffers. Eventually, buffers will overflow, and packets will be lost.

Since the traffic is random, some packets will be dropped even at values of packet generation rate lower than the maximum one supported by the network. The chance of a packet being dropped increases with the number of hops they need to travel to reach the sink. Because of this, even when the network packet loss ratio is very low, packets from some nodes may exhibit a significant packet loss. Fig. 43a illustrates how big the difference in the packet loss ratio for packets originating from different nodes can be. The figure shows the dependence of packet delivery ratio on packet generation rate for nodes $N[0]$, one hop from the sink, and $N[18]$, five hops from the sink. These two nodes are marked in Fig. 42a.



(a) Packet delivery ratio

(b) Fairness

Figure 43: Simulation results.

The throughput dependence from Fig. 42b shows that all centralized algorithms achieve similar maximal throughput values in this case. Based on this figure, it appears that maximal throughput could be a good metric for comparing different schedule calculation algorithms. However, as illustrated in the previous example, even though total throughput is high, the packet loss rate for the packet originating from certain nodes might be high. Therefore, traffic fairness, as a metric of difference between packet loss ratio among nodes in the network, should also be considered. Fig. 43b shows this network's dependence on fairness from packet generation rate. The figure shows that a higher maximal throughput does not mean better fairness in the network. For example, the algorithm from Lai *et. all.* achieves higher maximal throughput than the algorithm by Ergen and Varaiya (Fig. 42b); however, it achieves considerably lower fairness (Fig. 43b).

Based on these observations, the maximal packet generation rate a network can support is defined as a packet generation rate value at which traffic fairness drops to

95%. In this case, fairness is calculated using the metric proposed in [32]; this metric was previously explained in Section 2.3. Form Figs 43a and 43b, it can be observed that when the packet generation rate reaches a value of 150 packets per second, the delivery ratio for node N[18] starts dropping rapidly; at the same time, traffic fairness drops down to around 95%. This justifies the choice of this metric. The comparison of different algorithms, performed using network from Fig. 42a, using this metric, is shown in table 2.

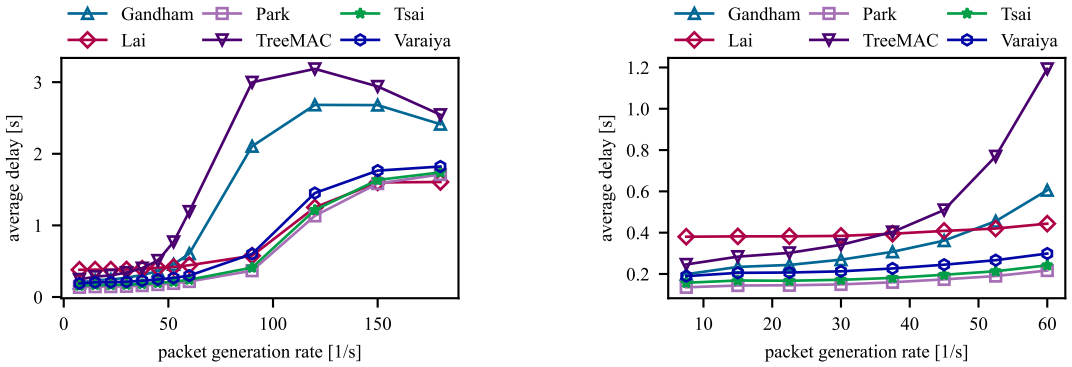
Table 2: Performance comparison

Algorithm	Cycle Length	Max Packet Generation Rate *[1/s]*	Max Throughput *[kb/s]*
TreeMAC	88	99.6	11.9
Gandham *et. all.*	76	114.6	13.7
Ergen and Varaiya	53	164.9	20.6
Lai	48	150.42	19.56
Park *et. all.*	49	179.4	21.1
Tsai and Chen	50	175.3	20.7

Besides the maximal packet generation rate, defined in the described way, the table lists two additional performance parameters: cycle length and maximal throughput. Maximal throughput is defined based on the maximal packet generation rate; it is defined as the throughput value at the maximal packet generation rate. As expected, TreeMAC produces somewhat longer schedules, and the maximum packet generation rate it can support is consequently lower. However, the reduced packet generation rate comes with the benefit of lowered implementation cost and shorted network construction time (as explained in Section 3.1.1. Lai's algorithm achieves the shortest schedule length. However, despite such a short schedule, the maximum packet generation ratio is lower than for all other distributed algorithms (Ergen and Varaiya, Park *et. all.*, Tsai and Chen). The reason for the worse performance of Lai's algorithm is the lack of traffic flow adaptation.

Another performance parameter where the algorithm by Lai performs worse is packet latency. This is especially pronounced at lower packet generation rate values. Fig. 44 shows the dependence of average packet delay on packet generation rate for all algorithms under consideration. The dependence of average packet delay versus packet generation rate is shown in Fig. 44a. At first glance, the algorithm by Lai does not perform worse than the other algorithms. However, examining the lower part of the range, plotted separately at Fig. 44b, reveals significantly higher latency. Larger delays when Lai's algorithm is used are an effect of a schedule not adapted to the traffic

flow. Namely, in such a schedule, a packet might traverse as little as one hop per cycle, resulting in a considerable delay.



(a) Average packet delay at whole packet generation range

(b) Average packet delay at low packet generation values

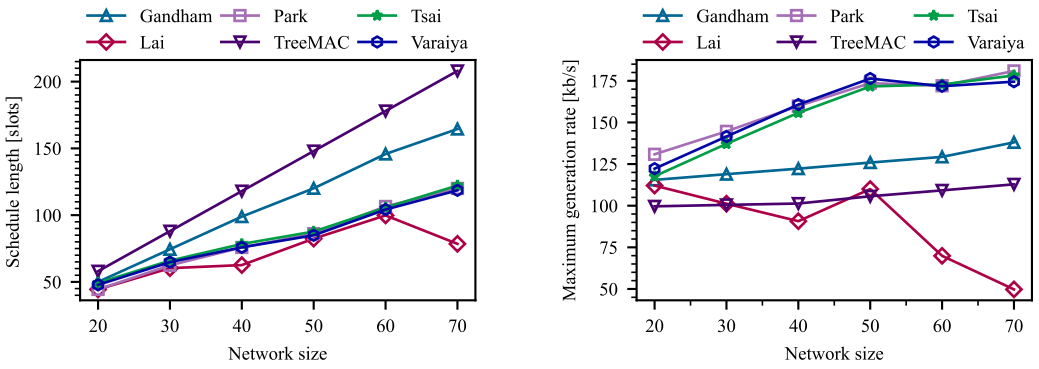
Figure 44: Dependence of average packet delay from packet generation rate.

The simulation results obtained using one network are not very relevant for a comparative evaluation since protocol performance depends on the network size. They are displayed here to illustrate differences between the algorithms used and explain the choice of metrics used for comparison. A relevant comparison is obtained based on the results of many simulations performed on all networks from the network set described in Section 3.2.5.

The first topology considered is the circular deployment area and the sink positioned in the middle. In the case of this topology, the networks from the set have sizes in the range of 20 to 60 nodes, with a step of 10 nodes. There are ten random networks with each size, meaning there are 50 networks in total. For each network, the packet generation rate is varied to obtain dependencies of fairness and throughput on it. Then based on these dependencies, the maximal packet generation rate is calculated.

For each network size, there are ten different networks. First, the maximal packet generation rate and the schedule length are calculated for each of these networks. Then, the values are averaged to obtain the average value for that size. The dependencies of these two parameters on the network size are obtained by repeating this for every network size. These dependencies are shown in Fig. 45.

According to the results, schedule calculation algorithms can be divided into two groups. The first group is the Trade-off group, comprised of protocols that achieve a good trade-off between complexity and network construction time on one side and throughput on the other. TreeMAC and algorithm by Gandham *et. all.* belong to



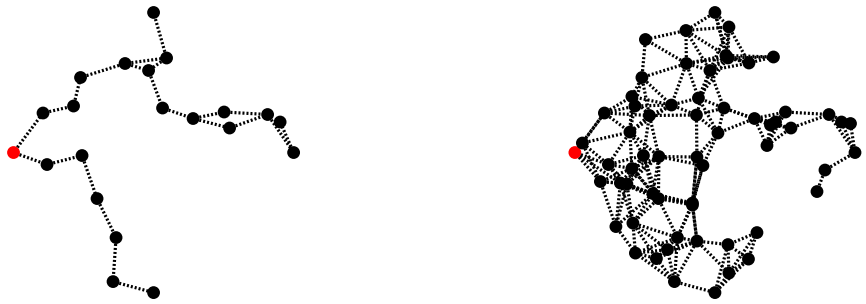
(a) Dependence of schedule length from number of nodes

(b) Dependence of maximal throughput from number of nodes

Figure 45: Performance of various algorithms vs different network sizes, in the case of circular deployment area with the sink in the middle.

this group. The second group is the high-performance group. Protocols in this group achieve outstanding performance but at the cost of increased complexity and network construction time. Algorithms that belong to this group are algorithms by Park, Tsai, and Ergen and Varaiya. Lai's does not belong to any group since it fails to achieve good performance, despite having high complexity. The reason for this is the lack of consideration of the traffic flow, which becomes evident when the network size increases.

The second network topology used for simulations is the circular network with the sink located at the network's edge. The network set contains networks with the same size ranges as in the case of a centrally positioned sink; this range is 20 to 60 nodes. Two networks from this set are displayed in Fig. 46; the first is comprised of 20 and the second of 60 nodes. These are just two out of 50 such networks in total.



(a) 20 nodes

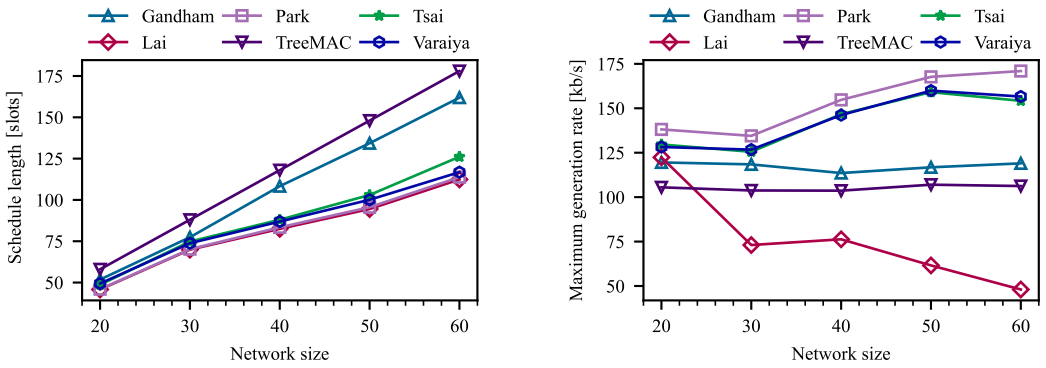
(b) 60 nodes

Figure 46: Two of 50 circular networks with sink at the edge used for simulations.

The dependencies of the maximal packet generation rate and schedule length on the

network size are obtained in the same way as in the case of a centrally located sink; they are shown in Fig. 47. In this case, there are two differences, revealing some important properties of the algorithms by Park *et. all.* and Gandham *et. all.*. The first one is that the algorithm by Park *et. all.* performs better than the other algorithms from the high-performance group. This makes this algorithm the best choice for a general network if a high packet generation rate is the priority.

The second difference is that algorithm by Gandham *et. all.* does not provide such a significant improvement compared to TreeMAC as in the case of the network with a centrally located sink node. The explanation is that, in this case, the sink has a lower number of neighbor nodes (due to its position). Consequently, there will be fewer top sub-trees, and they will be closer to each other. This drastically reduces the chance of two sub-trees having no interference edges between them. Therefore, the algorithm by Gandham *et. all.* will rarely be able to schedule two sub-trees in parallel, explaining the lower maximum packet generation rate values.



(a) Dependence of schedule length from number of nodes

(b) Dependence of maximal throughput from number of nodes

Figure 47: Performance of various algorithms vs different network sizes, in the case of circular deployment area with the sink located at the edge.

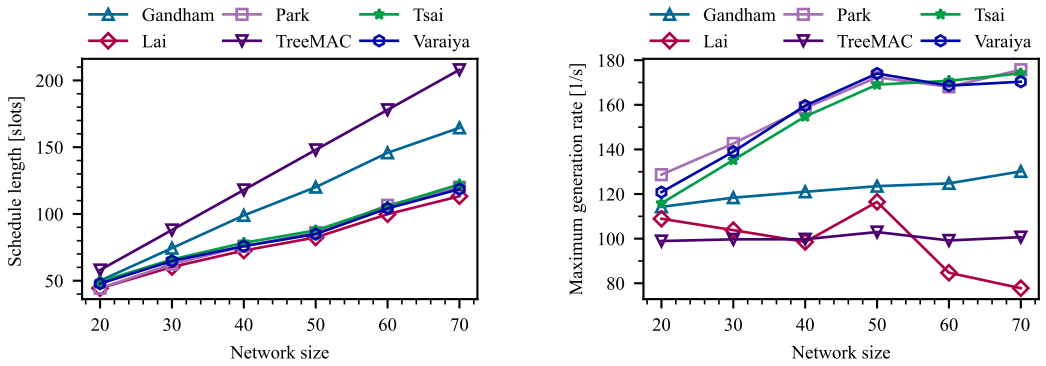
The third network type has a rectangular network deployment area; the width of this area is 250m and the length 500m. The sink node is positioned in the middle of the shorter rectangle edge. The set of networks used for evaluation contains networks with this topology with sizes ranging from 20 to 70 nodes. The step size is ten nodes, and there are ten networks of each size, resulting in 60 networks in total. Fig. 48 shows two networks with this deployment area; one with 20 and one with 70 nodes.

Same simulations as in the previous two cases are performed for these networks. The simulations are used to extract dependencies of the maximal packet generation rate and the schedule length on the network size. The results are shown in Fig. 49. The results

(a) 20 nodes (b) 60 nodes

Figure 48: Two wide rectangular networks.

for this case show higher dependence on the network size than in previous cases. For a low number of nodes, the centralized algorithms achieve almost no improvement (20 nodes) or not a considerable one. Therefore, the benefits of a more complex centralized protocol should be carefully weighed in these cases. On the other side, when the number of nodes is higher, complex protocols achieve even larger improvement than in the case when circular networks were considered.



(a) Dependence of schedule length from number of nodes (b) Dependence of maximal throughput from number of nodes

Figure 49: Performance of various algorithms vs different network sizes, in the case of wide rectangular deployment area.

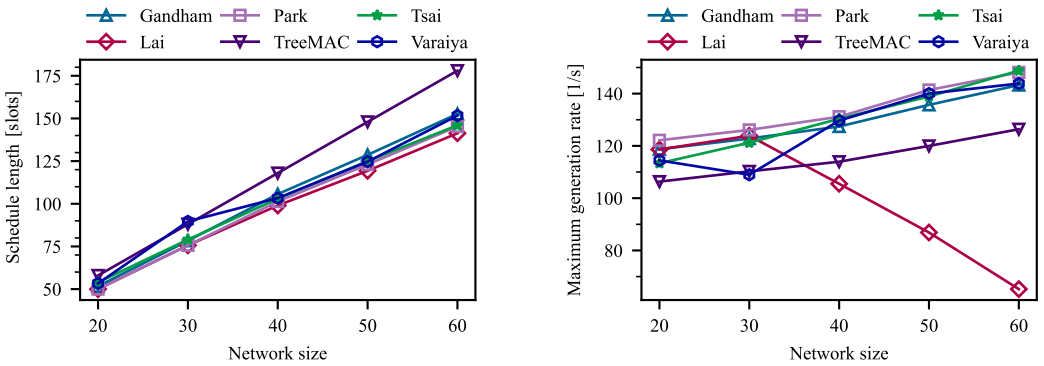
Finally, the evaluation is performed on a group of narrow rectangular networks; the width of the deployment area for the networks in this group is only 100 meters, which is equal to the radio range. In this case, the number of nodes is varied from 20 to 60 nodes with a step of 10 nodes, resulting in 50 networks (there are ten networks for each size). Fig. 50 shows two networks from this group, with sizes 10 and 60. This case is substantially different from the previous three cases because these networks are linear, and all sub-trees are very close. This means that it is impossible to schedule the whole sub-trees simultaneously, but only their different parts. Furthermore, parallel scheduling in total is more challenging in such a network, and the number of parallel

transmissions that can be achieved is lower.

(a) 20 nodes (b) 60 nodes

Figure 50: Two narrow rectangular networks.

The simulations are performed in the same way as in the previous three cases. Furthermore, the same dependencies are used to compare different algorithms; these are dependencies of the cycle length and the maximal packet generation ratio on the network size. These results are shown in Fig. 51. As expected, parallel scheduling of different sub-trees does not result in a significant benefit in this case. As a result, complex centralized algorithms provide little improvement over the algorithm by Gandham *et. all.*. This makes high-performance algorithms unsuitable for such applications, as their drawbacks outweigh the minor packet generation improvement they offer in this case.



(a) Dependence of schedule length from number of nodes (b) Dependence of maximal throughput from number of nodes

Figure 51: Performance of various algorithms vs different network sizes, in the case of narrow rectangular deployment area.

On the other hand, TreeMAC performs very well in this case. Even though the algorithm by Gandham *et. all.* improves over TreeMAC, the improvement still might not outweigh the benefits of TreeMAC's simplicity and very fast network construction. Therefore, the choice between these two protocols must be made for each specific application by carefully examining the requirements and performance parameters both protocols can achieve. It is also worth noting that the algorithm by Gandham *et. all.* does not improve over TreeMAC through parallel scheduling of sub-trees in this case. Instead, the improvement comes from the advanced linear network scheduling algorithm

alone.

3.4 Conclusion

Multiple TDMA protocols for convergecast networks are proposed up to date; these protocols use various approaches to design scheduling algorithms to create the shortest possible schedule that considers traffic flow. They can be divided into two groups: the high-performance group and the trade-off group. Protocols from the high-performance group achieve similar performance in theory; therefore, it is not clear which one of them will perform the best in which case. This chapter provides an in-detail comparative analysis of these protocols and their evaluation using simulations and an extensive set of random networks. The most appropriate protocol can be selected for each application based on these results. Furthermore, expected protocol parameters, such as throughput, fairness, and latency, can be estimated using the results provided in this chapter.

4 Optimizing Scheduling Algorithms to Consider Multiple Packet Transmissions in a Time Slot

Time schedule calculation algorithms assign one packet to each node and then calculate the schedule so that all those packets reach the network sink during one cycle. It is assumed that a node may transmit only one data packet during each time slot. Theoretically, such slot length would result in the best performance. However, it is rarely possible to adjust the time slot duration to be exactly as long as a packet transmission duration. In these cases, the TDMA performance can be increased by considering these multiple transmissions when the schedule is calculated.

Adjusting the time slot duration to equal the packet transmission time may not be possible for various reasons. In most cases, the reason is traffic comprising different packet sizes. However, even when all packets have equal size, equal transmission and slot time may be impossible due to other reasons. For example, one may be MAC protocol limitations, which define minimal slot duration time. That time is often longer than the packet transmission time. Another common reason is long synchronization delays. In that case, it is possible to equalize transmission and slot time, but it would result in poor performance. The reason is that the synchronization delay might be longer than the slot duration. For example, ISA.100.11A defines a default synchronization delay of around 2ms. In comparison, packet transmissions times lasting a few hundred microseconds are not uncommon in wireless sensor networks.

The example in Fig. 52 illustrates how ignoring multiple packet transmission has a negative effect on performance. In this example, it is possible to transmit three packets in one time slot. This was not considered when the seven slots long schedule displayed in Fig. 52 was calculated.

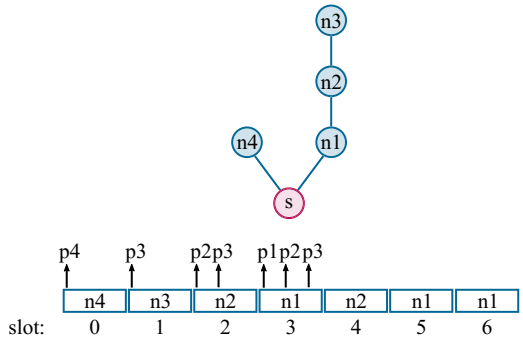


Figure 52: Example of unused time slots due to transmission of multiple packets in one time slot.

In the example (Fig. 52), three time slots are allocated to node $n1$. One to transmit

its own data and two to forward data packets received from nodes $n2$ and $n3$. However, the second two time slots, intended for forwarding data, are not used; that can be seen in Fig. 52. In the figure, transmissions are represented by arrows. In the fourth time slot, node $n1$ transmits three packets to the sink: its own packet $p1$, and packets $p2$ and $p3$, from nodes $n2$ and $n3$. All three packets are transmitted during the same time slot, intended for transmitting only packet $p1$ by the algorithm. Therefore, slots five and six, intended for transmitting packets $p2$ and $p3$, remain unused.

Since equalizing time slot duration with packet transmission time is not feasible, optimizing the time schedule to account for multiple packet transmissions is an approach that can provide benefits in many cases. In this thesis, a TDMA MAC protocol, which considers this, is proposed. Because it is inspired by TreeMAC and takes multiple transmissions into account, it is called M-TreeMAC.

4.1 M-TreeMAC

M-TreeMAC is based on TreeMAC, but the number of frames required by each node is calculated to take multiple packet transmissions during a single time slot into consideration. The number of frames each node gets is called active frames count. It is dependent on the maximal packet count, which is equal to the number of packets that can be transmitted during a time slot. Maximal packet count is a protocol parameter, and it should be set by the user. The value of this parameter should be chosen considering time slot duration and expected traffic in the network. It must be selected by the user for each particular application.

An M-TreeMAC schedule is divided into frames, in the same way as in the case of TreeMAC. Each frame consists of three slots. To each node, an appropriate number of frames is assigned (not time slots like in many TDMA protocols). Based on its level in the network, a node in which time slot of the assigned frame it may transmit. Compared to TreeMAC, slot order within a frame is reversed in M-TreeMAC. This means that children are scheduled first, and then their parents, opposite from TreeMAC. Such scheduling order increases the chance that a node will have more than one packet to transmit.

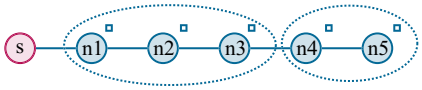
The implementation of M-TreeMAC remains distributed and low-cost while significantly reducing packet latency. The calculation of active frames count is done during the network construction phase, using a modified network discovery algorithm. The modification does not introduce any new packet types. Instead, it adds one additional parameter to the existing packet type, *dscResp* packet. As a result, the complexity and execution time remains almost the same. The only difference is a slight increase in the size of *dscResp* packets. However, the effect of the increased size of this packet is

insignificant and has a negligible impact on protocol performance.

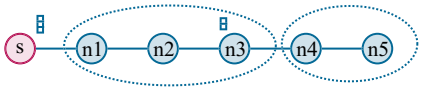
4.1.1 Active Frames Count Calculation

In TreeMAC, the value of a node's active frames count parameter is equal to the size of the sub-tree rooted at that node. In the case of M-TreeMAC, the active frames count of each node depends on the whole topology of the sub-tree rooted at that node; therefore, it can not be calculated based on the sub-tree size only. This dependence exists because of the way M-TreeMAC calculates the schedule. When M-TreeMAC schedule is calculated, one packet is assigned to each node. Then, nodes are scheduled, one linear sub-network at a time. Packets are followed, and whenever one part of the linear sub-network has transmitted all of its packets, another linear sub-network is scheduled. Therefore, the number of frames one sub-tree requires to transmit all of its packets depends on how well these packets can be grouped and transmitted together in the same time slot.

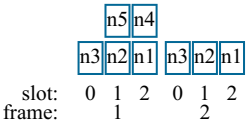
The order of time slots during a frame in M-TreeMAC is reversed compared to TreeMAC. This means that, for example, the sink's neighbor transmits after its child node has forwarded the packet to it and not before (like it would in TreeMAC). Additionally, that child node transmits after receiving a packet from its child as well. In this way, three packets are grouped and transmitted to the sink in the same time slot. This is illustrated using an example, shown in Fig. 53. The example shows a liner network and M-TreeMAC schedule calculated for it. In this case, the maximal packet count, denoted by k, is three. Each node calculates its slot within a frame using equation $slot = 2 - level \mod 3$.



(a) Network and packet distribution before the first frame.



(b) Network and packet distribution after the first frame.



(c) Time schedule.

Figure 53: M-TreeMAC schedule for a linear network, and packet grouping.

As the example shows (Fig. 53), packets originating from certain nodes will be

grouped on their way to the sink. Nodes whose packets will be grouped belong to the same group. Groups are marked with dashed lines on the figure. Based on this example, it can be concluded that the number of frames required to schedule a linear network is equal to the number of groups in the network. In this case, two frames are needed. Additional frames would have been required if the maximal packet count were less than three.

The dependence of the active frames count of a node on the topology of the sub-tree rooted at that node is illustrated in Fig. 54. The figure shows two different sub-trees of the same size and different topologies. The root of these sub-trees is node nr. The first sub-tree shown in Fig. 54a needs two frames to transmit all of its packets to the parent node. The schedule of this sub-tree is shown in the same figure. The second sub-tree, showed at Fig. 54b, needs one frame more, because of the additional branching at node $n2$.

(a) Network

(b) TreeMAC schedule

Figure 54: Dependence of active frames count on sub-tree topology.

4.1.2 Protocol Definition

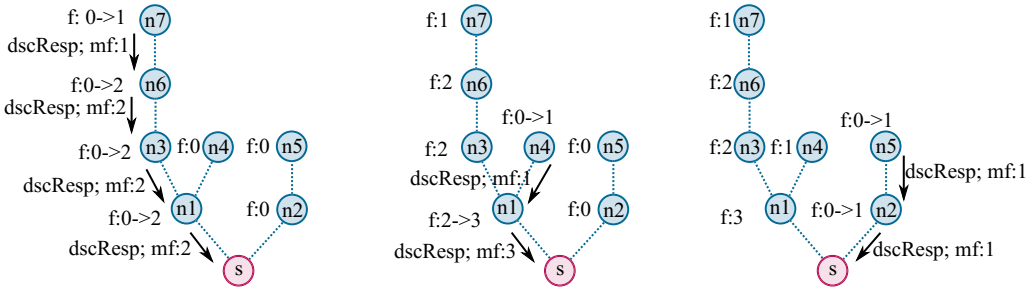
The schedule calculation of M-TreeMAC is performed in a similar way as in the case of TreeMAC. Each node calculates its own schedule based on the values of four parameters. These are active frames count, level in the tree, starting frame, and schedule length. Each node determines active frames count and level values during the network construction phase; this is achieved using modified *dscResp* packet. After the network construction, two parameters remain undetermined, schedule length and starting frame. Each node gets values of these two parameters from its parent in a protocol-specific packet called *Sch*. After the tree construction is finished, the sink calculates these two parameters based on the information collected. Then, it starts the dissipation process by transmitting a *Sch* packet to its children nodes.

Network discovery is performed in the same way as for TreeMAC. It was previously

described in detail Section 3.2.2. The packet type of importance is *dscResp* packet. Each node sends this packet to the sink after discovering its neighbors. This packet contains the list of the sender's neighbors located at the next level. M-TreeMAC uses this packet to calculate the active frame count for each node in the network. To perform the calculation, one additional parameter is added to this packet; this is my frames count, denoted by mf.

The active frame count value is determined during the network construction phase for each node in the network; this value is denoted by f, symbolizing frames. In addition to its value of f, each node learns this value for its children. Besides network topology, the active frame count depends on maximal packet count, k (maximal number of transmissions in a time slot). Each time a node receives a *dscResp* packet originating from a leaf node, it updates the value of f. Whether a *dscResp* is originating from a leaf node or not is determined by examining the list of neighbors in this packet; leaf nodes have no neighbors at the next level.

Each time a node receives a leaf-*dscResp* packet, it reads the value of mf. That value corresponds to the current active frame count of the child node sending the packet. Then it adds the value of mf to f, because it has to be active when its child is active. If the node is the last in its group, it also increases f for one. This is because the last node in a group can not merge its packet with the packet from its child. The reason is that it receives that packet after transmitting its own. After these actions are performed, the node checks if it has received the *dscResp* packet from all children. If that is true, the node checks if it can transmit its packets during the number of frames allocated; the number of packets it can transmit is calculated as $f \cdot k$. If necessary, it increases the value of f to a number that can accommodate all its packets. Finally, the node can forward the *dscResp* packet to its parent. Before this is done, the node sets the mf value in the *dscResp* packet to the value of its newly calculated f.



(a) *DscResp* from $n7$ (b) *DscResp* from $n4$ (c) *DscResp* from $n5$

Figure 55: Active frames count calculation using *dscResp* packets originating from leaf nodes.

Calculation of active frames cont and propagation of *dscResp* packet is illustrated using an example network, as displayed in Fig. 55. The path of the *dscResp* packet originating from node $n7$ is illustrated on Fig. 55a. Before node $n6$ receives this packet, its value of f is zero. Then it is increased two times, once for the value of mf in the packet and once because node $n6$ is the last in its group. As a result, the value of f at node $n6$ is set to two. Subsequently, the value of mf in the *dscResp* packet is set to the same value. On its way to the sink, this packet sets f to two in all further nodes on its way ($n3$ and $n1$).

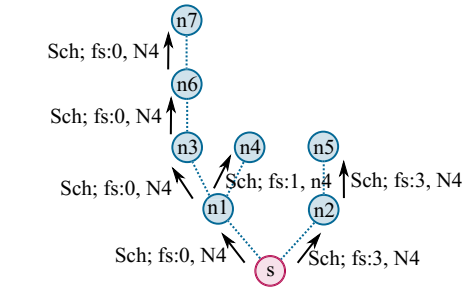
The path of *dscResp* packet sent by node $n4$ is illustrated in Fig. 55b. Node $n1$ receives this packet after previously receving *dscResp* from node $n7$. Therefore, when node $n1$ receives *dscResp* from node $n4$, it increases f from two to three. Afhtherwards, it forwards the packet to the sink, setting mf to three previously.

Once the network formation phase is completed, all nodes have the value of active frames count parameter set and they have obtained and saved values of active frames count parameter of their children nodes. The only remaining values needed by every node to calculate its own time schedule are starting frame and schedule length. The value of theses parameters will be obtained from *Sch* packet, which each node receives from its parent. The sink node initializes the schedule calculation by calculating total schedule length and starting frames of its children nodes, and sending them a *Sch* packet, which contains these values. When a node receives this packet and obtains its starting frame, it can calculate starting frames for its own children and send them a *Sch* packet. The usage of *Sch* packets to calculate and propagate information about staring frames is illustrated using an example on Fig. 56.

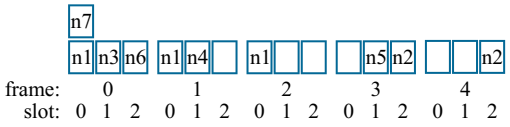
The example (Fig. 56) uses the same network used to illustrate the propagation of *Sch* packets. The sink of this network has two children, nodes $n1$ and $n2$. During the tree construction phase, the sink received information that node $n1$ has an active frame count of three and node $n2$ of one. Therefore it first calculates the cycle length to be four frames. Then, it assigns the first frame to the starting frame of the branch rooted at node $n1$. Since that branch needs three frames, the next free frame, frame four, is assigned to the branch at node $n2$. Finally, it sends a *Sch* to its children. This is repeated at each node until all nodes learn their starting frames and the cycle length. Complete schedule is shown in Fig 56b. To calculate their slot within an active frame, nodes use the equation $slotNo = level \mod 3$.

4.1.3 M-TreeMAC Performance

M-TreeMAC reduces time schedule length significantly, reducing packet delivery latency. However, the throughput and maximal generation rate remain the same in a



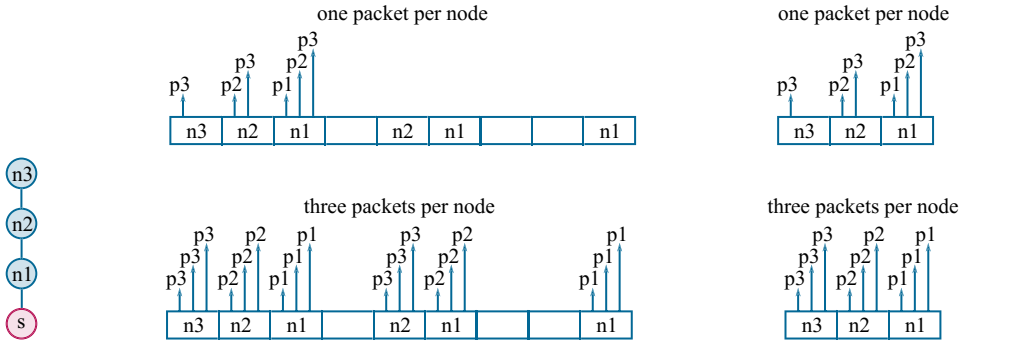
(a) Example network and *Sch* packets used for schedule calculation



(b) M-TreeMAC time schedule

Figure 56: Calculation of stating frames and total schedule length dissemination using *Sch* packets.

general case. The lack of improvement in terms of throughput is explained using an example shown in Fig.57. In the example, the schedule created using M-TreeMAC is compared with a modified TreeMAC schedule. The modified TreeMAC schedule uses reverse slot assignment within a time frame, as M-TreeMAC does. This means that nodes at higher levels are scheduled before nodes at lower levels. Nonetheless, the results and conclusions remain even without this modification because the pipe-lining effect would result in packet grouping and multiple transmissions in the case of the unmodified TreeMAC schedule.



(a) Network (b) TreeMAC schedule (c) M-TreeMAC schedule

Figure 57: Interference graph creation and color assignment.

As it can be seen in Fig. 57, M-TreeMAC produces significantly shorter schedules. This is because additional time slots assigned to nodes by TreeMAC for forwarding

packets from upper layers (forwarding slots) are now no longer needed. Instead, nodes can forward the upper layer packets with their own packet in the same time slot. As seen in Fig.57c, under low-traffic conditions, these forwarding slots are not used. However, when the traffic is sufficiently high, all nodes will have a few packets in their buffer, and forwarding time slots will be used to transmit these packets. That is why the maximal packet generation rate remains the same even though the schedule length is significantly reduced. The benefit of M-TreeMAC compared to TreeMAC is, nonetheless, significant, as packet latency will be substantially reduced.

4.2 M-TreeMAC Evaluation

Since M-TreeMAC is an improvement of TreeMAC, to evaluate its performance, it is compared with TreeMAC. The comparison is made with the same set of networks previously used to compare schedule calculation algorithms. The set contains networks with four different deployment area shapes. For each shape, the set includes networks with a different number of nodes. Furthermore, for each shape-size pair, there are ten different networks. Additionally, several randomly generated linear networks are simulated for this evaluation. Linear networks are introduced because M-TreeMAC is especially effective for them.

As explained before, M-TreeMAC reduces schedule length significantly, but maximal throughput remains the same. What changes is how packet delivery fairness varies with increasing traffic in the network and packet latency. In theory, fairness should be higher for the case of M-TreeMAC because the number of time slots assigned to each node considers multiple transmissions in the same time slots. However, in reality, the traffic is random. This means that the number of packets that can be grouped is not predictable and that the actual traffic flow is different than the one M-TreeMAC optimized the schedule for.

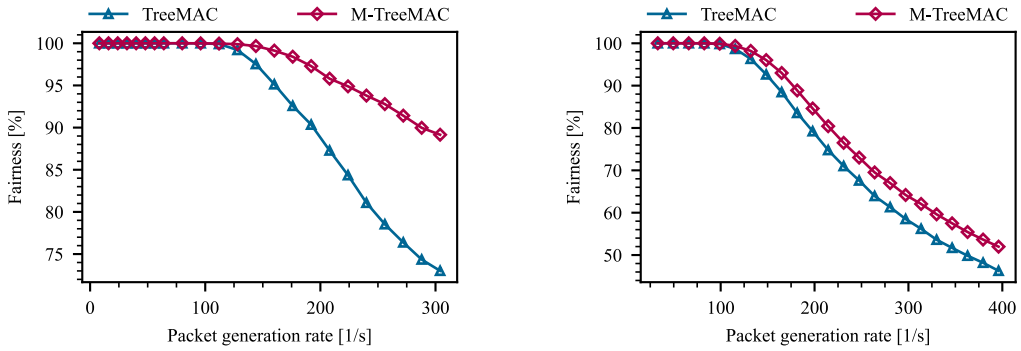
In the case of linear networks, the benefits of M-TreeMAC are more significant. Apart from reduced delay, fairness is increased. This means that maximal throughput, defined in relation to network fairness, also increases. In the cases of more complex networks with circular or rectangular deployments areas, the fairness achieved is similar to TreeMAC. The packet latency is, however, significantly lower. For these reasons, the two cases are analyzed separately.

4.2.1 Linear Network

To evaluate the performance of M-TreeMAC, six linear networks containing different numbers of nodes are used. The number of nodes is varied in the range from 8 to 33.

The radio range is 100 meters, while the spacing between nodes is 40m. This means that the sink node will have two child nodes; therefore, there will be two top-subtrees in each network. For each network, the packet generation rate is varied. As the packet generation rate increases, values of throughput, fairness, and average packet latency are calculated and saved. This is repeated for both TreeMAC and M-TreeMAC.

Two separate cases are analyzed before analyzing the dependence of network performance on the number of nodes in the network. These are the smallest network from the set, containing eight nodes, and the largest, with 33 nodes. The dependence of fairness on packet generation rate for these two networks is plotted in Fig. 58. In both of these networks, M-TreeMAC achieves a significant improvement compared to TreeMAC. In the first case, the improvement is especially significant for the network with eight sensor nodes. The maximum packet generation rate, calculated at a fairness value of 95%, is 250 packets per second when M-TreeMAC is used, compared to 180 when TreeMAC is used. This significant improvement is achieved without increasing the implementation cost, complexity, or network construction time.



(a) Network with 8 nodes

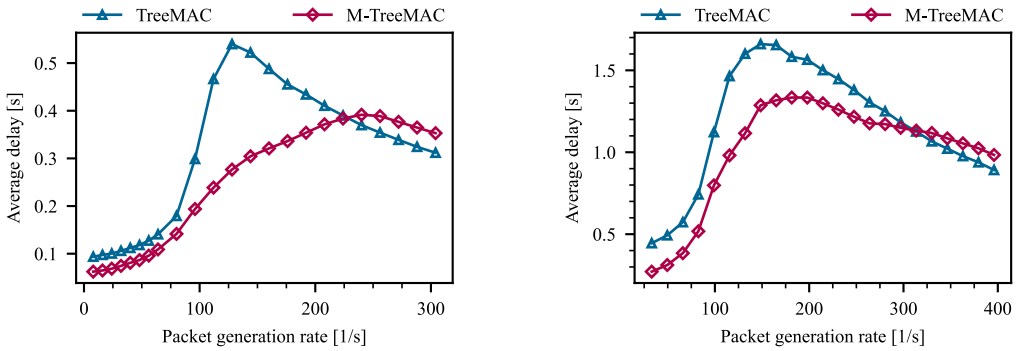
(b) Network with 33 nodes

Figure 58: Dependence of traffic fairness on packet generation rate.

The improvement is considerably smaller for the larger network, with 33 sensor nodes. This is because M-TreeMAC calculates the active frame count by assigning one packet to each node and following it. However, in real-world applications, the data traffic is irregular; due to this, not all nodes will have a packet simultaneously. As the network size increases, the effect of irregular traffic becomes more pronounced. Due to this, the M-TreeMAC schedule becomes less optimal, resulting in a less pronounced improvement.

Fig. 59 shows the dependencies of average delay on packet generation rate for the same two networks. Unlike maximal throughput increase, which becomes less pronounced, packet delivery latency improvement remains consistent over different net-

work sizes. The reason is that the delay is less affected by the traffic flow mismatch. The figure shows a reduced average delay for all usable values of packet generation rates. The improvement is especially significant at lower generation rates, where the delay is reduced to around 50%. This makes M-TreeMAC especially suitable for usage in networks with low traffic and requirement for low packet delivery latency.



(a) Network with 8 nodes

(b) Network with 33 nodes

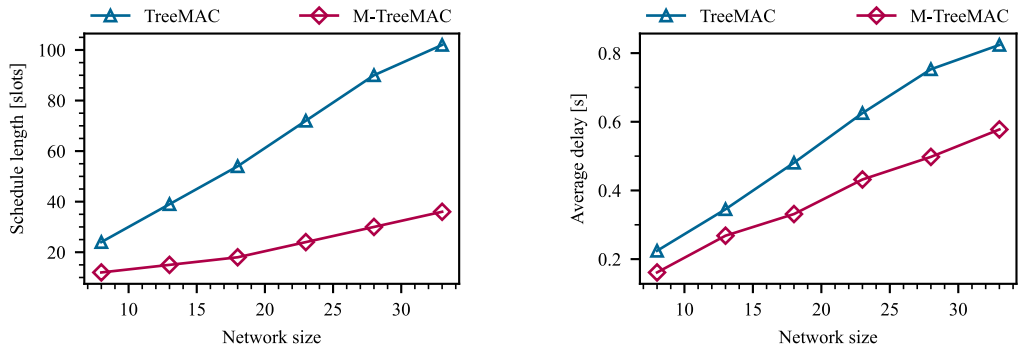
Figure 59: Dependence of average packet delay on packet generation rate.

The dependence of average delay on network size is shown in Fig. 60. Average delay at 80% of the maximal packet generation rate was used as a reference average delay value of each network. The maximal packet generation rate was calculated as the theoretically maximal packet generation rate for TreeMAC. This value is always within the region with reasonably high fairness and, at the same time, with a considerable amount of data traffic. Therefore, it provides a relevant average delay value for evaluating the improvement archived by M-TreeMAC. The results (Fig. 60) show significant improvement for all network sizes.

4.2.2 Randomly Generated Networks

Multiple randomly generated networks with two different deployment areas are used to evaluate M-TreeMAC performance in the case of more complex networks. The first deployment area is circular, with the sink node positioned in its center. The radio range is 100m, and the radius of the deployment area is 280m. The number of nodes is varied from 20 to 60, and ten different networks are generated for each number of nodes. The second deployment area used is rectangular, with a length of 500m and a width of 250mm. Again, the number of nodes is varied in the same range, and ten different networks of each size are generated in this case as well.

First, simulation results for a circular network containing 20 nodes are analyzed.

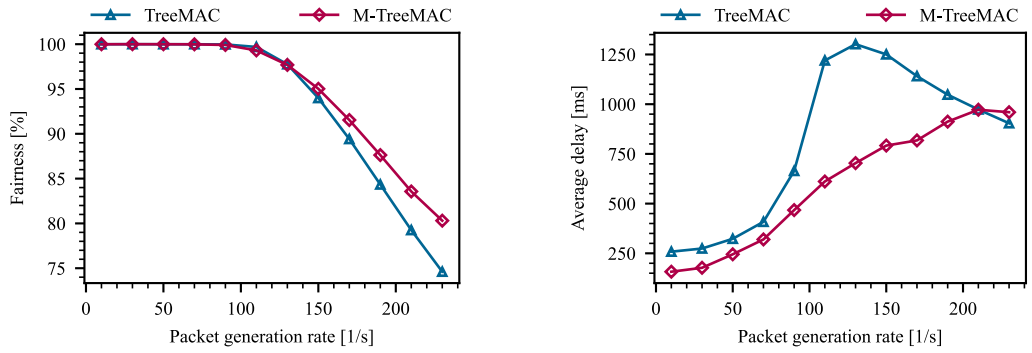


(a) Schedule length

(b) Average delay

Figure 60: Dependence of schedule length and average delay on network size.

Dependencies of fairness and average delay for the case of this network are shown in Fig. 61. The improvement in terms of fairness is minimal in this case. The fairness is almost identical for both TreeMAC and M-TreeMAC under low packet generation rate conditions, Fig. 63a. When the packet generation rate is increased, M-TreeMAC achieves slightly better fairness. However, this only occurs when fairness drops below 90%. Since sensor networks are usually not operating in this region, this can not be considered an improvement. However, the improvement in terms of average delay remains significant for all packet generation rate values. The improvement is especially substantial at lower traffic generation rates, more than 100% in some cases.



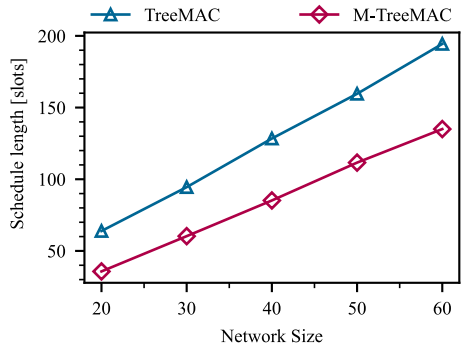
(a) Fairness

(b) Average delay

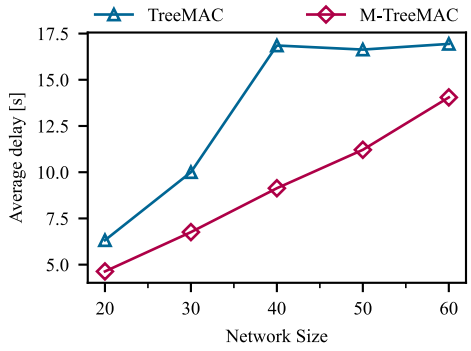
Figure 61: Dependence of fairness and average delay on packet generation rate.

The dependence of schedule length and average delay on network size is shown in Fig. 62. Average delay at 80% of maximal packet generation rate is used to describe the average delay of each network. The theoretical maximal generation rate of TreeMAC is

used to calculate this value. The results show significant delay reduction for all network sizes.



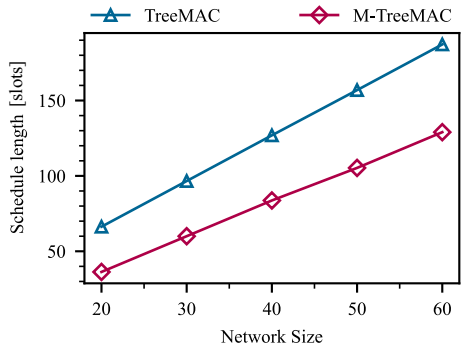
(a) Schedule length



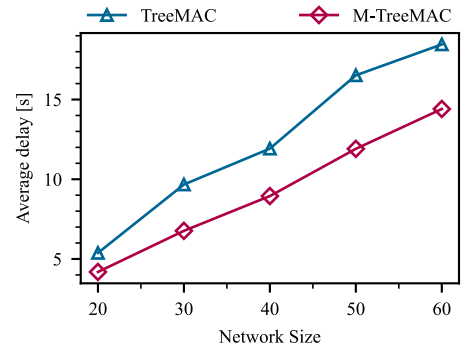
(b) Average delay

Figure 62: Dependence of schedule length and average delay on network size for circular deployment area.

The same procedure was repeated for the rectangular network deployment area. The number of nodes was varied in the same range, and both protocols performance was simulated to obtain a comparison. The results are plotted on Fig. 63. The improvement achieved by M-TreeMAC is significant in this case across all network sizes.



(a) Schedule length



(b) Average delay

Figure 63: Dependence of schedule length and average delay on network size for rectangular deployment area.

4.3 Routing Topology Optimization

Unlike a time schedule calculated using TreeMAC, whose length depends on the network size only, the length of a schedule calculated using M-TreeMAC is dependent on the

topology of the routing network. The dependence on topology exists because the active frames count of a node depends on the topology of the subtree rooted at that node, as explained in the previous section. Therefore, the performance of M-Tree MAC can be additionally increased by optimizing the network topology. This chapter of the thesis proposes an algorithm that finds the optimal topology. It is based on the idea previously published by the author of this thesis in [14]. The proposed algorithm is evaluated in a simulator, using many randomly generated sensor networks with different sizes and deployment area shapes.

4.3.1 Optimization Criterion

TreeMAC uses breadth-first search (BFS) algorithm for creating the shortest-path spanning tree. This algorithm goes through the network level by level. The algorithm goes through nodes of one level in the order of their discovery. When the algorithm examines a node, it assigns all available eligible nodes to be children of that node. A node is eligible if it is located one level above and does not yet have a parent. Due to the greedy nature of the algorithm, some nodes will have a large number of children and some none. Thus, the resulting routing tree will be unbalanced, which means that some top-sub trees can be significantly larger than the others.

Fig. 64a shows a shortest-path tree constructed using BFS algorithm. The constructed routing tree has two top sub-trees, one of size five and one of size one. A top-load balanced routing tree, created for the same sensor network, is shown in Fig. 64b. Here, the size of both sub-trees is equal.

(a) BFS shortest path tree

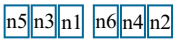
(b) Balanced shortest path tree

Figure 64: An example of two different routing trees constructed for the same network.

Fig. 65 shows schedules created using M-TreeMAC for two networks form Fig. 64. In the case of the routing tree created using BFS algorithm, the schedule is seven frames long. When a top-load balanced tree is used for routing packets in the same network, the schedule length produced by M-TreeMAC is reduced to three frames. This example illustrates how can M-TreeMAC schedule length be reduced by optimizing the routing

tree. A top-load balanced tree was used to illustrate that. However, a top-load balanced tree is not optimal in the general case.

(a) Schedule in the case of BSF Tree



(b) Schedule in the case of top-load balanced tree

Figure 65: M-TreeMAC schedules for the two different routing trees.

To define the criteria for the optimal routing tree for M-TreeMAC, the *DscResp* packets originating from leaf nodes are observed. The number of frames required by each node is calculated based on these packets. As a *DscResp* packet is being forwarded towards the sink, the value of parameter f, representing the number of frames required, is increased by the value of parameter mf in the packet. Therefore, to construct the optimal tree, *DscResp* packets should be routed in the direction that results in a minimum increment of active frames count.

The proposed criteria for optimizing routing topology is to minimize the number of leaf nodes in the network. If the number of leaf nodes is minimal, the length of time schedule produced by M-TreeMAC will be minimal. This is because each leaf node results in an additional *DscResp* packet. The additional packet, in turn, increases the values of active frame count parameters in all nodes on its way to the sink.

To justify the proposed criterion, network section from Fig. 66 is considered. Assignment of children to nodes $n1$ and $n2$ is considered. It is assumed that node $n3$ is assigned to node $n1$ and effects of assigning node $n4$ to node $n1$ or $n2$ are analyzed. If node $n1$ is chosen as the parent of node $n4$, node $n2$ is left without possibility to be assigned a child node. Therefore, this choice introduces an additional leaf node to the network, node $n2$. The effect of both of these assignments on schedule length are analyzed. The flowing notation is used for the analysis. Any *dscResp* packet received by node $n4$ from its parent is denoted by $DscResp_4$. Let the value of parameter mf in any *dscResp* packet sent by node $n4$ to its parent be mf_4. Values of parameter f at nodes $n1$ and n_2 are denoted as f_{n_1} and f_{n_2} respectively. Let their values before receiving $DscResp_4$ packet be f_1 and f_2. The two cases are analyzed under the assumption that there is only one $DscResp_4$ packet. Additionally, it is assumed that nodes $n1$ and $n2$ are not the last nodes in the group. Nevertheless, the conclusions remain unchanged in the general case.

In the first case, node $n1$ is chosen as a parent of node $n4$. In this case, node $n1$ receives $DscResp_4$ packet. After receiving this packet, node $n4$ increases value of its active frame count parameter to $f_{n_1} = f_1 + mf_4$. In this case, node $n2$ is a leaf node.

(a) Network section (b) First case (c) Second case

Figure 66: Dependence of number of required frames on parent children assignment.

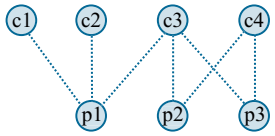
Therefore, it transmits an additional $DscResp$ packet. Due to this, the active frames count value at node $n2$ is increased to $f_{n_2} = f_2+1$. The total number of frames required by nodes $n1$ and $n2$ is $f1 + mf_4 + 1$.

In the second case, node $n2$ is chosen as a parent of node $n4$. In this case, $DscResp_4$ packet is received by node $n2$. When node $n2$ receives this packet, it increases its value active frames count to $f_{n_2} = f_2 + mf_4$. The value of f_{n_1} remains unchanged. The total number of frames required by nodes $n1$ and $n2$ in this case is $f_1 + mf_4$. This is one frame less than in the previous case. Thus, it can be concluded that an additional leaf node results in an increased length of time schedule for at least one frame.

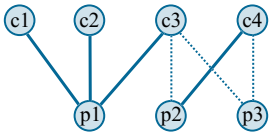
4.3.2 Maximum Matching Optimization Algorithm

The number of leaf nodes should be minimized to find the optimal shortest-path spanning tree. Since all nodes at the last level must be leaf nodes, the number of leaf nodes at all other levels (low-level leaf nodes) should be minimized. A low-level leaf node is created whenever one of the nodes on the current level does not get any children nodes assigned to it. The number of low-level leaf nodes at one level is independent of their number on other levels. Therefore, children assignment can be optimized separately for each level. In this way, the topology optimization problem is divided into multiple sub-problems, minimizing low-level leaf nodes at every level in the network. Finding a child-parent assignment that results in a minimal number of low-level leaf nodes is called the children assignment problem.

The children assignment problem for one level of a simple example network is illustrated in Fig. 67. The considered level contains three parent nodes; this level is called parent level. Nodes from one level above the parent level, called children level, can be assigned to these three nodes. In this case, there are four nodes on the children level. An optimal assignment leaves the lowest possible number of parent nodes without children. In this case, any assignment that assigns at least one node to each parent is optimal. For example, assigning $c1$ to $p1$, $c3$ to $p2$ and $c4$ to $p3$ will result in no low-level leaf nodes.



(a) Children assignment problem



(b) One non optimal children assignment problem solution

Figure 67: An example illustrating children assignment problem.

However, even in this example (Fig. 67), children assignment problem is not trivial. One non-optimal solution is shown in Fig. 67b. This sloution creates one low-level leaf node. Here, node $c3$ was assigned to node $p1$, leaving nodes $p2$ and $p3$ to compete for one child node, node $c4$. As the result, node $p3$ becomes a low-level leaf node.

Topology optimization problem was divided into multiple children assignment problems, one for each level of the network. Then, the same strategy is applied to children assignment problem. It is divided into two sub-problems; assignment of exactly one child to as many parent nodes as possible and assigning parents to the remaining nodes on the children level. The second sub-problem does not affect the number of low-level leaf nodes; therefore, it can be solved to satisfy some other requirement. However, the first sub-problem is critical for finding the optimal topology. It is called parent-children matching problem.

Parent-children matching problem is equivalent to bipartite graph maximum matching problem. The nodes at two levels correspond to two sets of nodes in the bipartite graph, while possible connections between them represent the edges of this graph. A matching of such a graph is comprised of node pairs, where each node in a pair is from a different set. A matching is equivalent to a child-parent assignment. A maximum matching is a matching containing the largest number of pairs. Finding a bipartite graph maximum matching is a well-known and studied problem. Therefore, it can be solved using one of the state-of-the-art algorithms.

The proposed algorithm for solving children assignment problem is shown on Algorithm 3. It is called maximum matching children assignment algorithm, and a routing tree created using this algorithm is called maximum matching tree. The input of the algorithm is a bipartite graph G. A bipartite graph is comprised of two sets of nodes: a set of all nodes at the parent level, U, and a set of all nodes at the children level, V. The set of all edges between nodes at the two levels, excluding edges between the nodes located at the same level, is E. A bipartite graph is usually denoted by $G = (U, V, E)$.

The first part of the problem is finding a maximum matching of the bipartite graph provided as the algorithm's input. Ford Fulkerson's algorithm is used to find this

matching, line 2. In the algorithm, an inputted bipartite graph is G; based on it, the algorithm computes a maximum matching, M. This matching is then used to perform children assignment, in line 3. As a result, one children node is assigned to some or all nodes at the parent level. After this step, some of the nodes from the children level may not have a parent node.

The second part of the problem is the assignment of parents to nodes from the children level that did not match with any parent node. Even though this assignment does not affect the M-TreeMAC schedule length, a more balanced number of children among parent nodes is favorable. Therefore, the proposed algorithm balances each parent's number of children nodes. It goes through nodes at the parent level and examines their neighbors located one level above, line 6. Here, a parent node is labeled n_p, and the set of its neighbors at one level above, $Neigh(n_p)$. The first such neighbor without a parent is assigned to node n_p. Then, the algorithm moves over to the next node from the parent level. This is repeated until all nodes on the children level get a parent, line 5.

Algorithm 3 Maximum matching children assignment

```
G = (U,V,E) - bipartite graph
M = FordFulkerson(G)
for all (n_p, n_c) in M do
    assign n_c as child of n_p
while there is a child node without a parent do
    for all n_p in U do
        for all n_c in Neigh(n_p) do
            if not hasParent(n_c)) then
                assign n_c as child of n_p
                break
```

4.3.3 Evaluation

Maximum matching children assignment algorithm is implemented in the OMNeT++ simulator to evaluate its efficiency. The same set of networks used to compare M-TreeMAC and TreeMAC was used to evaluate this algorithm. This set comprises networks with sizes varying from 20 to 60 nodes with the step of 10 nodes. Furthermore, networks from this set have two different deployment areas, circular and rectangular.

The proposed algorithm is the only algorithm for solving this optimization problem to the best of the author's knowledge. However, load balancing algorithms result in more optimized network topology and reduced TreeMAC schedule length. Therefore, comparing the proposed algorithm with state-of-the-art load balancing algorithms

makes sense. Two such load balancing algorithms were used. Additionally, the most commonly used BFS search algorithm was simulated to provide a notion of how much topology optimization can influence TDMA MAC performance.

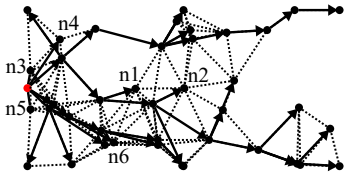
The first load balancing algorithm used is proposed by Ö.D. Incel *et. all.* [50]. This approximation algorithm tries to create a top-load balanced routing tree. When assigning a child node to a parent, the algorithm forms a search set for each possible child node. When this set is being formed for a node, all its neighbors located one and two levels above are considered. It is observed what happens with these nodes if the current node is chosen as the child. If this choice leaves a node from the search set without a choice for joining more than one top sub-tree, that node is added to the search set. Children with the smallest search set size are favored during the assignment.

The second load-balancing algorithm simulated is ETC (Energy Driven Tree Construction), proposed in [55]. This algorithm aims to create a tree as close to the fully load-balanced tree as possible. Such a tree is called a near-balanced tree. For that purpose, the ideal branching factor is defined as $k = \sqrt{h}N$, where h is tree depth, and N is the number of nodes. In a fully load-balanced tree, each node has the number of children equal to the value of k. To balance the tree, the algorithm instructs all nodes with more than k children to change the parent. This is achieved through the negotiation process, which uses protocol-specific packets.

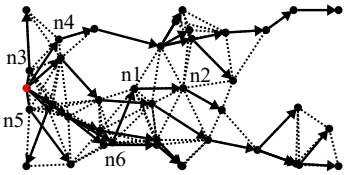
All four algorithms are implemented in the OMNeT++ simulator. To evaluate the algorithms, M-TreeMAC is run on the routing tree created by them. The schedule length is used as the performance metric of each algorithm. Simulations are performed for each network from the set, which contains networks with two different deployment area shapes, rectangular and circular. In addition, the set includes networks with a different number of nodes, for both shapes. Furthermore, there are different random networks in the set for each shape and number of nodes.

Fig. 68 shows two different routing trees constructed on the same rectangular network with 40 nodes. The first one is created using BFS algorithm and the second using maximum matching algorithm. Maximum matching algorithm reduces the number of low-level leaf nodes from 18 to 12. Six nodes, marked $n1$ to $n6$ on the figure, that leaf nodes when BFS tree construction algorithm is used, are successfully assigned a child when the proposed algorithm is used.

As the network size increases, the number of child assignment options increases. Therefore, it is expected that the proposed algorithms will reduce the schedule length more significantly as the network size increases. The dependency of schedule length on the network size is obtained by averaging the schedule length over networks with the same size and network deployment area shape (ten networks for each size-shape pair).



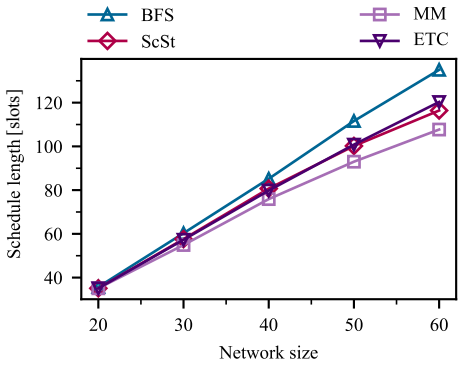
(a) BFS tree construction algorithm



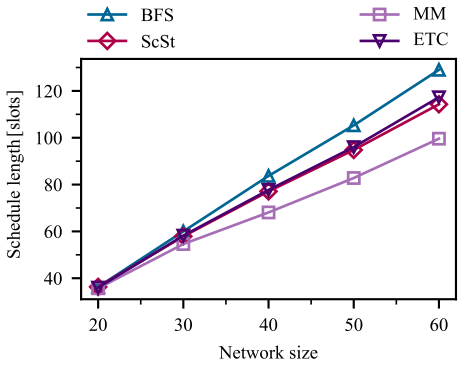
(b) Maximum matching tree

Figure 68: Routing trees constructed using BFS and maximum matching algorithms.

The results for both deployment area shapes are plotted in Fig.69. In the figure, the results for BFS algorithm are denoted by BFS. The algorithm by Ö.D. Incel *et. all.*, which utilizes search sets, is denoted by $ScSt$. Finally, ETC algorithm is marked by its name, and MM denotes maximum matching algorithm.



(a) Circular deployment area



(b) Rectangular deployment area

Figure 69: Dependency of schedule length on network size for different tree construction algorithms.

The results show that the proposed algorithm outperforms all the other algorithms in most cases considered. There is no improvement only when the number of nodes in the network is low. In the simulations performed, that occurred in all networks with 20 nodes. In these networks, all algorithms resulted in the same schedule length. The reason is that there are not enough alternative children nodes for assignment due to low network density. for the first larger network size considered, 30 nodes, the proposed algorithm results in reduced schedule length. In this case, the schedule length reduction is not very large, but it is still significant. However, as the number of nodes in the network increases further, the proposed algorithm results in a more substantial reduction in the schedule length.

4.4 Conclusion

State-of-the-art TDMA mac protocols for tree-based convergecast networks use many advanced techniques to reduce schedule length and adapt the schedule to the traffic flow in the network. However, most of them assume that one packet can be transmitted during each time slot. Unfortunately, that is often not the case, and multiple packet transmissions occur during a time slot. This leads to unused time slots and longer schedule lengths than necessary. This chapter proposes a TDMA protocol that considers multiple packet transmissions, named M-TreeMAC. The proposed protocol increases maximum throughput and reduces latency in linear networks significantly. For networks with a topology other than linear, throughput is slightly increased, but latency remains decreased considerably. Furthermore, it is observed that the length of an M-TreeMAC schedule depends on the routing tree. An algorithm that finds an optimal routing tree is proposed, increasing the performance of M-TreeMAC even further.

5 Adaptive Interference Model

TDMA protocols schedule many nodes in parallel, in order to reduce the schedule length and increase the throughput. To determine which nodes may transmit at the same time, an interference model is used. An interference model predicts the interference range of each transmission. If two nodes are within each other's interference range, they may not be scheduled in parallel. Otherwise, a collision occurs and the packet being transmitted is lost. A large number of state-of-the-art TDMA protocols rely on 2-hop interference model. According to this model, nodes that are two or fewer hops away from each other may not transmit at the same time. This model is used very often due to its simplicity; however, it is has low accuracy. Despite its lack of accuracy, resulting in high packet loss rates, it is still used because it can significantly reduce the implementation cost and complexity. Thanks to this model, protocols like TreeMAC are able to achieve significant throughput while keeping implementation costs low and network construction time short.

Alternatives for 2-hop interference model are protocol and physical interference models. Though these models have higher accuracy, they are significantly more complex to implement. The usage of these models results not only in higher network construction time and complexity, but also arises the need for installing additional hardware. For example, the protocol interference model uses the physical distance between nodes to calculate the interference range [59]. This requires either usage of a GPS module or some kind of metering module to measure distances between all nodes in the network. Moreover, when one of these two models is used, the network construction time increases drastically as the number of nodes in the network increases.

The physical interference model calculates the interference range based on signal strength measurements. For each receiver-transmitter pair, the transmitter is instructed to transmit a test signal and the receiver to measure the strength of the received signal. Based on this measurement, the signal-to-noise ratio (SNR) at all receivers is calculated. If this ratio is higher than a threshold value, set according to the properties of the physical layer, the parallel transmission under consideration may take place. There are two major drawbacks of this model. Firstly, the signal detection threshold of many radios is higher than the intensity of the signal which can cause interference. Therefore, it is not possible for the radios to detect all interference sources. One solution for this problem is proposed in [57]. The authors propose to use the highest possible radio power for detecting interference sources and lower ratio powers for data transfer. The obvious flaw is that the data transfer is limited to the second-highest transmission power, reducing the effective radio range. Secondly, usage of this model results in a

long network construction time. This is especially a problem in large networks because network construction is increased with the square of network size.

When designing a TDMA protocol, there are two choices. Either usage of 2-hop interference model or of more complex models. Usage of 2-hop interference model allows fast network construction and distributed implementation. This, however, comes at the cost of increased interference in the network and, consequently, a higher packet loss rate. On the other hand, protocol and physical interference models offer efficient interference reduction. However, this comes at the cost of long network construction time and increased complexity. Additionally, either radio range needs to be sacrificed [57] or additional hardware needs to be included.

As a compromise between the two options, an adaptive interference model is proposed in this thesis. This model is based on 2-hop interference model. The proposed model increases the interference range in hops from two to a higher value. The interference range in hops is chosen using an algorithm, which takes the physical layer, network geometry, and deployment environment into consideration. The proposed model keeps the simplicity of 2-hop interference model, allowing distributed implementation and fast network construction. At the same time, it significantly reduces the packet loss rate in comparison with 2-hop interference model.

5.1 Path-Loss and Radio Modeling

The adaptive interference model defines the interference range of each node in number of hops. To choose the interference range, the worst-case scenario, in regards to positions of the transmitter, the receiver, and the interfering node is identified. Then, the signal-to-noise ratio is estimated and the interference range adjusted so that the value of the signal-to-noise ratio remains under the maximal allowed threshold value. This maximal threshold value is denoted by SNR^{thr}. To estimate the signal-to-noise ratio value, two dependencies must be approximated using an appropriate model. These are the dependency of signal strength on distance from the signal origin (defined by the path-loss model) and bit error rate from signal to noise ratio (defined by the radio model). To achieve good performance, these two models must be chosen and adjusted to achieve a good match with real-world values.

The path-loss and the radio model should be chosen and adjusted for every application separately. To achieve the best results, the dependencies calculated using these models should be compared and fitted with measurement results. If this is not possible, the adaptive interference model can still be used, but it might not be as effective. If the application allows for that, the interference range may always be adjusted on-site, by observing the protocol performance. Namely, the interference range is always an integer

value (number of hops). Therefore, if the interference range estimated is higher or lower than the optimal one, detected by observing performance drop, it can be increased or decreased manually.

In the general case, when precise adjustment of the path-loss model is not possible, two-ray ground-reflection model can be used. In this thesis, all algorithms and guidelines use this model. It is shown, using a real-world example, that this model performs well in the case of networks deployed in outdoor areas. For networks deployed indoors, this model can be fitted, by adjusting the model parameters.

Two-ray ground-reflection model calculates path loss L using the equation:

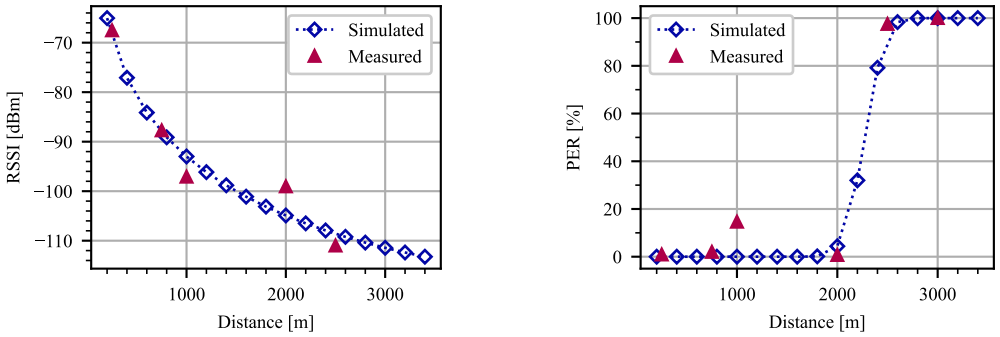
$$L = G_t G_r \frac{h_t^2 h_r^2}{d^k}. \tag{2}$$

In this equation, G_t and G_r are gains of transmitter and receiver respectively. Values h_t and h_r are the heights of transmitter and receiver. Term d is the distance between receiver and transmitter and k is a coefficient dependent on the environment geometry. In the case of the outdoor networks, k is equal to two. In the case of networks located indoors, the value of this parameter varies between two and four, based on the building geometry.

The dependency of bit error rate on the signal-to-noise ratio is defined primarily by the modulation scheme used for wireless communication. Models for many modulation schemes exist in WSN simulators like OMNeT++. These dependencies can also be adjusted based on measurement results if needed. For the application of the adaptive interference model, this is less critical, since deviations are usually lower than for path-loss.

To show that these models can be used to accurately represent real-world sensor networks, measurement results published in [60] were used. The measurements were performed using RC1780HP radio modules. To estimate the range and expected packet loss in a sensor network comprised of these modules, they were placed alongside a highway section. To perform the measurements, one main node (called gateway because it has access to the internet) transmits packets to every other node. The other nodes are placed at different locations, to obtain dependencies on the node distance. The gateway was configured to send seven packets to each node, every six and half hours. The system was running for almost three months. For calculation of the interference range, and later for the evaluation of the proposed interference range model, two-ray ground reflection models and QPSK modulation model from OMNeT++ were used. A comparison of the results obtained using these two models with the measurement results is shown in Fig. 70. The graphs in the figure show a very close match between

the simulated and measured radio performance. This justifies the suitability of the proposed approach for estimating interference due to parallel transmissions.



(a) Received signal strength

(b) Packet loss ratio

Figure 70: Comparison of measured and simulated signal strength and packet loss rate values.

5.2 Selecting the Interference Range

The first step in the interference range selection is the choice and adjustment of path-loss and radio models, as explained in Section 5.1. Afterward, the maximum allowed signal-to-noise ratio SNR^{thr} should be chosen based on the maximal permitted packet loss rate in the network and the radio model (dependence of bit error rate on signal-to-noise ratio). In a general case, the signal-to-noise ratio at the receiver is expressed as the ratio of powers of the useful signal (signal originating from the transmitter) and all other signals (radio noise). The radio noise includes background noise and signals from all interfering transmissions. To simplify calculations, adaptable interference model assumes that the power of background noise is negligible. Furthermore, only the closest interference source, the node located at the number of hops equal to the interference range, is considered. Under these assumptions, signal to noise ratio is given by:

$$SNR = \frac{P_{tx}}{P_{tx_i}}. \tag{3}$$

In this equation, P_{tx} denotes the power of signal coming from the transmitter, while P_{tx_i} denotes signal power coming from the closest interfering transmitter. If both of these radio modules are identical, i.e., have identical antenna gains and are located at the same height, the signal-to-noise ratio can be expressed in terms of distances between the receiver and the transmitter. Distance between the transmitter and the receiver is denoted by d_{tx} and between the transmitter and the interference source by

d_{tx_i}. Using equation 2 and expressing signal to noise ratio in a more commonly used unit, decibel-milliwatts, SNR at the receiver can be defined as:

$$SNR_{dBm} = k \cdot 10 \log_{10} \frac{d_{tx_i}}{d_{tx}}. \tag{4}$$

As this equation shows, signal to noise ratio is dependent on these two distances; the distance between the receiver and the transmitter and the receiver and the interfering node.

After the SNR^{thr} is defined, the interference range in hops should be selected so that the signal-to-noise ratio, given by equation 4, is lower than SNR^{thr}. The distance between the receiver and the transmitter is always fixed for the given receiver-transmitter pair. To reduce the signal-to-noise ratio, the distance to the interfering node should be increased by increasing the interference range in hops. Therefore, it makes sense to consider the ratio of these two distances. This ratio is called interference ratio and can be calculated as: $m = \frac{d_{tx_i}}{d_{tx}}$. Based on equation 3, the value of interference ratio m should satisfy the following condition:

$$m \geq 10^{\frac{SNR^{thr}_{dBm}}{10k}}. \tag{5}$$

To select the interference range, the value of the interference ratio should be expressed in terms of the interference range. To do so, the node placement that results in the lowest possible signal-to-noise ratio should be identified. The worst-case node placement in the general case is already published in [15] by the author of the thesis. However, such a worst-case is extremely unlikely in real-world sensor networks, resulting in an unnecessarily high interference range. In real-world applications, two nodes are rarely very close to each other. Instead, in many cases, minimal and maximal distances between two nodes in a network can be approximated. In this thesis, a more generalized adaptive interference model compared to the one in [15] is proposed.

Let Δ_{min} and Δ_{max} be minimal and maximal distances between two nodes and R the maximal radio range. The worst-case scenario can be identified for each interference range value in hops, and the interference ratio expressed in terms of Δ_{min} and Δ_{max}. When the interference range equal to two hops is considered, there are multiple cases, depending on the ratio between Δ_{min} and radio range R. Two of these cases are illustrated in Fig.71.

In the first case, the value of radio range satisfies the relation $R < 2\Delta_{min}$. The worst-case node placement for this value is shown on Fig. 71a. In this case, the interferer, node i, is at the distance equal to $2\Delta_{min}$ from the receveir, node r. The transmitter,

(a) The first case, $R < 2 \cdot \Delta_{min}$

(b) The second case, $2 \cdot \Delta_{min} \geq R < 3 \cdot \Delta_{min}$

Figure 71: Worst case scenarios when interference range of 2 hops is used.

node t, is located at the highest possible distance, equal to Δ_{max}. The interference ratio is given by $m = \frac{R}{2\Delta_{min}}$. In the second case, Fig. 71b, relation $2\Delta_{min} \geq R < 3\Delta_{min}$ holds. In this case, the interference factor is given by $m = \frac{R}{3\Delta_{min}}$.

To define an algorithm for selecting the interference range, it is necessary to determine the interference ratio m in the general case. The distance between two nodes that are two hops away from each other can be expressed in the general case as an integral multiple of Δ_{min}. Let n be the result integer division of radio range and Δ_{min}, $n = R/mod\Delta_{min}$. Then, the minimal distance between two nodes two hops away from each other can be expressed as $n\Delta_{min} + \Delta_{min}$. As the interference range in hops increases, the distance between the receiver and the interfering node increases. If the interference range in hops is denoted by r_{int}, then the interference ratio can be expressed as:

$$m = \frac{n(r_{int} - 1)\Delta_{min} + \Delta_{min}}{\Delta_{max}}. \tag{6}$$

Based on this equation, an algorithm for selecting interference range, algorithm 4 is designed. At the beginning, line 1, values of parameters dependent on network properties, SNR^{thr} and k , are chosen. The value of SNR^{thr} is set based on radio properties and desired maximal acceptable packet loss value. Path-loss coefficient k is chosen based on the network deployment environment. Values Δ_{min} and Δ_{max} are set based (line 2) on the estimation of distances between nodes in the network. Next, the value of n is calculated, and the interference range is initialed to two. Then, value of m is initialized using equation 6. Finally, interference range is incremented in while a loop, line 6, until the value of ratio m satisfies the relation given by equation 5. The value of the interference range after the termination of the while loop is the selected interference range value.

5.3 Evaluation

The effectiveness of the adaptive interference model is strongly dependent on the relative positions of the transmitter, receiver, and all interfering nodes. Hence, the interference range value, which performs well in one particular case, might perform poorly in the

Algorithm 4 Interference range selection

Select SNR^{thr} and k
Estimate Δ_{min} and Δ_{max}
$n \leftarrow R \mod \Delta_{min}$
$r_{int} \leftarrow 2$
$m = \frac{n(r_{int}-1)\Delta_{min}+\Delta_{min}}{\Delta_{max}}$
while $m < 10^{\frac{SNR^{thr}_{dBm}}{10k}}$ **do**
  $r_{int} = r_{int} + 1$
  $m = \frac{n(r_{int}-1)\Delta_{min}+\Delta_{min}}{\Delta_{max}}$

second. The adaptive interference model selects a range value that performs well in all networks that satisfy predefined topology constraints. Therefore, for the evaluation to be relevant, it must use a large number of randomly generated networks. As this can only be achieved using a simulator, OMNeT++ was used to evaluate the adaptive interference model.

Previous chapters analyzed the effects of schedule length and slot order on packet propagation in the network. Therefore, interference between parallel transmissions was not of primary concern, and the ideal radio model usage was appropriate. However, an accurate physical layer model is crucial for evaluating interference models using a simulator. For these reasons, the scalar radio model from OMNeT++ is selected. Furthermore, this model was fitted using measurement results to ensure realistic results.

The scalar radio model calculates the average signal-to-noise ratio during each reception. To do so, the scalar radio model averages values of all detectable signals during a reception. Then, the bit error and packet loss ratios are calculated based on the modulation scheme used. Finally, a random number generator is used to decide if the packet reception is successful or not, based on the number drawn and the packet loss rate calculated.

The adaptive interference model considers many parameters, including deployment environment, radio properties, and network topology. In this thesis, the evaluation is performed in two different cases. The availability of the measurement results influenced this choice. In both cases, an outdoor network consisting of nodes equipped with RC1780HP radio modules is considered.

The evaluation is performed using a large number of randomly generated linear networks. Linear topology is chosen because the primary source of packet loss in each linear sub-network is the transmitter-receiver with the lowest signal-to-noise ratio. Namely, for each receiver, the closest interfering transmitter is always located at the number of hops equal to the interference range. The packet loss ratio caused by this interfering

node depends on the interference ratio m, which depends on the node placement. The receiver-transmitter pair with the lowest value of interference ratio is the bottleneck for the sub-network it is located in. Such a pair can cause the loss of many packets, which previously traversed multiple hops.

As the network height increases, the chance of an unfavorable node placement, which would cause a high packet-loss ratio, increases significantly. Therefore, it is concluded that network heigh rather than size, makes the network more sensitive to interferences.

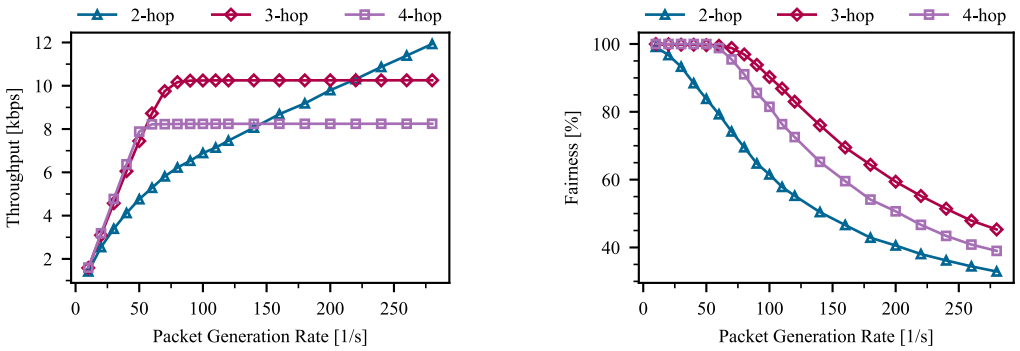
For the evaluation, two cases are considered. The difference between the two cases is the size of the data packet transmitted by the nodes; it is 20 in the first and 100 bytes in the second case. The larger the packet size, the larger the signal-to-noise ratio required for achieving an acceptable packet loss value. In the first case, SNR value of 8 dBm is sufficient, while in the second case, it must be increased to 10 dBm. This results in the interference range increasing from three to four. In both cases, network size is varied from 8 to 32 nodes. For each network size, ten different random networks are generated. Networks are generated so that Δ_{min} is 500m and Δ_{max} is 1500m. The radio range is 2000m, according to the radio model (Fig. 70b).

5.3.1 Small Packet Case (20 Bytes)

In the first case considered, all data packets generated by sensor nodes have equal size; they are 20 bytes large. By analyzing bit error ratio vs. signal-to-noise ratio dependency, it is concluded that a signal-to-noise ratio of 8 dBm (SNR^{thr}) or higher will result in a sufficient packet delivery ratio. Based on equation 5, the minimal interference ratio required to achieve this value of SNR^{thr} can be calculated. By substituting all values into equation 5, including $k = 2$ for outdoor scenario, the minimal interference ratio is $m \geq 10^{\frac{8}{20}} = 2.5$. The next step is to select the interference range that will guarantee an interference ratio greater or equal to this value. The algorithm starts with an interference range of two hops, $r_{int} = 2$, and increases it until the condition is met. In the first step, when $r_{int} = 2$, the interference ratio is equal to 1.6 (6). This is lower than the minimum acceptable value; therefore, the interference range is increased to three. For this interference range value, the minimal interference ratio is $m = 3$. As this satisfies the condition, the interference range of three hops is chosen, and the algorithm terminates.

To evaluate the proposed interference range selection algorithm and, at the same time, compare the adaptive interference model with the 2-hop interference model, the interference range is varied from two to four. This is repeated for all ten random networks of each size. One isolated case is considered before summarizing results and giving dependencies on network size. This is the case of one network comprised of

20 nodes and the network sink. Dependencies of network throughput and fairness on packet generation rate, in the case of all three interference range values, are shown in Fig.72.



(a) Throughput.

(b) Fairness.

Figure 72: Dependencies of throughput and fairness on packet generation rate in the case of a network with 20 nodes.

The results show that the interference range selected by the proposed algorithm, three hops, results in the best performance. Compared with the interference range of four hops, the chosen value results in higher throughput and fairness for all packet generation rates. On the other hand, the usage of the 2-hop interference range results in poor data delivery fairness, as the graph in Fig. 72b clearly shows. In this case, the throughput changes with the increased packet generation rate differently. In the beginning, it increases slower; however, it keeps rising after the throughput for the other interference range enters saturation. Eventually, it reaches values larger than when the three-hop interference range is selected. This happens when the packet generation rate reaches 200 packets per second; the value at which fairness drops below 50%. Therefore, the performance is not better, but most of the packets from the upper levels are lost due to interference. This allows nodes close to the sink to pass many packets to the sink, resulting in high throughput and low fairness.

The evaluation of the small packet case included 70 different networks of seven different sizes. The maximal generation rate is calculated for each network and interference range to summarize these results and represent them graphically. The maximal generation rate is defined as the generation rate at which the packet delivery fairness reaches the value of 90%. The results are then averaged for each network size. This gives the dependency of the maximal throughput on network size for different interference range values. The results are plotted in Fig. 73. The results show that the proposed interference range value, three hops, achieves the highest throughput for all

network sizes. Additionally, the results show that the proposed adaptive interference model significantly improves over the 2-hop interference model for all network sizes.

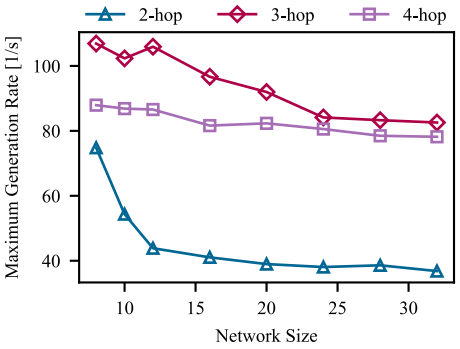


Figure 73: Maximal packet generation rate versus network size when 20 bytes long packets are used.

5.3.2 Large Packet Case (100 Bytes)

In this case, all data packets have the size of 100 bytes. Due to the larger packet size, the same bit error ratio will result in a higher packet error ratio than the first case (packet size of 20 bytes). Therefore, a lower bit error rate is targeted to meet the same performance requirements. Based on radio properties, a threshold signal-to-noise ratio value of 10 dBm is chosen in this case. By substituting this signal-to-noise ratio value into (5), a minimal interference ratio value of 3.2 is obtained. Therefore, the interference range must be increased to four to meet this demand.

The same set of simulations as for the small packet size is repeated to evaluate the adaptive interference model. Then, maximal throughput is calculated for each network and interference range. Finally, results are averaged over all networks with the same size to obtain dependency on network size. The results are shown in Fig .74. In this case, the interference range selected by the algorithm performs worse than the three-hop interference range when the network size is smaller than 20 nodes. The explanation is that the adaptive interference model chooses the interference range that would perform well in the worst-case node placement. However, when the network size is small, the chance of an unfavorable node placement is small, and a lower interference range performs better in most of the simulated networks. This results in a higher throughput on average. On the other hand, as network size increases, the chance of an unfavorable node placement gets higher, and the selected interference range performs better in most of the networks considered.

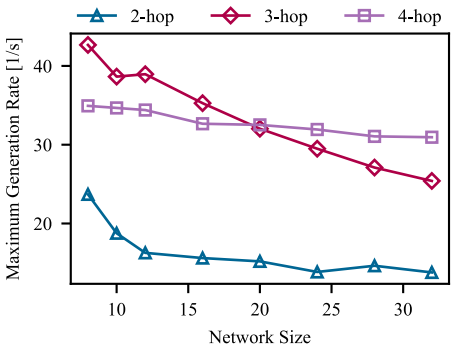


Figure 74: Maximal packet generation rate versus network size when 100 bytes long packets are used.

5.4 Conclusion

State-of-the-art TDMA protocols use either the two-hop interference model or a realistic interference model to determine which nodes may transmit in parallel (share a time slot). Two-hop interference model allows for distributed implementation and fast network construction times; however, it lacks accuracy, resulting in a higher packet loss ratio due to interference. On the other hand, realistic interference models effectively reduce interference and lower network packet loss rate. However, they are complex to implement, result in a long network construction time and require the usage of additional hardware modules. Adaptive interference model proposed in this chapter achieves a compromise between these two options. It can be used instead of the two-hop interference model with no modifications, keeping all the benefits the two-hop model offers. At the same time, it reduces interference and, subsequently, packet loss ratio significantly. The improved performance is confirmed through extensive simulations, which employ realistic radio and path-loss models.

6 Conclusion

Technological advances are leading to more widespread use of wireless sensor networks and their penetration into new fields and application scenarios. In some of these areas, device cost is a limiting factor, either due to the need for extensive networks (meaning a vast number of nodes) or for commercial reasons. Using complex standardized protocols is not an option in these applications due to costs and energy efficiency. When such networks require high throughput, custom TDMA protocols are the most common choice to achieve that using low-cost and energy-efficient hardware. This thesis studies this problem and provides guidelines for achieving the best results when designing a network stack for such a network.

The first contribution of the thesis is a comparative evaluation of state-of-the-art TDMA scheduling algorithms. No such study was previously done, to the best of the author's knowledge. Instead, very limited data was available about the performance of the existing scheduling algorithms; this made an algorithm selection for a particular application a guessing game. This thesis tested the six most advanced state-of-the-art scheduling algorithms using simulations performed on over 200 randomly generated networks. The results allow selecting the most appropriate algorithm for a particular application and estimating the throughput it can achieve.

The second contribution is M-TreeMAC, a TDMA protocol that improves performance when slot duration is longer than the transmission time of the data packets. State-of-the-art TDMA protocols are not considering multiple packet transmissions in a single time slot that occur in this case, leading to reduced fairness and increased packet delay. The proposed M-TreeMAC calculates the number of slots required by each node, taking into consideration multiple packet transmissions within the same slot. Its scheduling algorithm is a modification of the scheduling algorithms used by TreeMAC, hence the name. The proposed approach reduces average packet delay by up to 30%. Furthermore, a topology optimization algorithm is proposed. When this algorithm is combined with M-TreeMAC, delay can be reduced even more, up to 50% in comparison with TreeMAC.

The final contribution of this thesis is an adaptive interference model. Most TDMA protocols use the 2-hop interference model for scheduling parallel transmissions. This model allows centralized implementation and fast network construction. However, this model lacks accuracy, resulting in interference and an increased packet loss ratio; the impact of this effect becomes more pronounced as the network depth increases. The proposed adaptive interference model offers increased accuracy compared to the 2-hop interference model while keeping its simplicity; it can be used instead of the 2-hop

model without extensive modification, keeping centralized implementation and short network construction times. The evaluation of this model shows that its usage benefits networks with ten or more levels. Furthermore, if the network height is 15 hops or larger, the maximal throughput can be increased by over 80%.

This thesis proposes multiple solutions for increasing data throughput in the considered scenario. Nonetheless, there are still possibilities for future research. The most important topic to be researched is experimental evaluation of the adaptive interference model. This interference model was developed using a realistic radio model based on measurement results. Even though such model offers high precision, it is redoubtable that it deviates from real-world measurement to a some extent. Thus, the adaptive interference model can be further improved by adjusting it based on experimental results.

Besides that, there are two other possible directions for further research. The first one considers the recommendation for selecting the most appropriate schedule calculation protocols. This could be extended further and defined in a more formal manner. However, this would require a huge number of simulation. This thesis conducted simulations on 200 different networks. And for a more formal definition, around four times that number should be required, according to the author's estimation. The second topic for a further research is extending M-TreeMAC to advanced schedule calculation algorithms, like Park's algorithm. This would improve these algorithms even further, in scenarios where multiple packet transmissions in a single time slot are common.

References

[1] S. Zhang and H. Zhang, "A review of wireless sensor networks and its applications," in *2012 IEEE International Conference on Automation and Logistics*, pp. 386–389, 2012.

[2] S. R. Jino Ramson and D. J. Moni, "Applications of wireless sensor networks — a survey," in *2017 International Conference on Innovations in Electrical, Electronics, Instrumentation and Media Technology (ICEEIMT)*, pp. 325–329, 2017.

[3] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled tsch," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, (New York, NY, USA), p. 337–350, Association for Computing Machinery, 2015.

[4] K.-H. Phung, T. T. Huong, D. Khanh Dung, V. X. Tuong, T. Pham, T. Nguyen, and K. Steenhaut, "A scheduler for time slotted channel hopping networks supporting QoS differentiated services," in *2018 International Conference on Advanced Technologies for Communications (ATC)*, pp. 232–236, 2018.

[5] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Enhanced time-slotted channel hopping in WSNs using non-intrusive channel-quality estimation," in *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 217–225, 2015.

[6] "ISA-100.11a-2009," *An ISA Standard, Wireless systems for industrial automation: Processcontrol and related applications.*, 2009.

[7] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 377–386, 2008.

[8] D. Gislason, *Zigbee Wireless Networking.* USA: Newnes, pap/onl ed., 2008.

[9] "IEEE standard for low-rate wireless networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2015)*, 2020.

[10] N. Mirza and A. N. Khan, "Bluetooth low energy based communication framework for intra vehicle wireless sensor networks," in *2017 International Conference on Frontiers of Information Technology (FIT)*, pp. 29–34, 2017.

[11] H. Choi, J. Wang, and E. Hughes, "Scheduling for information gathering on sensor network," *Wireless Netw.*, vol. 15, pp. 127–140, Aug. 2007.

[12] J. Park, S. Lee, and S. Yoo, "Time slot assignment for convergecast in wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 83, pp. 70–82, 2015.

[13] N.-L. Lai, C.-T. King, and C.-H. Lin, "On maximizing the throughput of convergecast in wireless sensor networks," in *Advances in Grid and Pervasive Computing*, pp. 396–408, 2008.

[14] A. Ilić and M. Schölzel, "Packet Merging-driven Tree Construction in Wireless Sensor Networks," in *Fachgesprach Sensornetze*, 2018.

[15] A. Ilić, P. Langendoerfer, S. Weidling, and M. Schölzel, "Determining optimal parallel schedules in tree-based WSNs using a realistic interference model," in *Proceedings of Fifth International Congress on Information and Communication Technology*, (London, UK), pp. 572–582, 01 2021.

[16] G. Shanthi and M. Sundarambal, "FSO–PSO based multihop clustering in WSN for efficient medical building management system," *Cluster Computing*, vol. 22, pp. 12157–12168, Sept. 2019.

[17] M. Bottero, B. Dalla Chiara, and F. Deflorio, "Wireless sensor networks for traffic monitoring in a logistic centre," *Transportation Research Part C: Emerging Technologies*, vol. 26, pp. 99–124, 2013.

[18] E. Lawrence, K. F. Navarro, D. Hoang, and Y. Y. Lim, "Data collection, correlation and dissemination of medical sensor information in a wsn," in *Proceedings of the 2009 Fifth International Conference on Networking and Services*, ICNS '09, (USA), p. 402–408, IEEE Computer Society, 2009.

[19] G. G. Messier and I. G. Finvers, "Traffic models for medical wireless sensor networks," *IEEE Communications Letters*, vol. 11, no. 1, pp. 13–15, 2007.

[20] Y. Zhang, L. Sun, H. Song, and X. Cao, "Ubiquitous wsn for healthcare: Recent advances and future prospects," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 311–318, 2014.

[21] I. Talzi, A. Hasler, S. Gruber, and C. F. Tschudin, "PermaSense: Investigating permafrost with a WSN in the Swiss Alps," in *Proceedings of the 4th Workshop on Embedded Networked Sensors, EmNets 2007*, pp. 8–12, June 2007.

[22] A. Diaz-Ramirez, L. Tafoya, J. Atempa, and P. Mejia Alvarez, "Wireless sensor networks and fusion information methods for forest fire detection," *Procedia Technology*, vol. 3, p. 69–79, 12 2012.

[23] J. Lloret, M. Garcia, D. Bri, and S. Sendra, "A wireless sensor network deployment for rural and forest fire detection and verification," *Sensors*, vol. 9, no. 11, pp. 8722–8747, 2009.

[24] E. A. Kadir, H. Irie, and S. L. Rosa, "Modeling of wireless sensor networks for detection land and forest fire hotspot," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–5, 2019.

[25] H. Sharma, A. Haque, and Z. A. Jaffery, "An efficient solar energy harvesting system for wireless sensor nodes," in *2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pp. 461–464, 2018.

[26] M. Piñuela, P. D. Mitcheson, and S. Lucyszyn, "Ambient rf energy harvesting in urban and semi-urban environments," *IEEE Transactions on Microwave Theory and Techniques*, vol. 61, no. 7, pp. 2715–2726, 2013.

[27] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi, and W. Heinzelman, "Energy-harvesting wireless sensor networks (EH-WSNs): A review," *ACM Trans. Sen. Netw.*, vol. 14, apr 2018.

[28] P. Woias, F. Schule, E. Bäumke, P. Mehne, and M. Kroener, "Thermal energy harvesting from wildlife," *Journal of Physics: Conference Series*, vol. 557, p. 012084, 11 2014.

[29] A. Castagnetti, A. Pegatoquet, T. N. Le, and M. Auguin, "A joint duty-cycle and transmission power management for energy harvesting wsn," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 928–936, 2014.

[30] M. Michalopoulou and P. Mähönen, "A mean field analysis of CSMA/CA throughput," *IEEE Transactions on Mobile Computing*, vol. 16, no. 8, pp. 2093–2104, 2017.

[31] W. Song, R. Huang, B. Shirazi, and R. LaHusen, "TreeMAC: localized TDMA MAC protocol for real-time high-data-rate sensor networks," in *IEEE International Conference on Pervasive Computing and Communications*, pp. 1–10, 2009.

[32] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *SenSys'04 - Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, pp. 134–147, Jan. 2004.

[33] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. 01 2011.

[34] "Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016.

[35] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks," *ACM Sensys*, vol. 4, pp. 307–320, 01 2006.

[36] H. Wang, X. Zhang, F. Naït-Abdesselam, and A. A. Khokhar, "DPS-MAC: An asynchronous MAC protocol for wireless sensor networks," in *HiPC*, 2007.

[37] E.-Y. Lin, J. Rabaey, and A. Wolisz, "Power-efficient rendez-vous schemes for dense wireless sensor networks," in *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, vol. 7, pp. 3769–3776 Vol.7, 2004.

[38] S. Liu, K.-W. Fan, and P. Sinha, "CMAC: An energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks," in *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pp. 11–20, 2007.

[39] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," *Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No.04TH8769)*, vol. 1, pp. 244–251 Vol.1, 2004.

[40] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, (New York, NY, USA), p. 1–14, Association for Computing Machinery, 2008.

[41] P. Huang, C. Wang, L. Xiao, and H. Chen, "RC-MAC: A receiver-centric medium access control protocol for wireless sensor networks," in *2010 IEEE 18th International Workshop on Quality of Service (IWQoS)*, pp. 1–9, 2010.

[42] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, 2000.

[43] Y. Sun, S. Du, O. Gurewitz, and D. B. Johnson, "DW-MAC: A low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks," in *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '08, (New York, NY, USA), p. 53–62, Association for Computing Machinery, 2008.

[44] T. van Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, (New York, NY, USA), p. 171–180, Association for Computing Machinery, 2003.

[45] W. Ye, F. Silva, and J. Heidemann, "Ultra-low duty cycle MAC with scheduled channel polling," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, (New York, NY, USA), p. 321–334, Association for Computing Machinery, 2006.

[46] I. Rhee, A. Warrier, J. Min, and L. Xu, "DRAND: Distributed randomized TDMA scheduling for wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 10, pp. 1384–1396, 2009.

[47] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-MAC: A hybrid MAC for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 511–524, 2008.

[48] L. Bao and J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, (Rome, Italy), pp. 210–221, 01 2001.

[49] M. Alicherry, R. Bhatia, and L. E. Li, "Joint channel assignment and routing for throughput optimization in multiradio wireless mesh networks," *IEEE J. Sel. Areas Commun.*, vol. 24, pp. 1960–1971, Nov. 2006.

[50] Ö. D. Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi, "Fast data collection in tree-based wireless sensor networks," *IEEE Trans. Mob. Comput.*, vol. 11, pp. 86–99, Jan. 2012.

[51] G. Shashidhar, Z. Ying, and H. Qingfeng, "Distributed time-optimal scheduling for convergecast in wireless sensor networks," *Comput. Netw.*, vol. 52, pp. 610–629, Feb. 2008.

[52] W. Wang, Y. Wang, X.-Y. Li, W.-Z. Song, and O. Frieder, "Efficient interference-aware TDMA link scheduling for static wireless networks," in *IEEE GLOBECOM*, pp. 262–273, 2006.

[53] P.-H. Hsiao, A. Hwang, H. Kung, and D. Vlah, "Load-balancing routing for wireless access networks," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 2, pp. 986–995 vol.2, 2001.

[54] J. He, S. Ji, Y. Pan, and Y. Li, "Constructing load-balanced data aggregation trees in probabilistic wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1681–1690, 2014.

[55] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras, "ETC: Energy-driven tree construction in wireless sensor networks," in *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, pp. 513–518, 2009.

[56] A. Ichrak and P. Minet, "TRASA: TRaffic Aware Slot Assignment algorithm in wireless sensor networks," *International Conference on Communications and Information Technology - Proceedings*, 09 2012.

[57] S. C. Ergen and P. Varaiya, "PEDAMACS: power efficient and delay aware medium access protocol for sensor networks," *IEEE Trans. Mob. Comput.*, vol. 5, pp. 920–930, July 2006.

[58] H.-W. Tsai and T.-S. Chen, "Minimal time and conflict-free schedule for convergecast in wireless sensor networks," pp. 2808–2812, 01 2008.

[59] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. Inf. Theory*, vol. 46, pp. 388–404, Mar. 2000.

[60] S. Weidling, N. Todtenberg, T. Basmer, M. Schoelzel, M. Maaser, and M. Taubert, "Outdoor range measurements in sub-ghz license-free radio bands under realistic conditions," in *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*, MobiWac '17, (New York, NY, USA), p. 71–74, Association for Computing Machinery, 2017.