CHALMERS
UNIVERSITY OF TECHNOLOGY

Towards Wireless Communication with Bounded Delay

Master's thesis in Computer Systems and Networks

Henning Tuan Hy Phan

Department of Computer science
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:03

Towards Wireless Communication with Bounded Delay

Henning Tuan Hy Phan

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer science
Chalmers University of Technology
Gothenburg, Sweden 2016

Supervisor: Thomas Petig, Department of Computer Science and Engineering
Supervisor: Olaf Landsiedel, Department of Computer Science and Engineering
Examiner: Elad Schiller, Department of Computer Science and Engineering

Master's Thesis 2016:03
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg

Typeset in LaTeX
Gothenburg, Sweden 2016

Abstract

Many of the existing implementations of Wireless Sensor Networks (WSN) cannot be considered for safety critical systems that require a constant upper bound time delay delivering messages. To deal with that, Petig, Schiller, Tsigas proposed a self-stabilising Time Division Multiple Access (TDMA) Protocol without leaders and external references [1]. The system settings that Petig *et. al's* assume consider the case in which ambient noise does not exist and in the absence of concurrent transmission packets are always received. This work considers the implementation of Petig *et. al* in real systems. Namely, we consider message reception behaviour that is subject to ambient noise and other complex radio propagation patterns. We show that the strategy proposed by Petig *et. al* for mitigating the effect of concurrent transmissions is applicable for working WSN systems. We also propose to build atop Petig *et. al's* robust algorithm and use a masking technique we call link reliability to provide mitigation against repeated packet drop due. Simulations and real experiments show distinct benefits of utilising link reliability.

Keywords: time divided multiple access, self-stabilisation, link reliability, clock synchronisation, wireless sensor network, real time system, TinyOS

Acknowledgements

I would like to take some time here to show my gratitude for my sister Hilda Phan who not only supported me but also helped me finish the thesis. I want to extend my thanks to Thomas Petig, his insight in the project were invaluable when discussing the implementation. Also thank you Elad Schiller and Olaf Landsiedel for participating.

Henning Tuan Hy Phan, Gothenburg, March 2016

Contents

1 Introduction **1**
- 1.1 Related work . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 2
- 1.4 Limitations . . . 3

2 Technical Background **5**
- 2.1 CSMA . . . 5
- 2.2 TDMA . . . 5
- 2.3 TinyOS CSMA architecture . . . 6
- 2.4 TDMA timeslot alignment . . . 8
- 2.5 Self-stabilisation . . . 8

3 Architecture **11**
- 3.1 TinyOS TDMA application . . . 11
- 3.2 TDMA subcomponents . . . 13
  - 3.2.1 TDMA component send and receive process . . . 14

4 Implementation **17**
- 4.1 Terminology and variables . . . 17
- 4.2 TDMAPST: Algorithm description . . . 18
- 4.3 TDMAPST: Detailed algorithm description . . . 18
  - 4.3.1 Algorithm 1: Upon alarm . . . 18
  - 4.3.2 Algorithm 1: Upon SFD . . . 19
  - 4.3.3 Algorithm 2: Upon receive . . . 19
- 4.4 TDMA with link reliability: Link reliability design . . . 24
- 4.5 TDMAwLR: Detailed algorithm description . . . 24
  - 4.5.1 Algorithm 4: Upon alarm . . . 24
  - 4.5.2 Algorithm 5: Upon receive . . . 24
  - 4.5.3 Algorithm 7: Link reliability functions . . . 25
- 4.6 Self-stabilisation . . . 30
  - 4.6.1 Safe configuration . . . 30
  - 4.6.2 Legal execution . . . 30
  - 4.6.3 Convergence . . . 30
- 4.7 TDMA packet header . . . 31

5 Results **35**
5.1 Setup . . . 35
5.2 Cooja experiments . . . 37
5.2.1 Convergence time . . . 37
5.2.2 Gridlayout 100% transmit success rate . . . 39
5.2.3 Gridlayout 80% transmit success rate . . . 42
5.2.4 Linelayout 80% and 31% packet success rate . . . 45
5.3 Experiments on Indriya . . . 47

6 Discussion **51**

Bibliography **53**

A Appendix 1 **I**
A.1 TDMA component interface . . . I

1

Introduction

There has been an increased interest world-wide for WSN, they have huge potential in many sectors, *e.g.*, industry, military, research and others. The WSN market has been forecast to become a multibillion activity if there is major advancement in standards and technology [2]. Existing implementations of WSN cannot be used for safety critical systems that require a constant upper bound time delay for delivering messages. Carrier Sense Multiple Access (CSMA) protocols do not guarantee that outgoing packages will be received within a well defined time interval. To counter that, Petig *et. al* [1] proposed a self-stabilising TDMA, protocol. Their TDMA protocol being the foundation of our implementation in which the self-stabilising part has been given extra assistance by utilising a new feature called "link reliability", which we have created to handle external disturbances.

The challenge is to produce an implementation in WSN using an embedded platform such as TelosB motes [3] and Tiny Operating System (TinyOS) [4]. Achieving a (soft) real time implementation given the limited resources that the studied platform has in the presence of packet omission due to ambient noise is another challenge to which we aim at providing a degree of link reliability. The implementation in the considered platform also includes challenges related to the dynamic communication environment, node crashes and timing failures due to drifts of inexpensive clocks and that TinyOS does not guarantee (hard) real time computation. Furthermore, the protocol [1], assumes 100% transmission success rate between neighbouring nodes. Note that the assumption that Petig *et. al* does not hold for WSN. In WSN, packet drop can vary arbitrarily between 0 and 100%. Our proposal is to build upon Petig *et. al's* robust algorithm and use a masking technique for further providing mitigation against repeated packet drops that are due to concurrent transmissions by neighbouring nodes. The system development also require tailoring evaluation tools for debugging and testing the proposed protocol in a large scale testbed.

1.1 Related work

Sgora et. al's survey [5] on WSN TDMA protocols classifies existing TDMA protocols and presents the benefits and caveats of different approaches. It would classify Petig et. al's TDMA protocol [1] as a Media Access Control (MAC) solution, node scheduled, distributed, topology-dependent TDMA protocol. Other attributes it

has are self-stabilisation, and clock synchronisation without external reference. On the topic of clock synchronisation, Herman and Zhang [6] present the clock synchronisation concept converge-to-maximum, which is used by Petig *et. al*, though clock synchronisation in *ad-hoc* network has it's limitation as described by Fan et. al [7]. There exist TDMA algorithms that do not synchronise clocks or use reference pulses but instead make assumptions on total number of neighbours and use a large enough frame size to accommodate all neighbours [8]. Herman and Tixeuil [9] who created the first self-stabilising TDMA algorithm for wireless *ad-hoc* networks. They use an external reference to create timeslots which are assigned according to the neighborhood topology.

1.2 Motivation

The purpose of this thesis is to produce a TDMA implementation inspired by Petig *et. al's* [1], for an embedded platform and test the performance on a real testbed. Our implementation is referred as TDMAPST to honour Petig, Schiller, and Tsigas. TDMAPST is self-stabilising but will have degraded performance because of (intermittent) faults due to ambient noise which will degrade the transmission success rate. To increase the robustness we introduce a new feature, link reliability, which aims to increase robustness in radio links. TDMAPST with link reliability will henceforth be referred to as TDMAwLR, to distinguish it from TDMAPST.

1.3 Objectives

To create TDMAPST, an implementation heavily inspired by Petig *et. al's* TDMA protocol [1] for the embedded platform telosB, using the TinyOS framework and programming language NesC. We use the word "inspired" because it diverges slightly from the origin because of the limited processing power and resources of TelosB motes. The product is a (soft) real time system, capable of handling a dynamic environment and changes to the network topology.

Active nodes in TDMAPST require received packets to contain an acknowledgement for them to retain their status. Becoming passive is an important mechanism to ensure collision free communication between nodes but should sometimes be avoided, *e.g.*, a few missed acknowledgements because of ambient noise. Our solution is called link reliability which works by delaying the decision in those cases. By collecting data per timeslot on packet success rate and packet acknowledgements it can make an informed decision whether it should become passive by comparing the gathered data and the expected packet success rate.

1.4 Limitations

The thesis explore the use of a TDMA mac protocol without an external reference thus bandwidth utilisation or throughput is not considered. A radio environment is considered where node A can communicate with Node B, and B will eventually communicate with Node A. Interference due to ambient noise or concurrent transmissions is not distinguised between. Link reliability is non-adaptive, as it is a proof of concept, which means a static packet success rate for all environments which we can precomputate is assumed. Lastly, only WSN applications which requires repeated communication are considered therefore we will not consider power conservation.

2

Technical Background

2.1 CSMA

The MAC layer is responsible for channel access mechanisms to allow multiple nodes to communicate on shared media. Pure ALOHA [10] is a pioneering MAC protocol and used in the demonstration of the first wireless packet transfer. When any node has a packet to transmit it immediately sends it. Collisions occur when the transmissions of at least two packets overlap in time. This could lead to packet omission or corruption. If a packet is omitted the transport layer is responsible to retransmit. Carrier Sense Multiple Access with collision avoidance (CSMA/CA) [11] [12] is also a decentralised MAC protocol inspired by ALOHA net that reduces the probability of retransmissions by prior to transmission the node senses the medium if anyone else is currently transmitting. If an ongoing transmission is detected the node's transmission is deferred by waiting a random period of time (random back off) to reduce contention and by that avoiding collisions. CSMA has a well known unbounded packet delay because of the random backoff and random access to the media that can be solved by a more structured scheduling approach.

2.2 TDMA

TDMA is a schedule-based MAC approach where access time to the channel is divided into frames which are further divided into slots, see figure 2.1. These slots are distributed to nodes allowing them to transmit during that slot to avoid collisions. In FreeBSD 8.0's [13] implementation of TDMA a master station is used to broadcast a beacon frame containing TDMA settings and free slots. Client nodes can adjust their timers after the beacon frame. A client can request a free slot by sending a beacon frame to the master which acknowledges it in the next beacon frame.

To avoid unwanted collisions, it is common to use a single node as an access point that allocates timeslots upon request. There exists decentralised TDMA approaches *e.g.*, Leone and Schiller [14], that instead of depending on the availability of the access point to nodes use coloring algorithms for allocating the timeslots by themselves. Leone and Schiller assume the availability of a common synchronised clock and use this assumption for aligning the TDMA timeslots. Once the timeslots are

aligned, the node can use random colouring techniques for allocating the timeslots. Self-stabilisation often have a problem when it comes to clock synchronisation and bounded communication delay. Clock synchronisation algorithms often rely on MAC algorithms that offer bounded communication delay while MAC algorithms often assume access to synchronised clocks. Petig *et. al* proposed a bootstraping solution for the dilemma in [1]. TDMA has more packet overhead as it requires that to coordinate the schedule but that allows the packet delay time to be bounded which might be an important requirement for some safety critical systems.

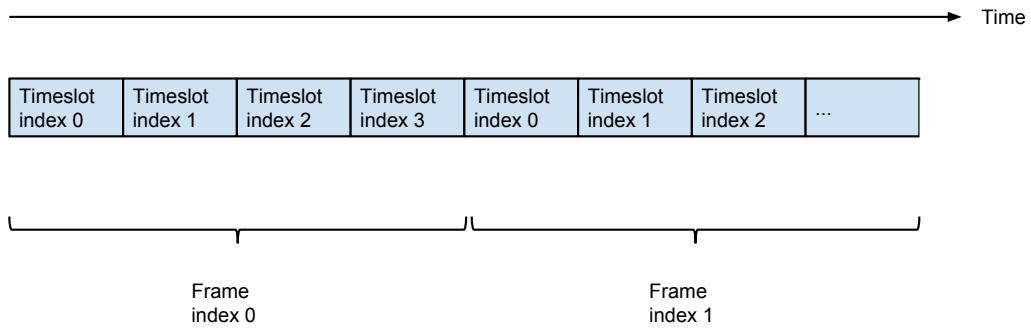

Figure 2.1: Visual representation of TDMA with four timeslots per frame

Mustafa *et al.* [15] consider the alignment of TDMA timeslots, but not the timeslot allocation as we do.

2.3 TinyOS CSMA architecture

TinyOS is an embedded operating system (OS) platform used in WSN. TinyOS is event driven and applications are built with components, which usually abstract the hardware and interact between each other by interfaces and deferred procedure calls. A typical TinyOS CSMA application is shown in figure 2.2 which shows the how the componentns are connected to create the application. By modifying the CSMA controller we get our TDMA application, see figure 3.1. A component description can be found in section A.1.

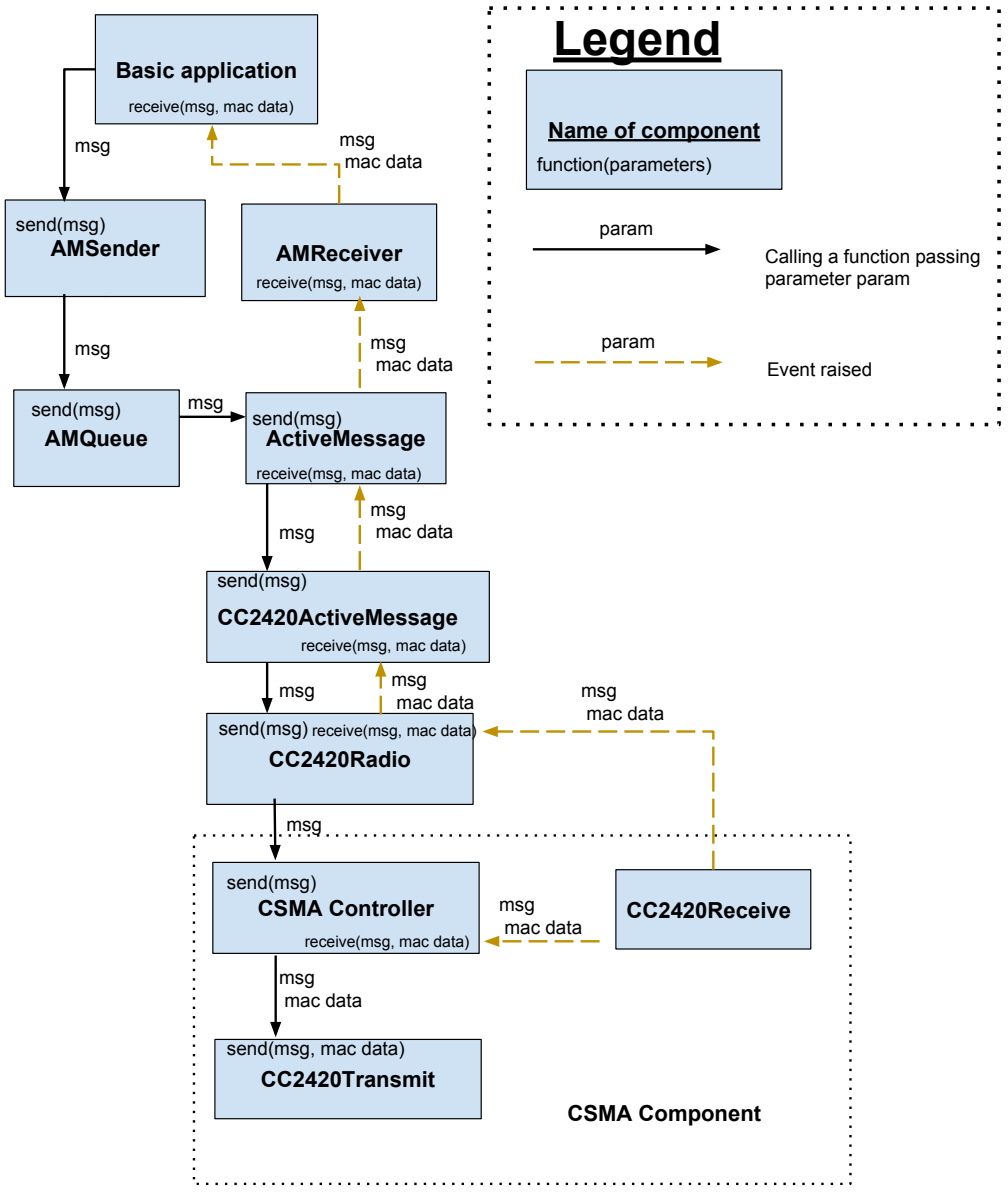

Figure 2.2: TinyOS CSMA architecture scheme

2.4 TDMA timeslot alignment

Synchronised clocks on nodes with aligned timeslots can significantly reduce the exposure time of collisions. In slotted ALOHA [16] with aligned nodes, collisions occurs when multiple sources transmit on the same timeslot. When nodes are misaligned collisions also occurs if a transmit is recorded on a neighbouring timeslot.

Various strategies has been developed to align timeslots. They usually synchronise clocks by provide a global timescale or a relative time. Examples are the use of a central node with the responsibility to propagate timestamps, which other nodes can adjust themselves after or alternatively use an external reference such as Global Positioning System (GPS) [17]. Commercial GPS receivers provide a global timescale which can achieve an accuracy better than 200ns relative to Coordinated Universal Time (UTC). There are also algorithms that does not propagate time at all but rely on pulses to align the timeslots [18].

Petig *et. al's* TDMA protocol synchronise the clocks by embedding the converge-to-maximum protocol [6] [19] to theirs. Converge-to-maximum is a distributed self-stabilising clock synchronisation protocol without leaders or dependency to external references. Converge-to-maximum has two rules: Firstly, a node's clock time is periodically transmit to its neighbours. Secondly, If the received clock value is greater than the local time, then advance the clock for the time difference so they are now equal. Assuming a bi-directed connected graph where nodes start with random clock values with no skews, all transmissions always succeed, and the packet delay is smaller than the margin of error. There most be at least one node with the highest clock value and that or those nodes will not update their clocks anymore because of rule two. Because their clock values are propagated the system will eventually all converge to the maximum clock value.

2.5 Self-stabilisation

E.W Dijkstra's seminal work of self-stabilisation in 1973 [20] remained somewhat unknown until L. Lamport praised the work in 1983 to be a milestone in the field of fault tolerance.

The interleaving model considers the system state as a collection of all the processors state together with their incoming messages. The system execution starts from a state that follows by a step that a single processor takes. That step brings the system to another state from which further steps are taken and other states are reached. The set of legal executions includes all the executions that satisfy the task requirements. Starting from an arbitrary state, a self-stabilising system reaches a legal execution within a bounded number of steps. A safe system state is a state after which only a legal execution follows.

There are two important properties in self-stabilisation: Convergence, a self-stabilising system is designed to start in any arbitrary state and after a bounded number of steps

reach a safe configuration. Closure, if the system is already in a safe configuration it will stay in a safe configuration given no faults occur. A system that satisfies both properties is said to be self-stabilising and when that system experiences (transient) faults it is capable to automatically recover.

3

Architecture

Our TDMA application is built on top of TinyOS, which uses a modular approach for building applications. TinyOS uses components which usually encapsulate a functionality or a hardware abstraction. This chapter describes which components are used building our TDMA protocol and how they are connected. Furthermore, we look at how the TDMA component is built by looking at its subcomponents.

3.1 TinyOS TDMA application

Our TDMA protocol transmits and receives packets. It is built with several components which figure 3.1 shows, where the black arrows with the text represent function calls to transmit a payload, denoted as msg in figure 3.1. It transmits the msg by using the component AMSender which indirectly uses AMQueue, ActiveMessage, CC2420ActiveMessage, CC2420Radio, TDMA Controller, and lastly CC2420Transmit. The components provide a send function, outgoing buffer queue, highest layer in the TinyOS radio stack, connects to high level abstraction of hardware, media access control, and lastly hardware abstraction, respectively. Note that the TDMA controller adds mac data, TDMA header, to the outgoing transmission and the header content is described in 4.1.

The yellow arrows represent a receive event, which contains transmit data and MAC header, and by following the yellow arrows leads back to the core component called basic application. The components CC2420Receive, CC2420Radio, CC2420ActiveMessage, ActiveMessage, and AMReceiver provides hardware abstraction, internal interface of the CC2420 stack to retrieve messages, connects to high level abstraction of hardware, provides the highest internal interface to other components, and provides a receive function to the application, respectively. The receive event also reaches component TDMA controller to provide it with necessary information to be able to provide MAC.

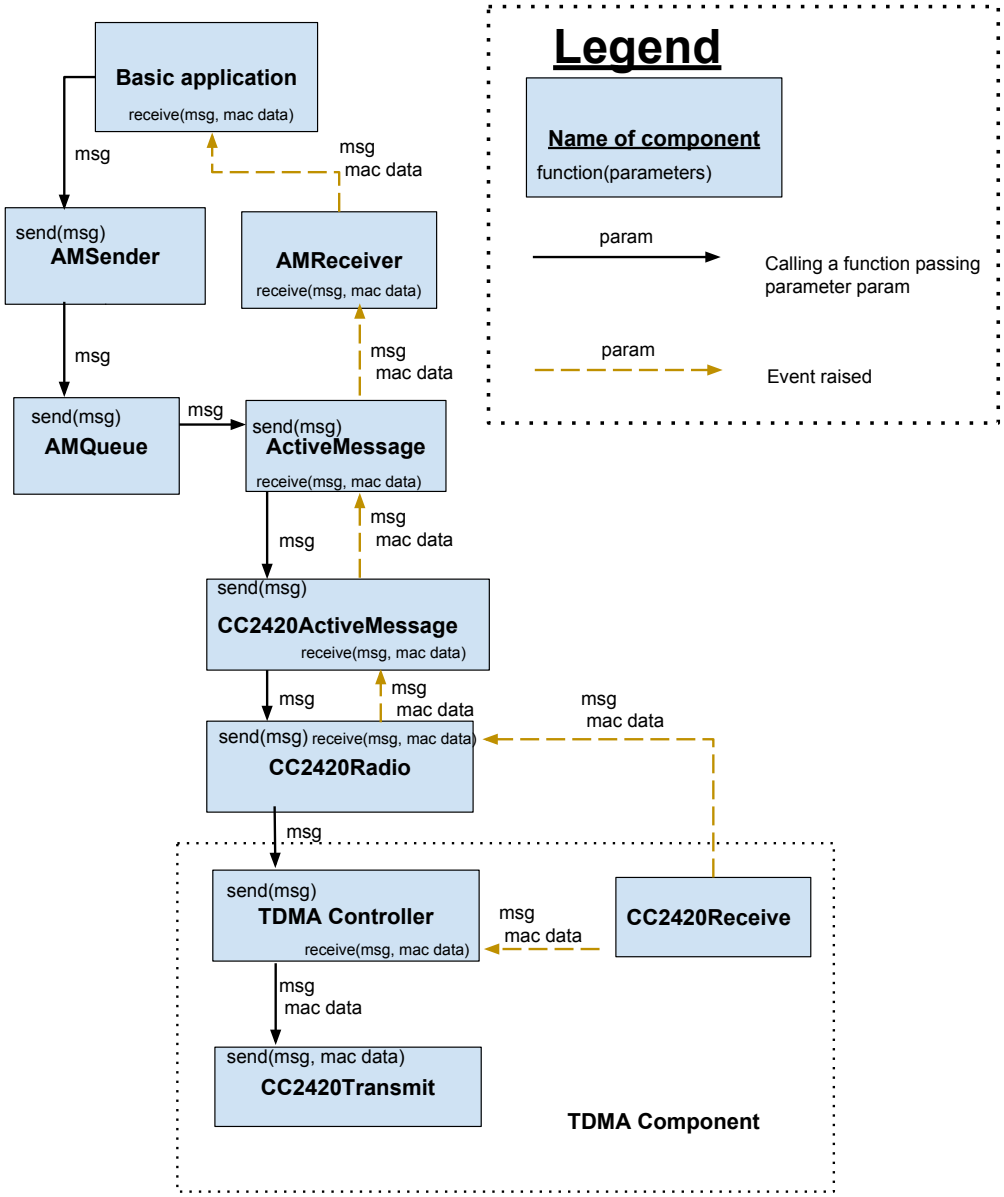

Figure 3.1: TinyOS architecture scheme. Only the most relevant components appear

3.2 TDMA subcomponents

For the TDMA controller to provide MAC protocol it needs to be notified of hardware events. Each hardware event is abstracted as a component which is connected to the TDMA controller, see figure 3.2. Each subcomponent is described in the following paragraphs.

Controller:
All the TDMA logic and information is stored in this component. All others subcomponents are either directly or indirectly connected to the controller. The unit decides when to access the media.

Alarm:
Provides an event called Alarm which is raised at the beginning of every timeslot at the controller.

GPIOCapture:
When the start frame delimiter (SFD) bit sequence is detected in a transmission GpioCapture raises the SFD event. The SFD event which is assumed to happen simultaneously at both sending and receiving node is the event that is timestamped and by comparing the timestamps we can get the clock difference.

CC2420Receive:
When a transmit is completed CC2420Receive passes the fully received message and passes it to the controller. It also passes the message to CC2420Radio which is responsible to forward it to the application.

CC2420Radio:
If the application wants to transmit data it goes through CC2420Radio which delegates the task and the timing when to transmit the message to Controller. If a packet is received CC2420Radio forwards it to the application.

CC2420Transmit:
CC2420Transmit provides the function transmit which loads the buffer with data and afterwards starts transmitting it. The function modify modifies the buffer and is capable of modifying during an ongoing transmission. It is used to change an ongoing transmissions timestamp to the timestamp of the SFD event.

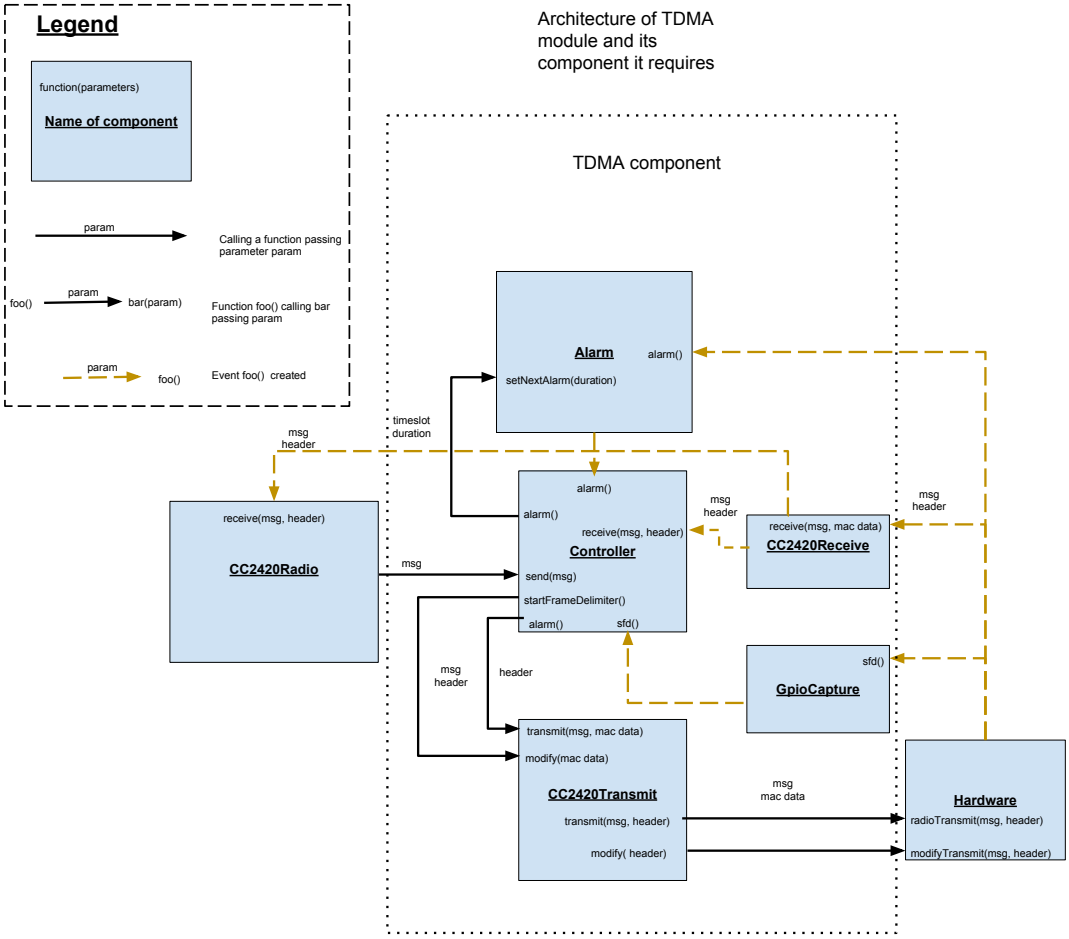

Figure 3.2: Figure of TDMA component's subcomponents. How they are connected and what functions and events they provide to enable the TDMA unit as a whole to work.

3.2.1 TDMA component send and receive process

The TDMA component expects events to occur in a specific order with room to have time to process them. The pseudocode in tables 3.1 and 3.2 explains which action the component takes to handle packet transmission and reception and which underlying events are triggered to facilitate the task.

Table 3.1: Table shows the pseudocode how the TDMA component handles packet transmission

1. The Alarm component raises the alarm event at the Controller component and updates variables related to TDMA.
2. Controller reaches the conclusion to send a packet and calls CC2420Transmit's transmit function and passes a new header and message.
3. CC2420Transmit passes the data to the hardware transmission buffer, a process slow enough to expire the header timestamp, and then calls the hardware to start transmitting.
4. During the hardware's transmission several things happens to accomplish late timestamping without ever stopping the transmission.
   4.1. The first byte transmitted is an SFD byte which triggers the hardware SFD event which raises the SFD event in the controller.
   4.2. Because the node's own transmission triggered the SFD event it calls CC2420Transmit modify function to update the expired timestamp. Hardware might have already started sending part of the header but not yet the timestamp thus an updated timestamp will be transmitted instead.
   4.3. Hardware completes sending header, msg and finishes by sending an SFD byte.

Table 3.2: Table shows the pseudocode how the TDMA component handles packet reception

1. The Alarm component raises the alarm event at the Controller component and updates variables related to TDMA.
2. Hardware captures an SFD byte at the beginning of a neighbours transmission, and raises the SFD event in controller.
3. The SFD event function locally timestamps the incoming packet with the current time.
4. After receiving the whole message, component CC2420Receive raises event receive in Controller.
5. Controller processes the header by checking for clock alignment, and other conflicts. Appropriate action will be taken depending on type of conflict.

4

Implementation

This chapter explains how TDMAPST and the extended version with link reliability works by first going through the terminology and variables necessary for the algorithm followed by an overview description of TDMAPST and then a full line by line explanation of the pseudocode. As link reliability is a feature we explain why its needed and the design choices followed by an overview description of link reliability and then a line by line explanation of the modified pseudocode but only the lines that are changed.

4.1 Terminology and variables

Distance one neighbours refers to all nodes that can receive a transmission from the reference node. Distance two neighbours are all nodes included from distance one and additionally all nodes that can receive a transmission from any distance one neighbour. The number of timeslot that makes up a frame is called frame size and in TDMAPST frames also have indexes and of the same size like timeslots and they are called super frames.

Each node has a set of variables to track the node's perception of the environment. *src*: Is a unique value that identifies the node. *wait*: is integer used as a counter to decrement to zero for the random back off strategy. *status*: a boolean representing if the node is active or passive. *s*: contains a timeslot index or is empty when the node is active or passive respectively. $rxAck$: an array containing $src's$ of neighbours that recently has successfully transmitted a packet. Each index in $rxAck$ refers to a timeslot index so the information on what timeslot the received packet was received on is preserved. *usedSlots*: Mapping of timeslot indexes and if they are in use by others.

The TDMA header, which is included in all packets, contains *s*, *src*, and $rxAck$ from the node variables and two new fields. Firstly field, $isCPKT$, tells us if the packet is a data or a control packet. The last field is *time*, a timestamp which marks when the SFD byte was detected by the radio to facilitate clock synchronisation.

4.2 TDMAPST: Algorithm description

A node's *status* is either passive or active. A passive node strives to become active by performing the random backoff strategy. The random backoff strategy uses *wait* which has been initialised with a random bounded integer. $Wait$ is decremented if the previous timeslot was unused until *wait* reaches zero. When *wait* is zero the node will transmit a control packet on the first timeslot that is free according to the *usedSlots* map to notify others that the timeslot has been claimed and sets its *status* to active and s to the timeslot index. *wait* is initialised again with a bounded random integer to be used by the now active node.

An active node always transmit a data packet on its timeslot. It also uses a modified random backoff strategy to send control packets. The modification is that *wait* is only decremented on unused timeslots on the frame index equal to s. The control packet is transmitted when *wait* is zero on the first free timeslot according to $freeSlot$ during the frame which has the same index as s. The control packets transmitted by active nodes are used to detect *e.g.*, timeslot collisions and are essential for the algorithm to ensure that all timeslots claimed by nodes do not interfere with each other.

When a node, regardless of status, receives a packet it contains a TDMA header. If the time field in the header is greater than the local time timestamp the local time is updated to the time in the header and status set to passive. Collision control is done to ensure that the transmitting node does not use the same timeslot, or other conflicts when updating $rxAck$ and *usedSlots* with the information in the header. If a conflict is found the node updates it status to passive and starts the random backoff strategy.

4.3 TDMAPST: Detailed algorithm description

This section explains how the code for TDMAPST works complemented with pseudocode.

4.3.1 Algorithm 1: Upon alarm

Alarm is the event executed at the beginning of every timeslot (line 4). Set the next alarm event for the next timeslot to ensure periodicity of event (line 5). Active nodes transmit their data packets upon their timeslots (line 6-7). Line 8 to 13 will be explained twice, from a passive and active node's perspective.

Passive node's perspective:
If *wait* is greater than zero and no transmission was detected on the previous timeslot, the previous timeslot is used because in hindsight we know that it was unused, then *wait* will be decremented (line 8-9). If the variable *wait* is zero and the current

timeslot is free according to *usedSlots* then we claim this timeslot by sending a control packet, *wait* is assigned a new bounded integer by the backoff strategy, *s* is set to the current timeslot index, and *status* is changed to active (line 10-13).

Active node's perspective:
If the current timeslot belongs to this nodes super frame, the frame with the same index as the *s* then proceed to line 9 (line 8). If *wait* is greater than zero and no transmission was detected on the previous timeslot then *wait* is decremented (line 9). However if *wait* is zero and current timeslot is free according to *usedSlots* then a control packet is sent and *wait* assigned a new random bounded integer by the backoff strategy (line 10-12).

Active and passive node's perspective:
The entries mapped to current timeslot index in *usedSlots* and *rack* is expired by having the value FALSE and EMPTY assigned (line 14-15)

4.3.2 Algorithm 1: Upon SFD

SFD is a bit sequence in a transmission which marks the beginning of a frame, not to be confused with a TDMA frame. Upon SFD is executed after the hardware encounter an SFD sequence in a transmission (line 16).
If the SFD originates from a neighbour transmitting
If node is receiving a transmission from a neighbour, which can be implemented by a boolean which is toggled when the node itself starts or stops transmitting, then proceed execution of line 18 (line 17). Mark that the current timeslot in *usedSlots* is in use and save the current time in $pktRecTime$, which will later be used in upon receive(), (line 18). If the $rxAck$ element corresponding mapped to current timeslot was not a control packet then overwrite it with noise (line 19-20).

If the SFD originates from the node's own transmission
Only line 22 is executed which modifies the outgoing transmission buffer which contains the TDMA header timestamp and updates it to the current time (line 22).

4.3.3 Algorithm 2: Upon receive

Receive is executed when a full packet has been received (line 1). The parameters in the receive function are variables from the header subscripted with an s to distinguish them from local variables. $time_s$ originates from line 18 in upon SFD(). time is assigned $pktRecTime$, which holds the timestamp of the SFD event for this packet (line 2). If $time_s$ is greater than time *time* but less then some buffer margin (128 in the pseudocode) then advance the local clock for the time difference and set *time* equal to $time_s$ (line 3-5). However if the time difference is greater than the buffer margin the node's clock is considered misaligned and must advance the clock by the difference, set *time* equal to $time_s$, empty the incorrect $rxAck$ and *usedSlots* maps, and drop the timeslot and become passive (line 6-11). Test if $time_s$ and *time* after

being converted from time to timeslot both matches the same timeslot index (line 12). Update $rxAck$ by adding src_s to the transmitting node's timeslot index (line 13). If any timeslot collisions are detected with the node then drop the timeslot (line 14). For each timeslot in $rxAck_s$ that is non-empty mark corresponding timeslot in $usedSlots$ as taken (line 15-16). The last two lines allows $usedSlots$ to contain the taken timeslots used by distance two neighbours.

Algorithm 1: Shared variables and Controller component

◁ : Categorises the different ways to reach dropTimeslot()
= : Pointer assignment
src : Unique identifier of the node
rxAck : Mapping of timeslot indexes to node *src* used to acknowledge that a packet has been received from a neighbour and which timeslot it used
usedSlots : Mapping of timeslot indexes to if they are in use or free
s : Contains the nodes timeslot index if its status is active
status : A node can either be active or passive
wait : Integer used as a countdown. Utilised in the backoff strategy
pktRecTime: Contains a timestamp for each received transmission

```
1  object Ack
2      src
3      bool cpktOrigin ← FALSE
4  upon alarm()
5      component Alarm:setNextAlarm(getTime())
6      if s() == s then
7          component Radio:transmit( getTime(), rxAck, FALSE, src,
           PAYLOAD)
8      else if status == PASSIVE or mod(currentFrame, FRAME)== s
       then
9          if wait > 0 and unused( s()-1 ) then wait ← max(0,wait-1)
10         else if unused( s() ) and wait <= 0 then
11             component Radio:transmit( getTime(), rxAck, TRUE, src,
               NOPAYLOAD)
12             backOff()
13             if status == PASSIVE then (status, s) ←( ACTIVE, s() )
14     usedSlots[ s() ] ← FALSE
15     rxAck[ s() ] ← EMPTY
16 upon SFD()
17     if receiving from neighbour then
18         (usedSlots[ s() ], pktRecTime) ← ( TRUE, getTime() )
19         ack = rxAck[ s() ]
20         if ack.cpktOrigin == FALSE then rxAck[ s() ] ← NOISE
21         if s == s() then dropTimeslot()                    ▷ Stolen
22     else component Radio:modify( getTime() )
```

Algorithm 2: Receiver component

```
1  upon receive(src_s, s_s, rxAck_s, isCPKT_s, time_s)
2      time ← pktRecTime
3      if (time_s − time) < 128 then
4          call component Alarm:advanceClock( time_s − time )
5          time ← time_s
6      else if (time_s − time) >= 128 then
7          call component Alarm:advanceClock( time_s − time )
8          time ← time_s
9          setall( rxAck, Ack(EMPTY,FALSE) )
10         setall( usedSlots, FALSE )
11         dropTimeslot()                                  ▷ Time advance
12     if timeslot(time_s) == timeslot(time) then
13         updateRxAck(isCPKT_s, src_s, timeslot( time), s_s)
14         neighbourProblems(rxAck_s, isCPKT_s, s_s)
15         for (index, value)← rxAck_s do
16             if value.src! = EMPTY then usedSlots[index] ← TRUE
```

Algorithm 3: helper functions

```
1  function transmit(time, rxAck, isCPKT, senderSrc, payload)
       // The ack.cpktOrigin in rxAck are not transmitted
2      call component CC2420Transmit(payload, header(senderSrc, time, isCPKT,
       rxAck))
3      setall(rxAck, empty, FALSE)
4  function unused(slot x)
5      return usedSlots[x] == FALSE
6  function timeslot(time)
7      return (int) time/TIMESLOT_DURATION
8  function backOff()
9      int tmp ← getRandom( threeDelta)
10     wait ← waitAdd+tmp
11     waitAdd ← threeDelta-r
12 function dropTimeslot
13     (s, status) ← (EMPTY, PASSIVE)
14     backOff()
15 function updateRxAck(isCPKT, senderSrc, slot, s_s)
16     if isCPKT == FALSE then
17         ack = rxAck[ slot ]
18         if ack.cpktOrigin == FALSE then
19             ack.src ← senderSrc
20     else
21         ack = rxAck[ s_s ]
22         (ack.src, ack.cpktOrigin ) ← (senderSrc, TRUE )
23 function neighbourProblems(rxAck_s, isCPKT_s, s_s)
24     if status == ACTIVE then
25         if rxAck_s[s] != EMPTY and rxAck_s[s] != NOISE and rxAck_s[s] != src
           then dropTimeslot()                                  ▷ Interference
26         else if isCPKT_s== FALSE and rxAck_s[s] != src and rxAck_s[s] !=
           NOISE then dropTimeslot()                  ▷ Missed acknowledgement
27         else if s_s == s then dropTimeslot()                        ▷ Stolen
```

4.4 TDMA with link reliability: Link reliability design

TDMAPST's line 26, in algorithm 3 handles missing acknowledgements. If line 26 could be ignored nodes would less frequently turn passive. However if an active node would seldomly receive acknowledgements from a specific neighbour, there could be interference just during the timeslot itself and then the best action is to switch to another timeslot or the link itself could be poor and the best call would be to ignore it.

The solution should ignore some missing acknowledgements but still be flexible to change to new timeslots. Assuming the case of 80% uniform packet success rate across the network, the chance of receiving an acknowledgement is 64% because it needs to be successfully transmitted to a neighbour and successfully received back. We expect three types of packet receptions events to occur on a link, to receive a packet containing an acknowledgement, packet without acknowledgement, or the absence of a packet. By sampling these events we can expect from the sample size of the most recent events, 80% of them to be received and 64% of those 80% to contain acknowledgements. Only links which have received 80% of the packets are considered and if less than 50%, because 64% is the expected value and some margin is helpful, contains an acknowledgement the timeslot is dropped.

4.5 TDMAwLR: Detailed algorithm description

In its essence TDMA with link reliability is the same as the TDMAPST algorithm with the modification that the responsibility of handling missed acknowledgements has been moved from function neighbourProblems, see algorithm 3 line 26, to the new function linkProblem. The following subsections will go through only the lines related to link reliability as how the TDMA protocol is explained in section 4.3. The lines in the pseudocode, 4 5 6 7, marked with "*" are related to the implementation of link reliability.

4.5.1 Algorithm 4: Upon alarm

If the link does not fulfill the requirements then drop the timeslot (line 9). Update the link corresponding to the current timeslot with a preemptive miss, but if a packet is received the miss will be changed, (line 10).

4.5.2 Algorithm 5: Upon receive

NeighbourProblems has been modified, the case which handles the missing acknowledgement has been changed and moved over to the function linkProblem (line 14).

If the packet is not a control packet then update the link mapped to the timeslot the packet belonged to (line 15).

4.5.3 Algorithm 7: Link reliability functions

A link object imitate a circular buffer that overwrites the oldest entry after reaching maximum capacity if used by the functions setMiss, setRec, and setAck. The getRecCount function returns the number of received packets which includes acknowledged packets. getAckCount and getMissCount returns number of acknowledgements and number of absent packets respectively. Function reset does what the name implies *i.e.* resets the link. LinkProblem decides if the node should drop the link. If enough packets has been received and the link object is filled to its maximum capacity then check the ratio of acknowledged packets with total received packets, if it falls below the percentage threshold then drop the timeslot (line 32-35). If not enough packets has been received the link is ignored. This prevents a neighbour with a high packet loss rate to force others out of their timeslots.

Algorithm 4: Shared variables and Controller component

*	: Marks that the function or line is different from the original
◁	: Categorises the different ways to reach dropTimeslot()
=	: Pointer assignment
src	: Unique identifier of the node
rxAck	: Mapping of timeslot indexes to node *src* used to acknowledge that a packet has been received from a neighbour and which timeslot it used
links	: Mapping of timeslot indexes to link objects which is used to evaluate links
usedSlots	: Mapping of timeslot indexes to if they are in use or free
s	: Contains the nodes timeslot index if its status is active
status	: ACTIVE or PASSIVE
wait	: Integer used as a countdown. Utilised in the backoff strategy
pktRecTime	: Contains a timestamp for each received transmission

```
1  object Ack
2      src
3      bool cpktOrigin ← FALSE
4  * object Link
5      * int array[SAMPLESIZE]
6      * int count
7  upon alarm()
8      component Alarm:setNextAlarm(getTime())
9      * linkProblem( s() )
10     * setMiss( s() )
11     if s() == s then
12         component Radio:transmit( getTime(), rxAck, FALSE, src, PAYLOAD)
13     else if status == PASSIVE or mod(currentFrame, FRAME)== s then
14         if wait > 0 and unused( s()-1 ) then wait ← max(0,wait-1)
15         else if unused( s() ) and wait <= 0 then
16             component Radio:transmit( getTime(), rxAck, TRUE, src,
               NOPAYLOAD)
17             backOff()
18             if status == PASSIVE then (status, s) ←( ACTIVE, s() )
19     usedSlots[ s() ] ← FALSE
20     if rxAck[ s() ].cpktOrigin == FALSE or status == PASSIVE then
21         rxAck[ s() ] ← EMPTY
22 upon SFD()
23     if receiving from neighbour then
24         (usedSlots[ s() ], pktRecTime) ← ( TRUE, getTime() )
25         ack = rxAck[ s() ]
26         if ack.cpktOrigin == FALSE then rxAck[ s() ] ← NOISE
27         if s == s() then * dropTimeslot()                          ▷ Stolen
28     else component Radio:modify( getTime() )
```

Algorithm 5: Receiver component

```
1  *upon receive(src_s, s_s, rxAck_s, isCPKT_s, time_s)
2      time ← pktRecTime
3      if (time_s − time) < 128 then
4          call component Alarm:advanceClock( time_s − time_i )
5          time ← time_s
6      else if (time_s − time) >= 128 then
7          call component Alarm:advanceClock( time_s − time )
8          time ← time_s
9          setall( rxAck, Ack(EMPTY, FALSE) )
10         setall( usedSlots, FALSE )
11         * dropTimeslot()                                  ▷ Time advance
12     if timeslot(time_s) == timeslot(time) then
13         updateRxAck(isCPKT_s, src_s, timeslot( time), s_s)
14         * neighbourProblems(rxAck_s, isCPKT_s, s_s)
15         * if isCPKT == FALSE then updateLink( timeslot(time), rxAck)
16         for (index, value)← rxAck_s do
17             if value.src != EMPTY then usedSlots[index] ← TRUE
```

Algorithm 6: helper functions

```
1  function transmit(time, rxAck, isCPKT, senderSrc, payload)
       // The ack.cpktOrigin in rxAck are not transmitted
2      call component CC2420Transmit(payload, header(senderSrc, time, isCPKT, rxAck))
3      setall(rxAck, EMPTY, FALSE)
4  function unused(slot x)
5      return usedSlots[x] == FALSE
6  function timeslot(time)
7      return (int) time/TIMESLOT_DURATION
8  function backOff()
9      int tmp ← getRandom( threeDelta)
10     wait ← waitAdd+tmp
11     waitAdd ← threeDelta-r
12 * function dropTimeslot
13     (s, status) ← (EMPTY, PASSIVE)
14     backOff()
15     * for i=0;i<FRAME;++i do
16         * reset(i)
17 function updateRxAck(isCPKT, senderSrc, slot, s_s)
18     if isCPKT == FALSE then
19         ack = rxAck[ slot ]
20         if ack.cpktOrigin == FALSE then
21             ack.src ← senderSrc
22     else
23         ack = rxAck[ s_s ]
24         (ack.src, ack.cpktOrigin ) ← (senderSrc, TRUE )
25 *function neighbourProblems(rxAck_s, isCPKT_s, s_s)
26     if status == ACTIVE then
27         if rxAck_s[s] != EMPTY and rxAck_s[s] != NOISE and rxAck_s[s] != src then
           dropTimeslot()                                              ▷ Interference
28         else if s_s == s then dropTimeslot()                        ▷ Stolen
```

Algorithm 7: linkreliability helper functions

```
* function setMiss(idx)
    * l = links[ idx ]
    * l.array[ l.count++ % SAMPLESIZE ] ← MISS
* function setRec(idx)
    * l = links[ idx ]
    * l.array[ (l.count-1)%SAMPLESIZE] ← REC
* function setAck(idx)
    * l = links[ idx ]
    * l.array[ (l.count-1)%SAMPLESIZE] ← ACK
*function getRecCount(idx)
    * l = links[idx]
    * result ← 0
    * for i=0; i<SAMPLESIZE and i<l.count; ++i do
        * if l.array[ i ] == REC or l.array[ i ] == ACK then ++result
    * return result
* function getAckCount(idx)
    * l = links[idx]
    * result ← 0
    * for i=0; i<SAMPLESIZE and i<l.count; ++i do
        * if l.array[ i ] == ACK then ++result
    * return result
* function getMissCount(idx)
    * l = links[idx]
    * result ← 0
    * for i=0; i<SAMPLESIZE and i<l.count; ++i do
        * if l.array[ i ] == MISS then ++result
    * return result
* function reset(idx)
    * l = links[ idx ]
    * l.count ← 0
* function linkProblem(idx)
    * l = links[ idx ]
    * if getRecCount(idx) > THRESHOLD and l.count >= SAMPLESIZE then
        * if getAckCount(idx) / getRecCount(idx) < PERCENTAGE then
            * if status == ACTIVE then dropTimeslot()        ▷ Missed acknowledgement
* function updateLink(slot, rxAck_s)
    * if rxAck[s] == src then setAck( slot )
    * else setRec( slot)
```

4.6 Self-stabilisation

A system must satisfy the convergence and closure property to be called self-stabilising. Subsequent subchapters define a safe configuration for TDMAPST and explains how the convergence and closure properties requirements are met [1].

4.6.1 Safe configuration

All nodes are active with their own unique timeslot index among their distance two neighbours. This is possible because nodes track which timeslots are used by their distance two neighbours in their *usedSlots* map which reflect correctly which timeslots its neighbours are using. Each node's clock deviate less than some error margin compared to its neighbours.

4.6.2 Legal execution

An active node must transmit a data packet during its timeslot, containing $rxAck$, allowing its neighbours *usedSlots* map to reflect correctly which timeslots are in use and keep neighbouring clocks synchronised.

4.6.3 Convergence

To ensure that TDMAPST provide collision free transmission the convergence phase must be capable of handling six cases to reach a safe configuration, unsynchronised clocks, passive nodes, *usedSlots* containing false timeslot claims which we call false positve, *usedSlots* missing valid timeslot claims which we call false negative, direct interference, and hidden terminals. How each case is resolved can be read in the next paragraphs.

Clock synchronisation
In a connected system with misaligned clocks atleast one node has the highest value which is considered the correct time. Other nodes will eventually receive the correct clock value and align themselves after that value, set their status to passive and perform the random backoff strategy to become active. After aligning themselves with the correct value once they will never have to do it again as no nodes exists with a higher clock value.

Passive nodes
To activate passive nodes. All nodes have a counter variable called *wait* containing a uniform random positive integer. For each free timeslot that passes the *wait* is decremented by one until reaching zero where it will claim the next free timeslot in the *usedSlots* map by sending a control packet during that timeslot notifying its neighbours, set status to active, and then set *wait* to some new bounded random

integer value. Note that the control packet can collide with another packet if the system is not in a safe configuration.

False positive
False positive claims in the *usedSlots* map do not have a node to update the claim so they will eventually expire, they do not affect the countdown of the variable *wait* because it only decrements if the previous timeslot was free. False positive claims can however delay nodes striving to become active because the node can only claim the next free timeslot according to *usedSlots* and if they contain false positive we have to wait until they expire or a free timeslot is encountered.

False negative
False negative claims on the *usedSlots* mapping have two ways to be resolved, one way to be relabeled as another problem, or repeat itself. The resolutions are either if the node owning the timeslot or a third node corrects the false negative in *usedSlots* by transmitting before the incorrect nodes transmits anything. If the node tries to become active by sending a control packet the problem is relabeled as a direct interference. If the node is already active and sents a control packet the node owning the timeslot might become passive or the resolution is reapplied on the same scenario.

Direct interference
Let two active nodes, $node_1$ and $node_2$, be within transmission range to each other and both owning the same timeslot. They will not be able to send data packets to each other because of the interference they create. They resolve this by sending control packets similarly to passive nodes trying to claim a timeslot but with a modified countdown mechanism. Control packets from active nodes are only sent on their super frames, so they only compete on unused timeslots with passive nodes and other active nodes with the same timeslot which should be few. Eventually a control packet from one of the two nodes will reach the other, informing it of the conflict which results in that node dropping the timeslot and becoming passive.

Hidden terminals
Let two active nodes, $node_1$ and $node_2$, be out of transmission range from each other and own the same timeslot. Let a third node, $node_{inter}$, be in range of both of them. When $node_1$ and $node_2$ transmit during their timeslot $node_{inter}$ cannot receive the packet due to interference. With some probability $node_1$ will succeed in sending a control packet on a free timeslot on its super frame to $node_{inter}$ informing the node of its usual timeslot which $node_{inter}$ can propagate to $node_2$ which will drop its timeslot thus resolving the issue.

4.7 TDMA packet header

The TDMAPST header has been embedded within the IEEE 802.15.4 header specification, for low-rate wireless personal area network (PAN), as it is already in place in TinyOS. The complete header can be seen in table 4.1 where TDMAPST specific bytes has been highlighted with a blue background colour.

Table 4.1: The table shows the TDMAPST header format that comes with all transmissions. FRAME in this example is 27.

offset bytes	0	1	2	3
0	length	fcf		dsn
4	destpan		dest	
8	src		network	s
12	competeAfter	notif	rxAck[FRAME]	
16	rxAck[FRAME]			
20	rxAck[FRAME]			
24	rxAck[FRAME]			
28	rxAck[FRAME]			
32	rxAck[FRAME]			
36	rxAck[FRAME]			
40	rxAck[FRAME]	comDelay		
44	comDelay	seq		
48	seq	isCPKT	timestamp	
52	timestamp		type	

Table 4.2: TDMA packet header content description

Field name	Description
length	Length of header and payload in bytes subtracted by one.
fcf	Frame Control Field, defined in the 808.15.4 specs.
dsn	Data Sequence Number, a number incremented for each packet sent. Used in filtering out duplicate packets.
destpan	The destination PAN ID, allows the network to sit beside another TinyOS network and not interfere.
dest	The Destination address of this packet.
src	Unique identifier of the transmitting node.
network	The TinyOS network ID, for interoperability with other types of 802.15.4 networks.
s	The transmitting nodes timeslot index.
competeAfter	Deprecated.
notif	Deprecated.
rxAck	Acknowledgement array of size FRAME containing node src'es which packets has been received.
isCPKT	If the packet is a control packet or a data packet.
comDelay	Deprecated.
seq	Deprecated.
timestamp	Generated when the message is sent, used to synchronise clocks.
type	TinyOS AM type is used to differentiate between different type of packets.

5

Results

Our goal is to increase the robustness in TDMAPST (called the original in this chapter), which we do by extending it with the feature link reliability. We evaluate two settings of link reliability, with sample size ten and twenty, by comparing them together with the original. This chapter discusses the settings available for the experiments, evaluation of the implementations in the Cooja simulator that covers convergence time and robustness. Convergence time is measured in frames and a special metric is used for robustness, total number of active nodes, particular to our protocol. All graphs on robustness are accompanied by bar charts presenting the total number of dropTimeslot function calls categorised by:

1. Missed acknowledgements - When a node in the original TDMA protocol receives a packet and its acknowledgement is not included in $rxAck$ it becomes passive respectively in link reliability when the function linkProblem results in the execution of dropTimeslot.
2. Stolen - When an active node is notified directly by a neighbour that they share the same timeslot.
3. Time advance - When a node receives a higher clock value than its own time and the time difference is greater than some predefined margin.
4. Interference - When a node is notified by a control packet that its timeslot is interfering with another node.

All graphs and bar charts are made out of ten simulations. In the graphs the blue line represent the average, the green outline represent the best data point out of the ten experiments and the red outline represent the worst data point. The convergence graph also show the median.

A full explanation of the dropTimeslot categories is found in the pseudocode in chapter 4, all categories are marked with a $\triangleleft$ in the pseudocode.

5.1 Setup

In this section, we discuss the configuration options available for the experiments. All settings are given an attribute, *e.g.*, environment, an experiment can not have two settings with the same attribute.

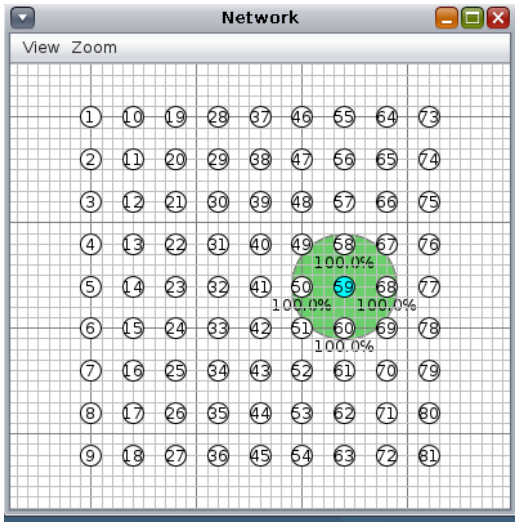

Figure 5.1: Grid layout, with 100% transmit success rate to neighbour. The transmit success rate can be modified

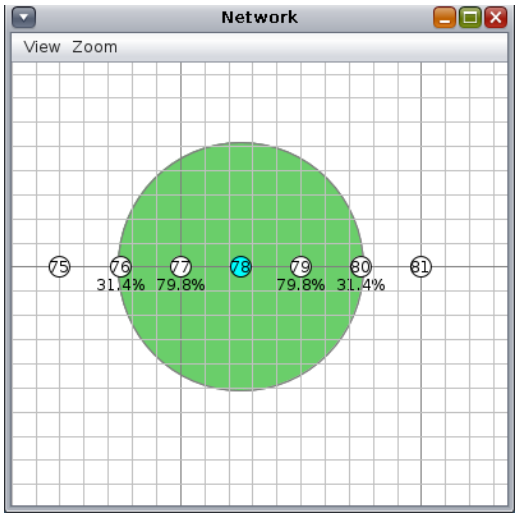

Figure 5.2: Line layout, with approximately 80% transmit success rate to close neighbours and 31% to the rest of the neighbours

- Cooja (Environment) - The simulation tool Cooja is the Contiki's network simulator. Cooja allows telosB motes to be simulated.
- Indriya testbed (Environment) - Indriya testbed is a three-dimensional WSN deployed across three floors at the National University of Singapore to facilitate research in communication protocols and other fields.
- Original (Code) - Original is the implementation we call TDMAPST based on Petig *et. al's* TDMA protocol.
- Link ten (Code) - Link ten is TDMAPST extended with link reliability configured with a sample size of ten.
- Link twenty (Code) - Link twenty is TDMAPST extended with link reliability configured with a sample size of twenty.
- Grid (Layout) - When nodes are placed in a grid format with maximum four distance one neighbours. Depicted in figure 5.1.
- Line (Layout) - When nodes are placed in a line with maximum four distance one neighbours. Depicted in figure 5.2.
- Indriya (Layout) - The node arrangement Indriya testbed uses.
- Transmit success rate (Tsr) - On Cooja the success rate can be set uniformly across the network and the experiments uses 100%, 80% and a combination of 80% and 31% transmit success rate. However on Indriya this setting is dynamic and uncontrollable.

5.2 Cooja experiments

Cooja [21] is a network simulator for Contiki and allows networks to be simulated. This section contains all the experiments run on Cooja. The first subsection will cover convergence time, the original versus the extended versions. The following subsections cover robustness and how the implementations handle packet loss.

5.2.1 Convergence time

Convergence time is the time taken for an arbitrary configuration to reach a safe configuration. Because there is no practical way in Cooja to verify a system achieved a safe configuration we instead redefine convergence time to the time from system start until all nodes are simultaneously active. We expect link ten and twenty to perform either better or worse than original but not both.

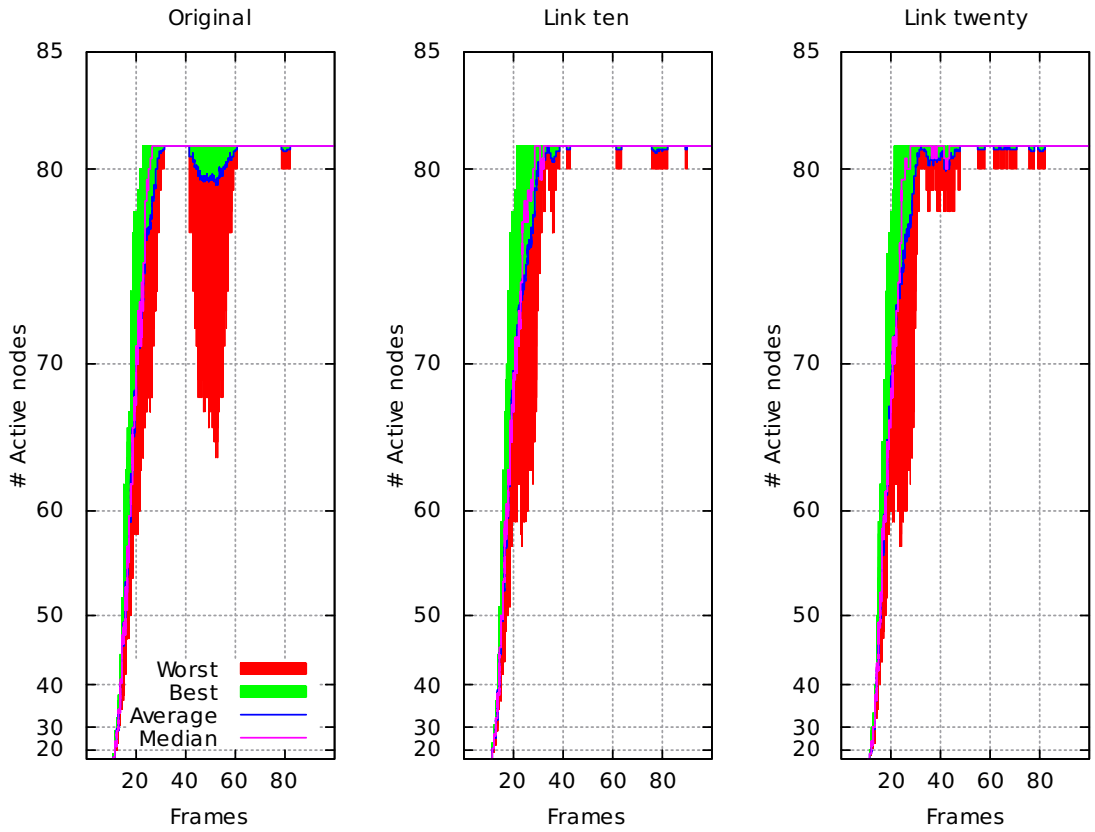

Figure 5.3: Gridlayout, 100% transmit success rate, thirteen distance two neighbours and a total of 81 nodes in the system. The last node is started within 21 timeslots of when the first node was started. By looking at the median, pink line, all three graphs have similar convergence time.

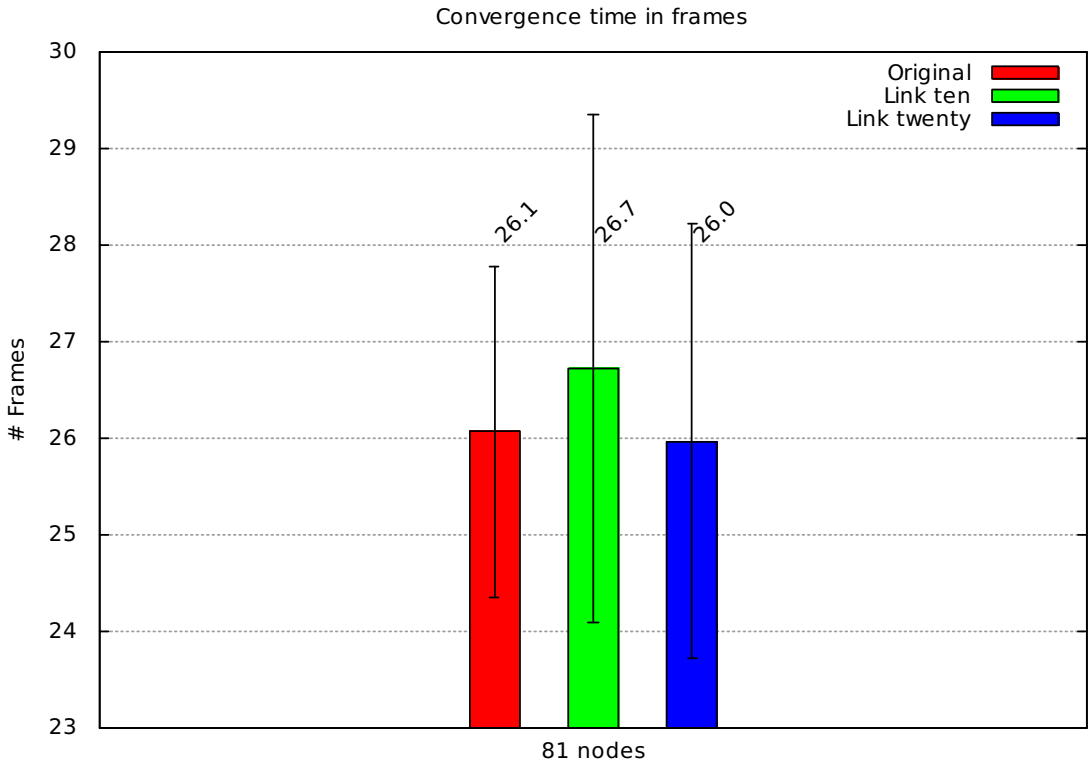

Figure 5.4: The bar chart shows how many frames required for all nodes to be simultaneously active for the first time from system start. Link ten and twenty is either both better than original or worse. Because ten is worse and twenty is better it is likely that the data is insufficient.

Figure 5.3 shows that original, Link ten and Link twenty have similar convergence time. Original have a red spike which is caused by a bug and is a distraction when we examine the convergence time. The bug gives rise to a faulty clock value which can be greater than the highest clock in the system. What follows is 80 nodes re-aligning themselves and claiming new timeslots which occurs between frames 40 to 60 in figure 5.3 and is limited to to a single experiment and therefore only the average is affected and not the median which is still at 81 in the y-axis. Because the figure in 5.4 shows Link ten's average performing worse than original but Link twenty performing better we draw the conclusion that the data is insufficient to base a winner on between original vs extended and are satisfied that the convergence times are similar.

5.2.2 Gridlayout 100% transmit success rate

This test is performed to give us a baseline of how the implementations perform in an environment with no clock skews, ambient noise and hardware failures *i.e.* fault free environment. The experiments are expected to be in a safe configuration and stay there due to the closure property and all achieve 81 in active nodes and have zero dropTimeslot calls.

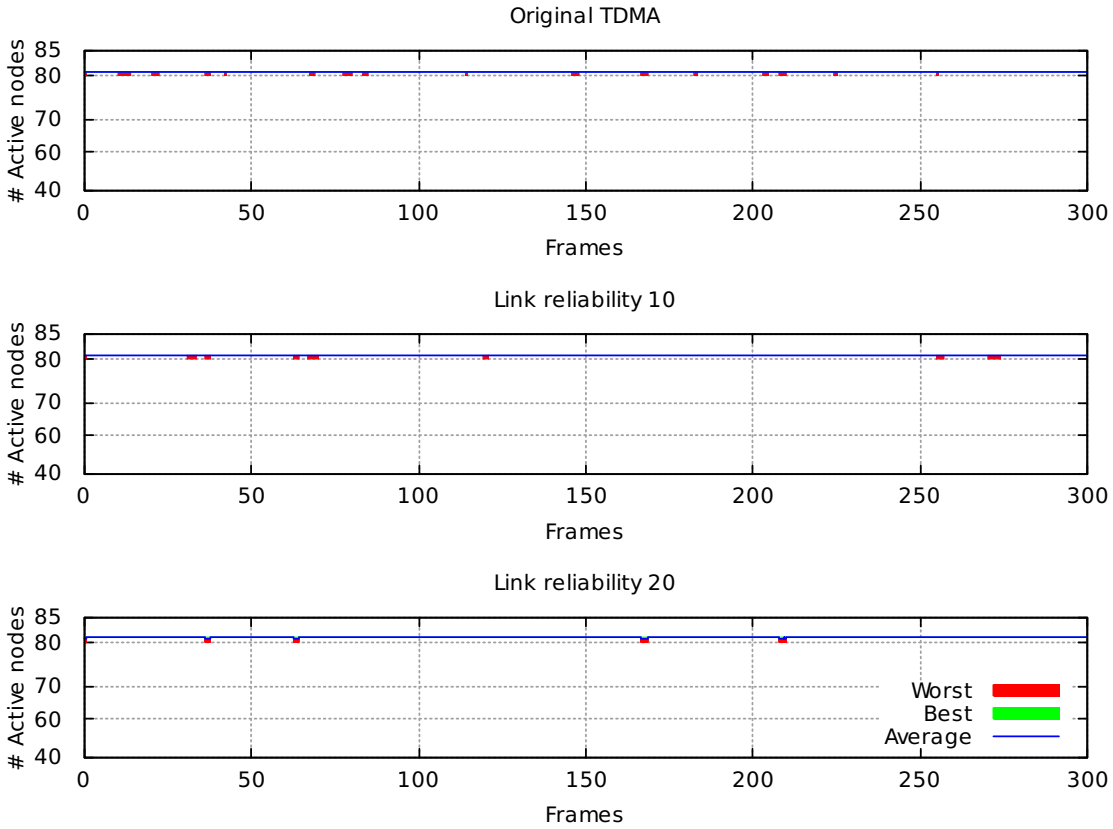

Figure 5.5: The experiment settings are 81 nodes in a grid layout with 100% packet success rate. The graphs show number of active nodes over time. A higher value is better. The lack of hardware and environment disturbances are the reason why the graphs present similar results. We expect 81 active nodes in a fault free environment at all time, there are a few losses as seen by the red marks which indicate that the closure property is not fully reliable.

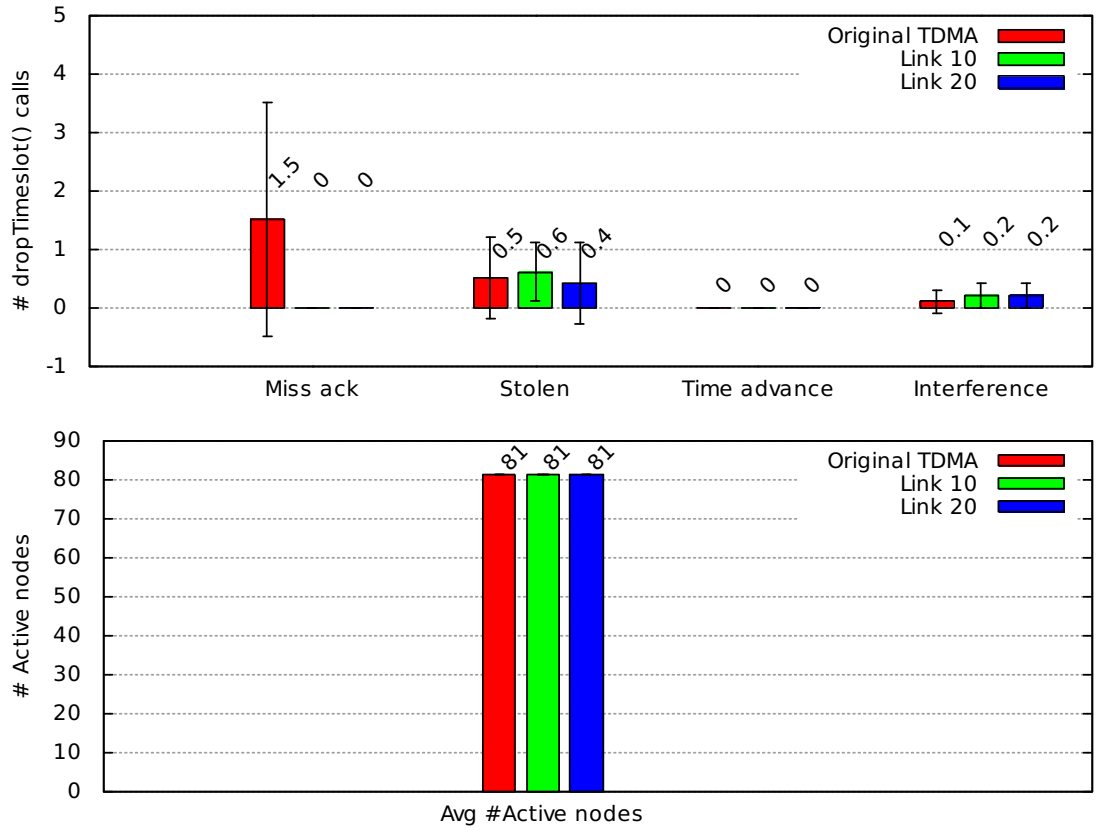

Figure 5.6: The top graph's columns represent total number of dropTimeslot calls. A low number of calls is better. The bottom graph's columns represent the average number of active nodes. A higher number is better with 81 being the maximum. The lack of hardware and environment disturbances are the reason why the graphs present similar results. We expect Link ten and twenty to perform better than the original but as the original achieved the highest score of 81 link ten and twenty have nothing to improve on.

Note that in figure 5.5 the red marks come from ten simulations and the average of red marks is only two per simulation. Though there exist errors it is so seldom that the average number of active nodes is not affected. Because the system is most likely in a non-conflicting configuration in the beginning of the graph a reasonable explanation would be that control packets are colliding with active nodes timeslots which means that *usedSlots* is not fully accurate. To summarise, there are problems with the implementation and therefore the closure property is not working however the problems happen so seldom that all the implementations still achieve the highest score 81/81.

5.2.3 Gridlayout 80% transmit success rate

This test was performed to see how the implementations cope with packet loss. Packet loss eventually leads to a missed acknowledgement which results in a node dropping its timeslot and turning passive. The more nodes that are passive the greater the chance of timeslot collisions and hidden terminals. We expect in the graph of active nodes over time, in figure 5.7, that the height between the best and the worst outline will decrease between the graphs: original, link ten, and link twenty in that order.

When it comes to total number of dropTimeslot in the missed acknowledgement category, depicted in figure 5.8, we expect the original to have the highest column followed by link ten and twenty and the same is expected for the stolen and interference category as well because of the transient nodes created by packet loss. The average number of active nodes is reversely correlated with total number of dropTimeslot events therefore we predict that link twenty should have the highest average of active nodes highest followed by link ten and then the original.

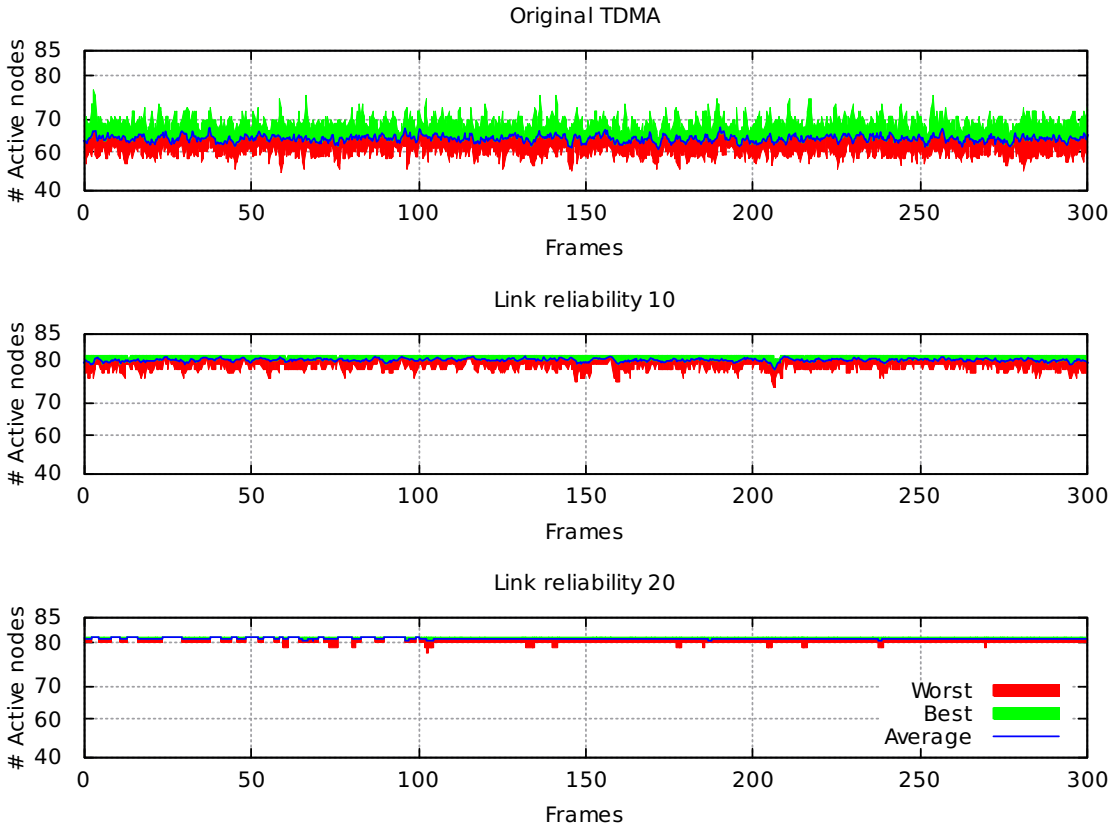

Figure 5.7: The graphs show number of active nodes over time. A higher value is better. The transmit success rate is 80%. The link reliability's sample size correlates with its robustness. Note that Link ten and twenty outperforms the original by at least 15 units of active nodes on average. The green and red outline in original are not only further from the average line but also more spiky compared to link ten and twenty. Predictable outlines closer to the average line is better.

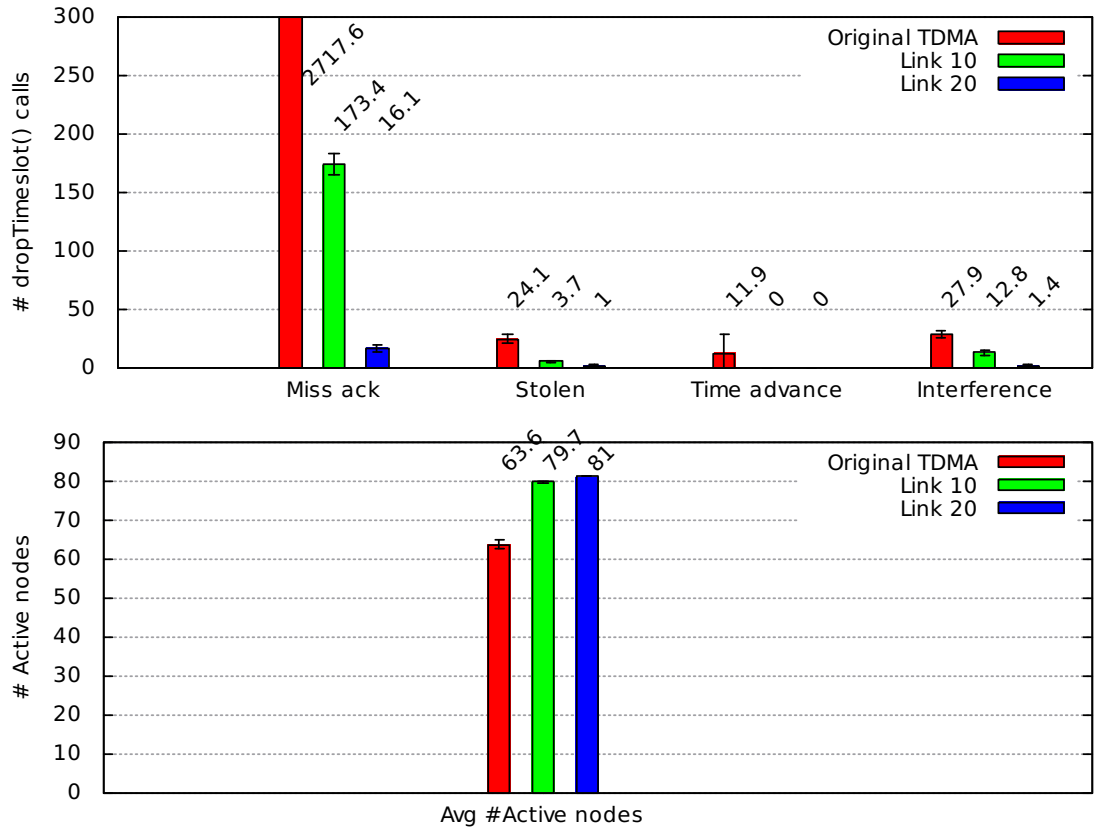

Figure 5.8: The top graph's columns represent total number of dropTimeslot calls. A low number of events is better. The bottom graph's columns represent the average number of active nodes. A higher number is better. The more nodes that are passive increase the probability of timeslot collisions, direct interference, and hidden terminal. The missed acknowledgement category which is majorly caused by missed acknowledgements creates transient nodes which drives up the stolen and interference category. Link ten and twenty is able to suppress missed acknowledgements compared to the original by a factor of fifteen and 160 respectively. The time advance category should be empty from a theoretical point of view.

The graphs and bars support our predictions with the exception of the time advance category as it is non-zero for the original implementation and likely caused by the strain from the high number of dropTimeslot events.

5.2.4 Linelayout 80% and 31% packet success rate

The test is performed to evaluate how the implementations cope with neighbours with different receive ratios. Two of them has 80% packet success rate, and the other two has 31%. The node layout is that of a line, depicted in figure 5.2, because the only way to have different packet success rate was to base that on distance from the transmitting node and a gridlayout did not allow it. We expect the original to perform similar to section 5.2.3 but with an increased number of dropTimeslot calls in missed acknowledgement and because of that we expect more passive nodes which should result in an increase in the categories of stolen and interference. This should results in a lower average of active nodes. Meanwhile we expect link ten and twenty to not be affected by the 30% receive ratio nodes due to link reliability.

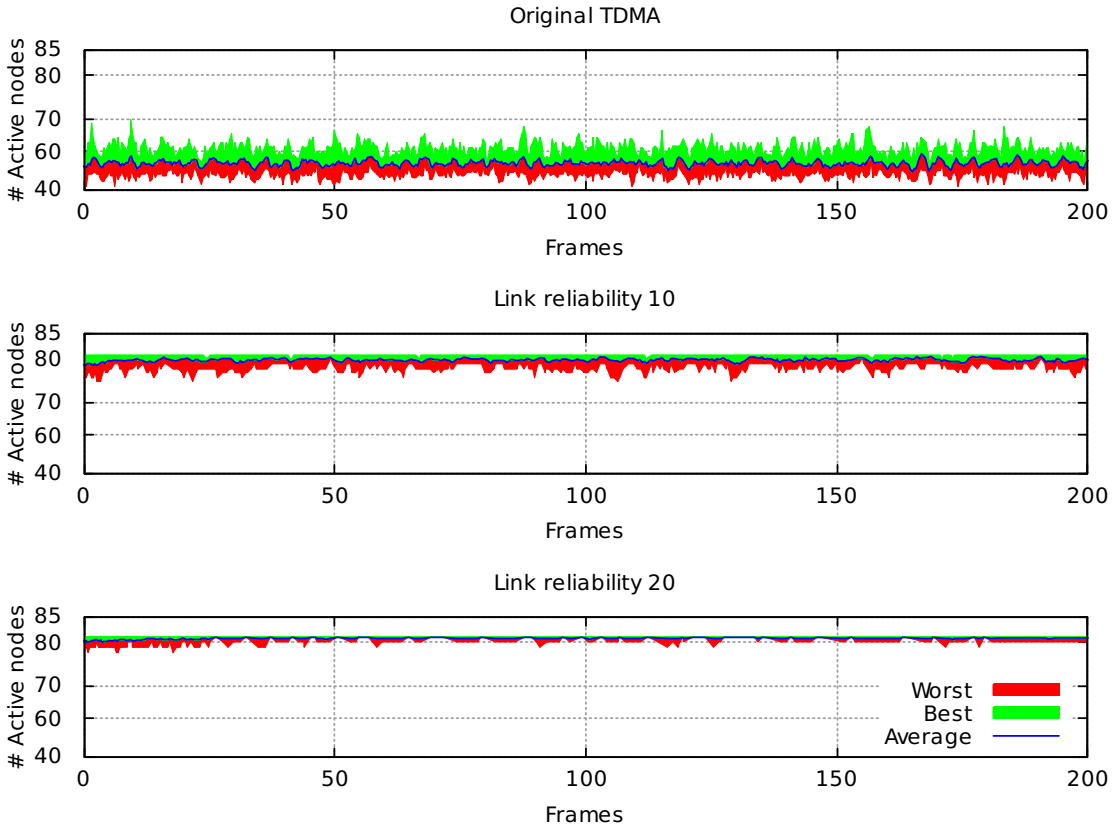

Figure 5.9: The graphs show number of active nodes over time. A higher value is better. The majority of nodes have two neighbours with 80% transmit success rate and two neighbours with 31% transmit success rate. The lowered success rate affects the original more than link ten and twenty. Original, link ten and twenty are 26.5, 2, 1 units from the maximum score respectively.

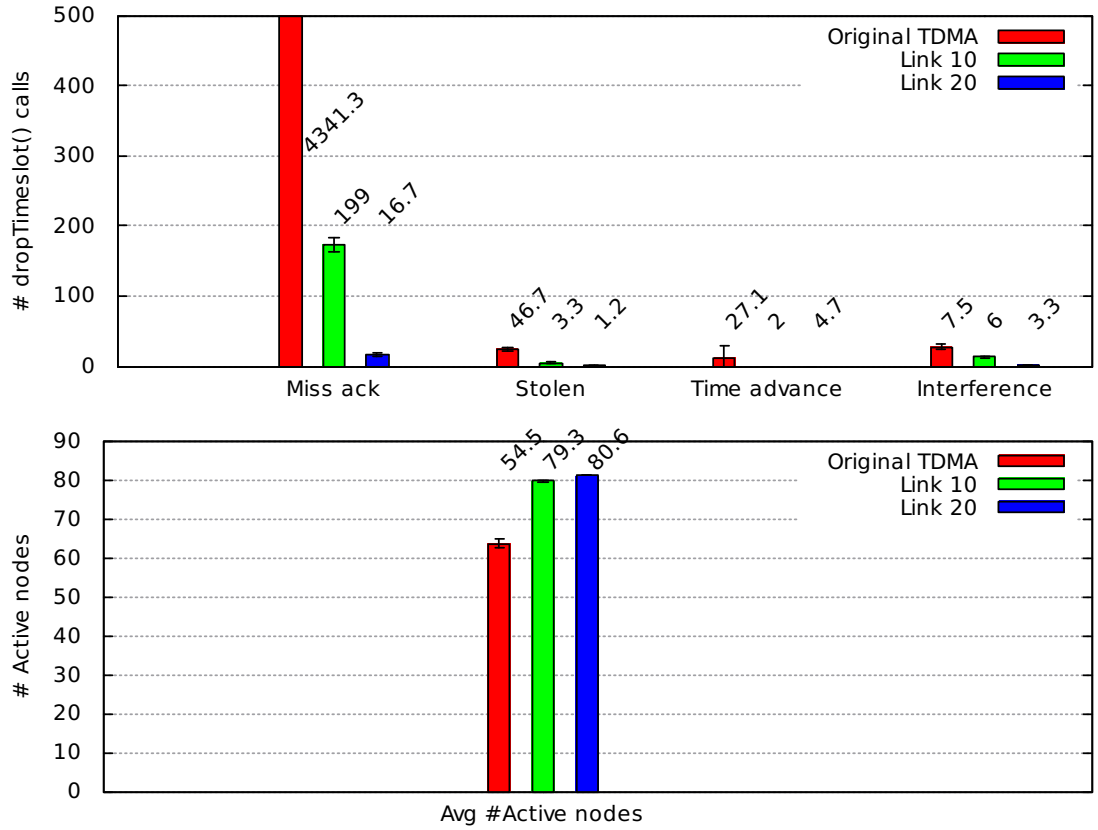

Figure 5.10: The top graph's columns represent total number of dropTimeslot calls. A low number of events is better. The bottom graph's columns represent the average number of active nodes. A higher number is better. Link ten and twenty are more robust than the original and a larger sampling size decreases the probability that a link turns passive. A larger sample size is better.

By comparing with the previous experiment where the system has 80% transmit success rate we notice that the original are more affected by the lowered success, by the nodes with 31% success rate, compared to link ten and twenty. By comparing the figures 5.8 and 5.10 with each other we see that original lost approximately nine more units while link ten and twenty lost less than one unit. This shows how well link reliability can ignore nodes with low packet success rate.

5.3 Experiments on Indriya

Real hardware is used in these experiments and clock skews are very much of reality. There is no guarantee that the graph is connected and ambient noise can exist along with faulty telosB motes. The packet success rate is not uniform and may vary during the experiments. On all experiments in Cooja the distance two neighbours is maximum thirteen and the frame size is 27. However on Indriya a node can have approximately 24 distance two neighbours with a frame size of 27. This means that the ratio between distance two neighbours per timeslot in a frame in Cooja to be approximately two while it varies in Indriya and the worst case scenario is closer to one in Indriya. A lower ratio increases the probability of timeslot collisions and a slower count down of *wait* which results in fewer control packets being sent. The category Stolen is deactivated because the telosB motes steal their own timeslots which does not happen in the simulator. Instead those cases will be handled with control packets by category interference. Stolen has the advantage over interference that it works faster but is a tolerable loss. We expect in graph 5.11 and the bars 5.12 that link ten and twenty always are an improvement of the result of original.

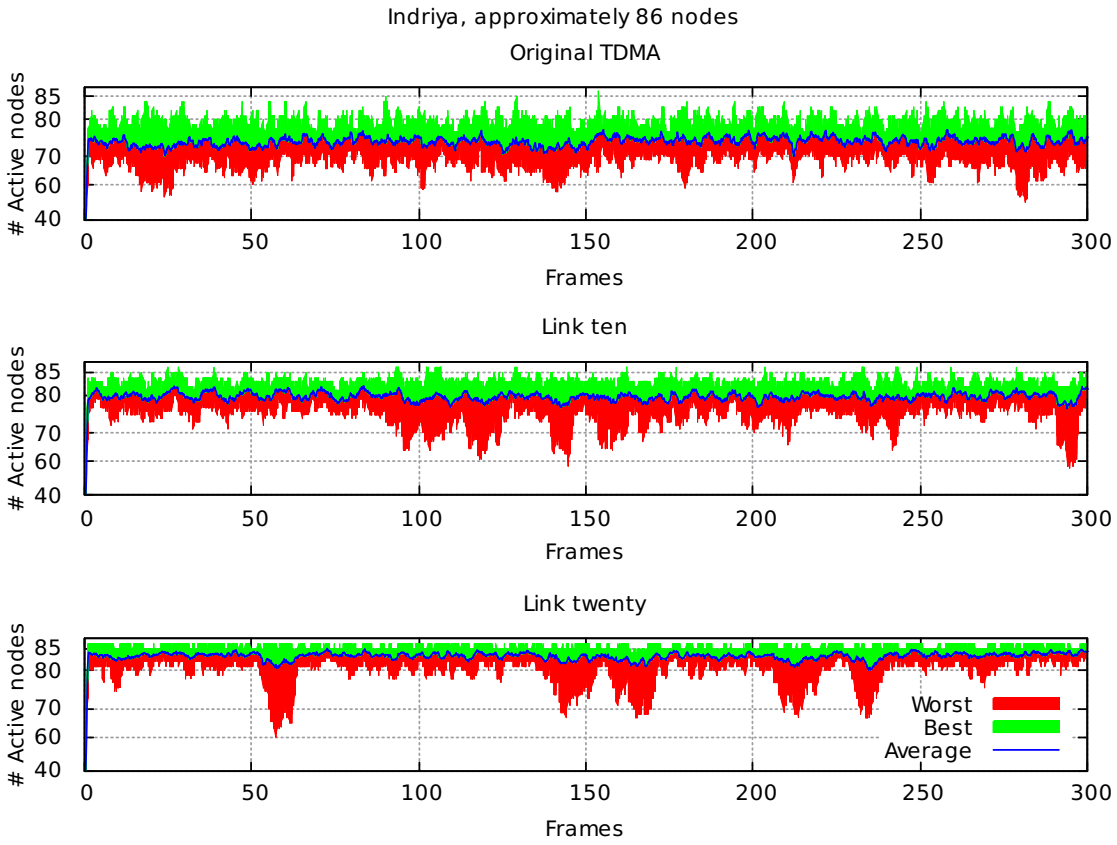

Figure 5.11: The graphs show number of active nodes over time. A higher value is better. The red downward spikes represent unsynchronised clocks synchronising. Link ten and twenty both are improvements compared to the original as the average is increased and the outlines of worst and best are both closer to the average.

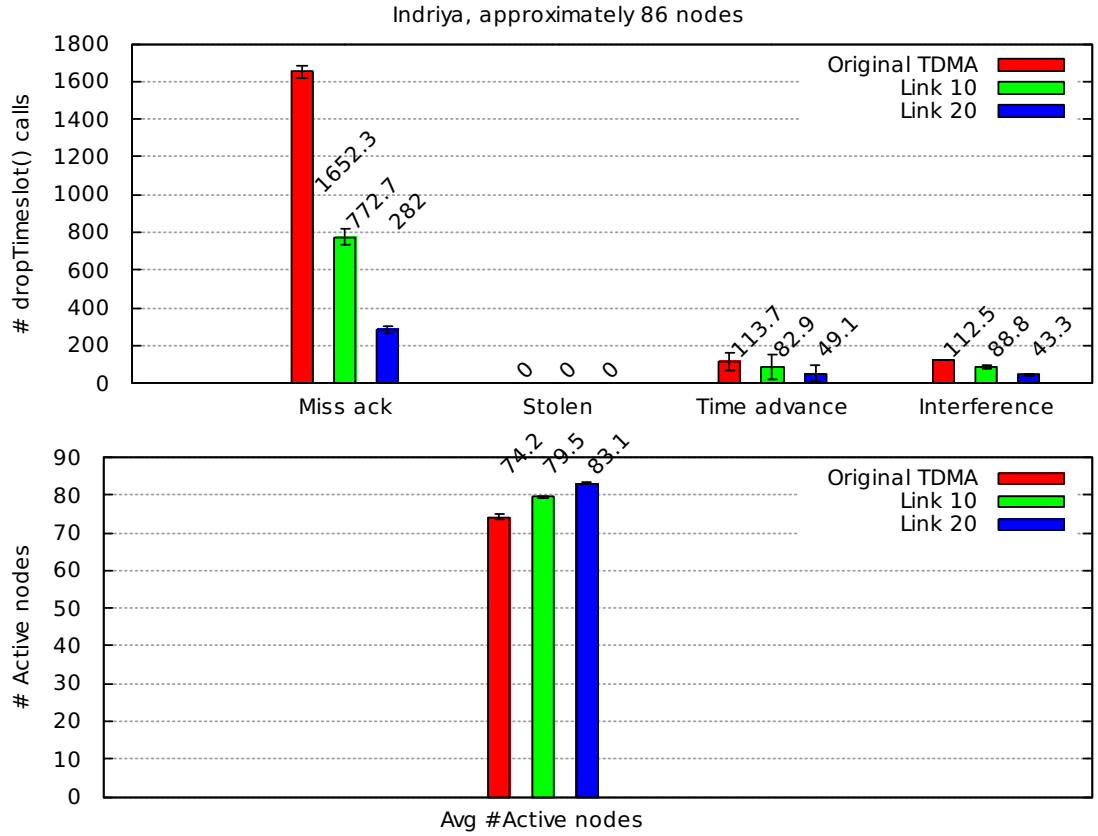

Figure 5.12: The top graph's columns represent events that forces an active node to become passive. A low number of events is better. The bottom graph's columns represent the average number of active nodes. A higher number is better. Link ten and twenty present better results than the original in all categories.

The figure 5.11 shows that link ten and twenty are both better than the original. Compared to the simulations the results from Indriya improve on the original in the same order, link twenty is better than link ten, but not with the same factor. In the figure 5.8 link ten's missed acknowledgements were less than original by a factor of five but in 5.12 the factor is around two.

6

Discussion

Our proposed protocol, which provides bounded communication delays, can facilitate the satisfaction of system safety requirements [22]–[24]. Our experiment show that link reliability can lower the rate at which active nodes become passive, because of omitted packet acknowledgments. Other valid reasons to become passive *e.g.*, breaking hidden terminals has been left unmodified. The experiment demonstrate that the original and the extended versions have similar convergence time.

The graphs in figures, 5.5, 5.7, and 5.9 5.11, support that link reliability successfully increases the robustness and robustness is correlated with sampling size. The figures in 5.6, 5.8, 5.10, and 5.12 support that the average number of active nodes is inversely correlated with total number of dropTimeslot() function calls.

The implementations are tested in simulated environments with (intermittent) faults in which link ten and twenty handle omitted acknowledgement over the original by a factor greater than fifteen and 160 respectively in the experiment with 80% transmit success rate. In the Indriya experiments link ten and twenty handle omitted acknowledgement better than the original by a factor of two and five respectively.

Bibliography

[1] T. Petig, E. M. Schiller, and P. Tsigas, "Self-stabilizing tdma algorithms for wireless ad-hoc networks without external reference", in *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, IEEE, 2014, pp. 87–94.

[2] P. Harrop and R. Das, "Wireless sensor networks 2010-2020", *Networks*, vol. 2010, p. 2020, 2010.

[3] T. Datasheet, *Crossbow inc*, 2013.

[4] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications", in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, 2003, pp. 126–137.

[5] A. Sgora, D. J. Vergados, and D. D. Vergados, "A survey of tdma scheduling schemes in wireless multihop networks", *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 53, 2015.

[6] T. Herman and S. Tixeuil, "A distributed tdma slot assignment algorithm for wireless sensor networks", in *Algorithmic Aspects of Wireless Sensor Networks*, Springer, 2004, pp. 45–58.

[7] R. Fan and N. Lynch, "Gradient clock synchronization", *Distributed Computing*, vol. 18, no. 4, pp. 255–266, 2006.

[8] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener, "Contention-free mac protocols for wireless sensor networks", in *Distributed Computing*, Springer, 2004, pp. 245–259.

[9] T. Herman and S. Tixeuil, "A distributed tdma slot assignment algorithm for wireless sensor networks", in *Algorithmic Aspects of Wireless Sensor Networks*, Springer, 2004, pp. 45–58.

[10] N. Abramson, "The aloha system: Another alternative for computer communications", in *Proceedings of the November 17-19, 1970, fall joint computer conference*, ACM, 1970, pp. 281–285.

[11] H. Wesolowski and K. Wesolowski, *Mobile communication systems*. John Wiley & Sons, Inc., 2001.

[12] A. Swami, Q. Zhao, Y.-W. Hong, and L. Tong, *Wireless Sensor Networks: Signal Processing and Communications*. John Wiley & Sons, 2007.

[13] S. Leffler *et al.*, "Tdma for long distance wireless networks", *White Paper*, 2009.

[14] P. Leone and E. M. Schiller, "Self-stabilizing tdma algorithms for dynamic wireless ad hoc networks", *International Journal of Distributed Sensor Networks*, vol. 2013, 2013.

[15] M. Mustafa, M. Papatriantafilou, E. M. Schiller, A. Tohidi, and P. Tsigas, "Autonomous TDMA alignment for vanets", in *Proceedings of the 76th IEEE Vehicular Technology Conference, VTC Fall 2012, Quebec City, QC, Canada, September 3-6, 2012*, IEEE, 2012, pp. 1–5, ISBN: 978-1-4673-1880-8. DOI: 10.1109/VTCFall.2012.6399373. [Online]. Available: http://dx.doi.org/10.1109/VTCFall.2012.6399373.

[16] L. G. Roberts, "Aloha packet system with and without slots and capture", *ACM SIGCOMM Computer Communication Review*, vol. 5, no. 2, pp. 28–42, 1975.

[17] D. K. Elliott and J. H. Christopher, "Understanding gps: Principles and applications", *Edition Kaplan*, 1996.

[18] M. Mustafa, M. Papatriantafilou, E. M. Schiller, A. Tohidi, and P. Tsigas, "Autonomous tdma alignment for vanets", in *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, IEEE, 2012, pp. 1–5.

[19] S. Dolev, *Self-stabilization*. MIT press, 2000.

[20] E. W. Dijkstra, "Self-stabilization in spite of distributed control", in *Selected writings on computing: A personal perspective*, Springer, 1982, pp. 41–46.

[21] F. Österlind, "A sensor network simulator for the contiki os", *SICS Research Report*, 2006.

[22] O. M. Ponce, E. M. Schiller, and P. Falcone, "Cooperation with disagreement correction in the presence of communication failures", *CoRR*, vol. abs/1408.7035, 2014. [Online]. Available: http://arxiv.org/abs/1408.7035.

[23] A. Casimiro, J. Rufino, R. C. Pinto, E. Vial, E. M. Schiller, O. M. Ponce, and T. Petig, "A kernel-based architecture for safe cooperative vehicular functions", in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, IEEE, 2014, pp. 228–237. DOI: 10.1109/SIES.2014.6871208. [Online]. Available: http://dx.doi.org/10.1109/SIES.2014.6871208.

[24] A. Casimiro, O. M. Ponce, T. Petig, and E. M. Schiller, "Vehicular coordination via a safety kernel in the gulliver test-bed", in *34th International Conference on Distributed Computing Systems Workshops (ICDCS 2014 Workshops), Madrid, Spain, June 30 - July 3, 2014*, IEEE, 2014, pp. 167–176, ISBN: 978-1-4799-4181-0. DOI: 10.1109/ICDCSW.2014.25. [Online]. Available: http://dx.doi.org/10.1109/ICDCSW.2014.25.

A

Appendix 1

A.1 TDMA component interface

Table A.1: An example table showing the structure of a component description

Component: Component name
Description: Component description

functionName	Function description.
functionParam1	Parameter description 1.
functionParam2	Parameter description 2.

Events (Events the component have handlers for.)

eventFunction()	Event description.
eventParam1	Parameter description 1.
eventParam2	Parameter description 2.

Component: TDMA
Description: Provides medium access control to incoming and outgoing messages.

send	Transmits message when the application has access to the medium.
msg	The payload.

Events

receive()	New message received.
msg	The payload.
header	TDMA header information.

Subcomponent: CC2420Transmit
Description: Communicates with hardware to transmit messages.

transmit()	Transmits a message.
msg	The payload.
header	Header information for TDMA protocol.
modify()	Modifies part of outgoing transmission that has yet to be sent yet without interrupting the transmission.
header	New header information for TDMA protocol to replace existing.

Events

	CC2420Transmit does not have events

Subcomponent: CC2420Receive
Description: Processes incoming messages and notifies other components.

Events

receive()	When a new incoming message has been received.
msg	The payload.
header	Header information for TDMA protocol.

Subcomponent: Alarm
Description: Provides the clock to the TDMA component and alarm Event.

setNextAlarm()	Duration before next alarm event.
time	Duration until next alarm event is fired.

Events

alarm()	Alarm was set.

Subcomponent: Controller
Description: The hub of the TDMA component and contains all TDMA specific logic.

send()	Loads controller with a message that will be sent next time it has access to the medium.
msg	The payload to be transmitted.

Events

_	Controller does not have events

SubComponent: GPIOCapture
Description: Raises events on pins, in this case on the SFD pin.

Events

SFD()	Every transmission has an SFD which raises this event.

Component: Hardware
Description: Used here to show other components dependency, the functionality the hardware provides are all abstracted for explanation purposes.

radioTransmit()	Sends the data by radio.
msg	The payload to be transmitted.
header	Header information for TDMA protocol.
modifyTransmit()	Modifies TDMA header of outgoing transmission without interrupting the transmission.
header	New header information for TDMA protocol.

Events

receive()	When a new incoming message has been received.
msg	The payload.
header	Header information for TDMA protocol.
SFD()	All transmission have an SFD bit sequence which raises this event.