ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Andreas Huber

A Load Adaptive & Frequency-Hopping MAC-Protocol

Semester Thesis, SA-2009-05
February until June 2009

Advisors: Roland Flury, Philipp Sommer
Professor: Prof. Dr. Wattenhofer

Abstract

A MAC-Protocol is proposed in this thesis that adapts dynamically to load variations. The protocol makes use of multiple frequencies to lower interference. It is energy efficient by scheduling the slots where packets should be sent or received allowing the nodes to power off the radio when no packets are transmitted. A method is proposed to allow the nodes to switch reliably between schedules of different data rates over an unreliable channel.

The protocol was implemented and tested in this thesis on the TinyNode platform. It was shown that it adapts well to variable traffic conditions and that multiple frequencies can be used to efficiently reduce interference in a wireless sensor network.

Acknowledgements

First of all I would like to express my sincere gratitude to Prof. Dr. Wattenhofer for giving me the opportunity to write this semester thesis in his research group.

I would also like to thank my advisors Roland Flury and Philipp Sommer for their constant support during this semester thesis. They helped me to solve the difficult problems of this thesis. Without their assistance, this work would not have been possible.

Furthermore, I would like to thank my family and my friends for supporting and motivating me during this thesis.

Contents

List of Figures

Chapter 1

Introduction

This thesis studies the transmission of data streams over several hops in a wireless network. When a sender needs to route a set of packets to a destination, several network nodes in-between may need to receive and retransmit the packets. As a result, several nodes may try to send a packet at the same time, resulting in packet loss due to interference. A schedule, which assigns to each node the time slots when it should listen for packets and when it should forward received packets not only helps to reduce interference, but can also save energy, as the nodes may power off the radio during idle times. The algorithm proposed in this thesis achieves this. It further assigns the time slots adaptively to load variations and offers the possibility to use multiple frequencies by selecting a random frequency for each assigned slot.

Typical sources of energy waste in wireless sensor networks are idle listening, packet loss caused by interference, overhearing of packets destined for other nodes, over-emitting which is caused by transmissions to nodes that are not listening and control-packet overhead [2]. The protocol introduced in this thesis addresses all of these sources of energy waste. Idle listening and overhearing are reduced significantly by powering off the radio outside the scheduled slots and the schedule also prevents over-emitting in most cases. The protocol has no control-packet overhead as long as there are packets to transmit. The necessary control data is piggy backed to the data packets and acknowledgements. The packet loss caused by interference is reduced by using random frequencies in each scheduled slot.

The utilization of multiple frequencies can also be seen as a collision avoidance (CA) measure in contrast to other MAC-Protocols where collision avoidance is achieved through separation in the time domain. Using the frequency domain as well reduces the probability of collision by the number of available channels.

This leads us to the second property of the proposed algorithm. The algorithm should choose the slots to use in the schedule adaptively to load

variations. This is done by using multiple schedules at different rates. The algorithm then chooses which of the schedules should be used at any time. A method is proposed and implemented in this thesis to construct these schedules and to change between the schedules of different rates even in the presence of packet loss (messages and acknowledgements). The schedule between the nodes may break apart if the changes between the schedules of different rates are done carelessly. The algorithm presented in this thesis ensures that the nodes can switch from one rate to another in the presence of packet loss.

The algorithm is computed locally on each node based on a random seed that needs to be exchanged between the neighboring nodes in the initialization phase. The nodes negotiate in every packet and acknowledgement at which rate they want to send the following packet(s).

The implementation of the proposed algorithm was a main part of this thesis. The algorithm was implemented for the TinyNode platform [8] running the TinyOS [10]. Several test cases were carried out and are presented in the evaluation of this thesis.

The following chapters contain an overview over the related work (chapter 2), a detailed description of the protocol (chapter 3) and its implementation (chapter 4), an evaluation with several test cases (chapter 5) and finally the conclusion (chapter 6).

Chapter 2

Related Work

There exist already various propositions for MAC-Protocols for wireless networks and wireless sensor networks. There are some MAC layer protocols designed for wireless devices such as IEEE 802.11 [1]. They are not suitable for sensor networks because they will consume a considerable amount of energy as they continuously sample the medium for activity. Further, such protocols require the nodes to exchange control packets to avoid collisions. The overhead introduced by such control packets is considerably high because the packets used in wireless sensor network are usually of small sizes.

Many applications for wireless sensor networks require only small data rates but a node and network lifetime that should be as long as possible. Energy consumption matters highly because a sensor node dies if it runs out of energy. The radio consumes much energy if it is turned on, even if the radio is just idle listening. This is also true for the TinyNode [8]. The radio module of the TinyNode has a current consumption of $33mA$ in transmit mode, $14mA$ in receive mode and below $1\mu A$ in sleep mode as stated in the data sheet of the TinyNode [9]. The CPU of the TinyNode uses $2mA$ in contrast. Therefore, it is reasonable to use an algorithm that allows to power off the radio when it is not needed.

Various other MAC-Protocols for wireless sensor networks were proposed which try to reduce the energy consumption by reducing the time where the radio has to be turned on. Some protocols use a scheduled access, others are based on random access [6]. The focus will be set on random access based protocols in the following because the protocol implemented in this thesis belongs to this category, too.

In many of the random access based protocols, the nodes wake up periodically to enter reception mode. The radio is turned off again if the channel is idle. Otherwise, they try to receive a packet.

The time at which all nodes perform this carrier sensing is synchronized in some protocols such as S-MAC [12] or T-MAC [11]. These protocols

are called synchronous protocols. S-MAC introduced the idea of a synchronized wake up schedule where all nodes wake up periodically to exchange signalling messages to determine which nodes will communicate afterwards. All inactive nodes will then turn off their radio again. Other protocols like Low power Listening [4] or WiseMAC [3] use an asynchronous random access schemes and do not synchronize the schedules between multiple nodes. Instead, each node periodically wakes up to listen for activity on the channel. To initiate a transmission, a sender sends a long wake-up preamble to ensure that the receiver is in reception mode before the actual data is transmitted. In WiseMAC, the overhead of transmiting a long wake-up preamble is reduced significantely by learning the wake-up schedule of the receiver and adapting the wake-up preamble to it. The wake-up preamble is chosen on the sender based on the time since the last transmission and the accuracy of the learned schedule of the receiver. It may be very short if packets are transmitted frequent and grows up to a maximum of a complete period as without any knowledge of the wake-up schedule of the receiver.

The mentioned protocols don't make use of multiple frequencies and they all contain a static period chosen at design time that is used to sample the medium. This means that this period has to be chosen carefully and the period will probably not be optimal for all nodes in the network as it is often the case that the nodes nearer to the root will have more packets to transmit than the nodes far away.

The protocol introduced in this thesis is different in this two parts as it makes use of multiple frequencies and it has no fixed length wake-up period but selects the slots to wake up for communication adaptively and independently for each link in the network depending on the current load.

Chapter 3

Protocol Design

The protocol proposed in this thesis is described in this chapter. Everything related to the actual implementation on the TinyNodes is described in chapter 4.

The protocol uses slots to receive and transmit packets. In each slot, a node either transmits a packet to a neighbor, receives a packet from a neighbor or has its radio turned off. Time synchronization is needed to be able to use a slotted schedule between the nodes. This appears to be a drawback but we think that time synchronization is needed anyway for many applications.

The peculiarity of the protocol lies in the selection of the slots. The protocol selects the slots adaptively at a rate suitable for the current traffic load and selects for each slot a random frequency selected to enable frequency-hopping. The slots to be used between two communicating nodes are selected randomly but such that the slots can be computed locally by the two neighbors. For this purpose, a random seed is exchanged during the initialization phase. How this is done is explained in section 3.1 and 3.3

The term schedule is used in this chapter to refer to the slots and frequencies selected at a specific rate between a given sender and receiver node. Each node runs multiple such schedules in parallel, the current implementation uses two schedules per neighboring node to communicate in both directions. This leads to four schedules per node for the linked list topology on which the algorithm was evaluated in this thesis. Each slot can be assigned at most to one of the schedules that run in parallel because the nodes can only communicate with one neighbor at a time. This means that all but one schedule have to skip a slot if multiple schedules want to use the same slot. Section 3.2 explains how the protocol handles these conflicts.

3.1 Schedules at Variable Rates

The challenge of switching between schedules at different rates in the presence of packet loss is addressed in this section and a solution is presented to construct the schedules such that it is simple to switch between rates. Only the communication between one sender and one receiver node is considered in this section.

3.1.1 Problems with Variable Rates

The nodes should be able to switch from a schedule at a specific rate to another schedule at another rate in our protocol. It is important that two communicating nodes switch simultaneously from one schedule to another. Otherwise, they are no longer able to communicate further with each other in the future because the sender and receiver will try to use different slots and/or different frequencies. But a simultaneous transition cannot be guaranteed: The sender can request from the receiver that it want to switch to another rate and wait for an acknowledgement. The sender can be sure that the receiver got the message if it receives an acknowledgement. Otherwise, it has to assume the receiver didn't get the message. But the receiver cannot know whether the sender actually received the acknowledgement successfully or not. To be sure, it could also request an acknowledgement for its acknowledgement. But the problem to be unsure is only passed back to the other node and not solved. The two nodes have to wait always for another acknowledgement to be sure that both of them will decide at the same time to switch to another rate.

Only one acknowledgement per real messages is sent in the usual implementations. Sequence numbers are added to the messages to detect and resolve cases where an acknowledgement was lost. This works fine for data messages but it is not sufficient for a command to switch to another rate. The two nodes would not be able to communicate any more if only one of the two nodes decides at one point in time to switch to another rate. Fig. 3.1 illustrates how it could look like if an acknowledgement is lost.

A possible solution in this scenario would be that the sender uses the old and the new schedule in parallel until it can be sure that the receiver has switched successfully to the new schedule. But this sounds simpler than it actually is. It is not enough if the two nodes finally use again the schedule at the same rate but it is also necessary that both nodes still use the same random seed. Otherwise they will not be able to compute the shared schedule any more. It may be the case that the two nodes end up with different random seeds because the random seed is updated during the computation of the schedule and maybe updated differently if they use different schedules. It would be necessary to exchange further control information between the two nodes to resolve this conflict. The first node could for example

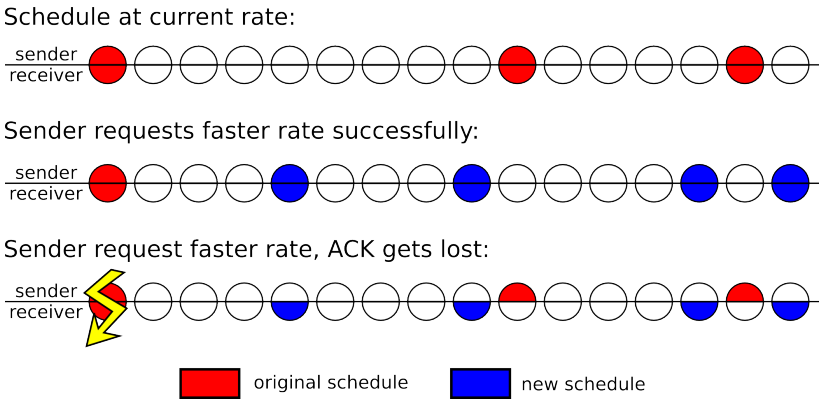

Figure 3.1: The figure illustrates the problem when switching between schedules of different rates if an acknowledgement is lost. The circles correspond to all available slots. The upper half of each circle corresponds to the actions performed by the sender, the lower half to the actions performed by the receiver. The nodes turn their radio on at the appropriate frequency in the colored slots. The red slots are selected by the original schedule, the blue slots are selected by the new schedule.

retransmit its request to change the rate on the old schedule and include its current seed for the new schedule until it receives an acknowledgement from the second node either on the old or the new schedule. Additional problems arise if the old and the new schedule want to use the same slot but at different frequencies.

A much simpler approach is presented and implemented in this thesis. The key idea is to construct the schedules at different rates such that they overlap. This way, the two nodes use a common schedule even if they currently do not use the schedule at the same rate. Further, the algorithm is designed such that the random seed shared by the two nodes is altered independently of the rate that is actually used. This makes it unnecessary to exchange the random seed any more after the initial exchange, especially not during a rate switch.

3.1.2 Construction of Schedules at Variable Rates

The schedules at different rates are constructed as follows: Half of all slots are selected randomly to construct the schedule at the fastest rate. The random selection serves the purpose to make the schedule independent from the schedules used between any other nodes. The fastest rate is named rate 0 in this thesis. Every slower schedules is constructed by reducing this first schedule by using only half of the slots of the next faster schedule. Therefore, rate 1 is the next slower schedule which includes every second slot of the fastest schedule. The schedule at rate 2 includes every second slot of the

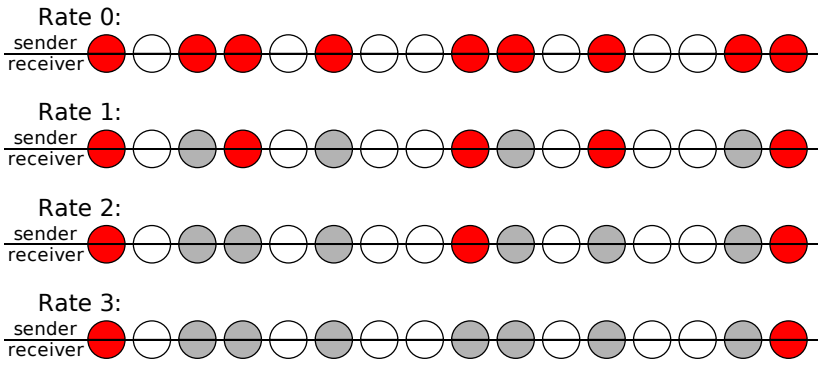

Figure 3.2: The overlapping schedules from rate 0 to 3

schedule at rate 1 and thus every fourth slot of the fastest schedule. And so on for the schedules at slower rates. This means that the slots selected at a specific rate are a subset of the slots selected at every faster rate. The construction of the different schedules is illustrated in Fig. 3.2.

The information which rate should currently be used is exchanged between the sender and the receiver in each message and acknowledgement. This allows the switch to another transmission rate in every transmission and can be demanded by the sender or receiver node.

3.1.3 Switch between Schedules of Different Rates

The transition from one rate to another is much easier with the schedules introduced above. The sender can request to switch to a schedule with a slower or a faster rate. The sender switches only to the new schedule if an acknowledgement is returned from the receiver. The receiver acknowledges the request and switches to the new schedule when it receives such a request from the sender. If the acknowledgement is lost, the sender and receiver will still be able to communicate with each other because they can still communicate in all the slots contained in the slower of the two schedules. The two nodes will use the same rate again as soon as they have successfully exchanged a message and an acknowledgement. The same holds if the receiver requests to switch to another schedule in the acknowledgement.

The switch-over to another rate where an acknowledgement is lost is illustrated in Fig. 3.3

3.2 Local Conflicts Between Schedules

The construction of the schedules introduced above is independent of any other schedules between other nodes. This makes its computation simple such that each node can do it locally for its own schedules. But when mul-

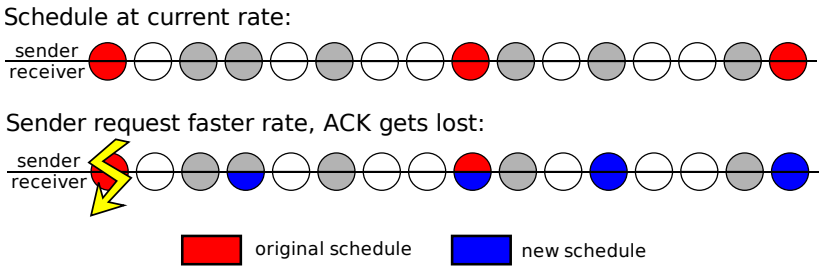

Figure 3.3: Rate switch with acknowledgement loss. The nodes use the same schedule again after the next successful transmission.

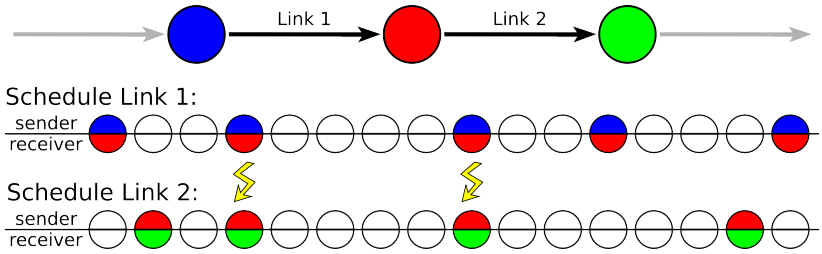

Figure 3.4: Local conflicts between multiple schedules. The red node has conflicts between the schedules of link 1 and link 2.

tiple schedules run in parallel on a given node, a schedule may try to choose the slots already assigned to another schedule with a certain probability. These conflicts are named local conflicts in this thesis. Such a case is illustrated in Fig. 3.4.

In a local conflict, a node will turn on its radio and listen or transmit unsuccessfully because its neighbor is busy doing something else in the very same slot. This would be a serious reason of energy waste, especially if the nodes run on very fast rates where the probability of local conflicts is higher.

One solution to this problem would be to compute a coloring for the links such that any adjacent link will have another color. In each slot, only links of one specific color are scheduled in the hole network to prevent local conflicts. The schedule as described above would then only take place on a subset of slots, maybe without the random part. But this solution comes at the high cost of the computation of a coloring.

This thesis proposes that the local conflicts should be accepted but the neighbor should be informed about local conflicts such that the conflicting slot can be skipped to prevent waste of energy. The priority in the reservation of the slots for the parallel schedules is done in a first-come-first-serve manner where the next slot for a certain schedule is reserved in the current slot. A node that detects such a local conflict will try to inform its neighbor about it in the packet or acknowledgement. The two nodes will then skip the

conflicting slots in their schedule. This will not work out in certain cases, for example if the receiver requests to skip some slots in the acknowledgement but the new next slot is already assigned to another schedule in the sender. The sender cannot inform the receiver in this case that now even more slots should be skipped. The solution to inform the neighbor about slots to skip worked out pretty well in the test cases presented in the evaluation.

3.3 Selection of the Appropriate Rate

This last section handles the question how the appropriate rate should be chosen between a sender and a receiver. Sender and receiver calculate their preferred rate and the slower of the two rates will be used. The sender calculates the preferred rate in each slot and transmits it in the packet to the receiver. The receiver acknowledges this rate if it agrees or it returns a slower rate in the acknowledgement if it doesn't agree. The sender will use the last rate for which it received an acknowledgement, the receiver will use the last rate for which it sent an acknowledgement.

How the sender and receiver should calculate their preferred rate is an interesting question. The sender has packets in a queue that it wants to transmit and the receiver has free memory available to receive messages. The first idea was to control the rate such that the number of packets in the queue stays about constant. This first implementation produced unstable rate changes even for periodic traffic which was understandable after thinking a little more about it. The second approach was to calculate the rate as a function of the queue size. This was done with the following idea: The sender calculates the preferred rate such that the average time the packets have to wait in the queue of this node is constant. This rule provides stable results for various scenarios as shown in the evaluation. How the rate is computed by this rule is explained next.

Time is measured in number of slots in the analysis. Let t be the average number of slots between two transmissions of the sender. When sending at rate r, we have that $t = 2 \cdot 2^r$. Then, the expected number of slots T a packets remains in the queue of fill size q is

$$T = q \cdot t = q \cdot 2 \cdot 2^r$$

And solved for r

$$r = log_2(T/q) = log_2(T) - log_2(q)$$

In the implementation, the rate is computed according to that rule as

$$r = s - \lfloor log_2(q+1) \rfloor$$

where s denotes the slowest rate which is used for $q = 0$. The logarithm has to be rounded to an integer because only integer rates are allowed and it is

computed from $q+1$ to avoid the case of $log(0)$ where an infinitely slow rate would be requested otherwise.

One has to pay attention when configuring the slowest rate s and the maximum possible queue size $qMax$ because they are related to the fastest rate f that will be chosen through the way the rate is computed:

$$f = s - \lfloor log_2(qMax + 1) \rfloor$$

In the current implementation, the receiver calculates the fastest rate it would accept depending on the rate it can forward messages and on the available memory to buffer incoming messages such that it will not run out of memory before it can slow down the incoming traffic.

Chapter 4

Implementation

The protocol was implemented in this thesis for the TinyNodes [8]. The protocol was written in the programming language nesC [5], a C-extension, and bases on the TinyOS [10] and the SlotOS library [7]. The implementation is splitted into three modules: RandomSlot, RandomSlotSchedule and SendRandomSlot.

The RandomSlot module handles the computation of the schedule between two nodes and chooses which slots should be used for communication. The RandomSlotSchedule module handles the coordination and conflicts between the schedules running in parallel. This module uses the TimeSchedule and TimeSync modules of slotos. The time schedule is used to divide the available time between multiple applications running on the nodes. The RandomSlotSchedule module places all time slots for the packets in the period assigned to it from the TimeSchedule. Then, it signals the start and end of a assigned slot. The third module, SendRandomSlot, actually sends the packets to the neighbors in the slots signalled by RandomSlotSchedule and handles the negotiation of the appropriate rate to use with the neighbors. Next, these three modules are explained in more detail.

4.1 RandomSlot

The RandomSlot module (implemented in RandomSlotC and RandomSlotP) chooses slots on a random basis to be used in a schedule. The module contains three commands. The command init() to initialize the schedule with a random seed, the command getNextSlot() to calculate the next slot in the schedule based on the rate given and the command getChannelForCurrentSlot() to get a random channel for the current slot.

The module choses half of all available slots randomly based on the random seed set at initialisation. A subset of these slots is chosen for the schedule, depending on the rate. For rate 0, all of the slots are used. Every second slot is used at rate 1, every fourth slot is used at rate 2, every eight

slot is used at rate 3 and so on.

The necessary calculations of random numbers is kept low by using every bit of the random number individually to select or deselect a slot. A counter which counts the selected slots is then used to decide if a slot is included for a given rate or not. This operation can be done using bitwise operations.

4.2 RandomSlotSchedule

The RandomSlotSchedule module (implemented in RandomSlotScheduleC and RandomSlotScheduleP) handles the local conflicts between the independent random slot schedules. It offers functions to add new schedules and to reserve the next slot for a schedule at a desired rate. Two schedules are added for each neighboring node, one schedule to send and one to receive packets.

This module subdivides its scheduled period from TimeSchedule into slots to send single packets and determines which of these slots are used for sending or receiving packets to and from the neighbors. The start and end of a slot that is used to communicate with a neighbour is signalled to the module SendRandomSlot which performs the actual communication. Further, the module sets the radio already to the appropriate channel if the use of multiple channels (frequencies) is enabled such that the owner doesn't have to care about this. But the module does not start the radio. This is left to the owner as it is also possible that the radio will not be used at all if the root node wants to communicate to the pc over the serial and not over the radio.

4.3 SendRandomSlot

The SendRandomSlot module (implemented in SendRandomSlotC and SendRandomSlotP) actually sends the packets over the radio. This module chooses also at which rate the packets should be transmitted and negotiates this with the neighboring node. To do this negotiation, the module uses one byte of the payload in the packets and the acknowledgements. To build a reliable channel, the sequence number module of slotos is used and sequence numbers are included in each message and acknowledgement.

The SendRandomSlot module is signalled by RandomSlotSchedule at the start and at the end of a slot for a given schedule. One packet is sent in every scheduled slot. If there are no packets in the queue, an empty message is sent which contains only one byte to negotiate the rate to use.

4.3.1 Overview over the Actions Performed in a Slot

Next, the steps performed by this module in each slot are explained shortly. For a send slot:

1. The radio is started unless the communication passes over the serial (destination id is zero).
2. A small backoff is waited after the successful start of the radio.
3. After this backoff, the message is prepared to send: The preferred rate is calculated and written into the message as well as the appropriate sequence number.
4. If the transmission is successful:
   - An acknowledgement is received which contains the sequence number for validation, the rate that will be used to chose the next slot as well as the number of slots that have to be skipped.
5. If the transmission (or acknowledgement) fails:
   - No acknowledgement is received and the last rate that was acknowledged by the receiver is used to chose the next slot.
   - The same message will be resent in the next slot with the same sequence number but possibly with a different rate.
6. The radio is shut down to finish the current slot.

For a receive slot:

1. The radio is started.
2. After a successful start, the radio waits for a message in receive mode.
3. If a packet is received successfully:
   - An acknowledgement is sent, containing the current sequence number, the rate that will be used to chose the next slot and the number of slots that have to be skipped.
4. If the transmission fails:
   - No packet is received in the current slot.
   - The last rate that was acknowledged to the sender is used to chose the next slot.
5. The radio is shut down to finish the current slot.

4.3.2 Negotiation of the Appropriate Rate

The negotiation of the rate between the sender and the receiver is implemented as explained in the protocol description. For that purpose, one byte is included in each packet and acknowledgement. Three values are encoded in this byte:

- Four bits of the message are used to encode the rate. This allows rates from 0 to 15 resulting in a factor of $2^{15} = 32768$ between the fastest and the slowest rate which is surely more than enough. Rate 0 is the fastest rate.
- Three bits of the message are used to encode the skip counter. The skip counter indicates that the next 0-7 slots should be skipped.
- The last bit of the message is used by the receiver to reject a message if no memory is available to store the message.

4.4 Further Modules

A couple of other modules were developed during this thesis to support testing, logging, the initialization of the protocol and other things. The most important ones are mentioned here shortly.

The FastList module allows to send and receive commands over a list topology. It was used to exchange the random seed in the initialization phase and to send commands to the nodes. The RadioChannel module allows to set the radio to several different frequencies. The transmission at different frequencies was evaluated and the distances between the offered channels were chosen such that transmissions on different channels do not interfere. The TrafficGenerator module allows to produce periodic traffic and packet bursts for test purposes. It is configurable via the FastList. The Router module is able to route packets in a network, buffers the packets in different queues for each neighbor until they can be sent and manages the memory buffer for the messages.

Chapter 5

Evaluation

The proposed algorithm was tested on TinyNodes [8] for this evaluation. The implementation was evaluated using five nodes in a list topology. Each node communicated with the node whose id was one higher or one lower than its own id. The five nodes were placed for the evaluation like indicated in Fig. 5.1 such that the nodes interfere definitely.

The algorithm scheduled slots to transmit messages in both directions of the list but data was transmitted only towards the root in the test cases presented next. In the other direction were only the hello messages exchanged at the slowest possible rate as it is done by the protocol if no data packets have to be transmitted.

The mixture of periodic traffic and a burst of many packets at a time was used in all evaluated test cases. Each of the five nodes generates one packet every five second that has to be transmitted to the root. This could correspond to a sensor reading in a real world application. After 200 seconds, a burst of a thousand packets is generated at node 5 and transmitted to the root. This could correspond to a read out of the flash memory of a sensor node.

The first set of tests was done using rates from rate 0 down to rate 7. The tests were run for different number of possible frequencies that could be selected by the schedule. The test was done using the same frequency on all links and using two, four and eight different frequencies. The second set of tests was done using rates from rate 1 down to rate 8 and using again one, two, four or eight different frequencies.

5.1 Generated Plots

Six plots were generated for each of the eight test case. The complete set of plots is listed in appendix A. To make the plots more readable, only information about the links towards the root is shown. But the schedule in the other direction will slightly influence the results as it will use some slots

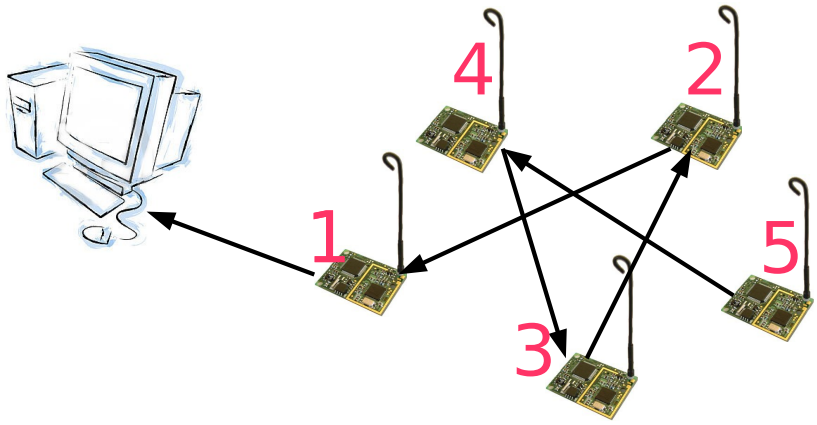

Figure 5.1: Test case setup.

that could have been used for the communication towards the root if the schedule in the other direction was not implemented at all. The contents displayed in the six plots are explained shortly in the captions of Fig. 5.2, 5.3 and 5.5. The first two plots are updated at every slot that is included in the current schedule. The other four plots are calculated over periods of 32 seconds to match the periods of the TimeSync module that underlies the whole algorithm. Like that, the effect that no packets can be transmitted for several seconds every 32 seconds is equal for every value displayed in the plot.

5.2 Discussion of the Results

Various results observed in the test cases will be discussed in the following with a focus on the adaptation of the rate and the benefit from the usage of multiple frequencies.

5.2.1 Adaptation of the Rate

Only the periodic traffic has to be handled before and after the packet burst. Each node participates in producing and transmitting this periodic traffic. The nodes nearer to the root have more packets to transmit because they have to forward all the packets from the higher nodes, too. This means that the nodes have to send at different rates. One can observe that this is actually the case as shown in Fig. 5.2. The nodes nearer to the root send at a faster rate and have more packets in their queues waiting to be sent as the send rate is a function of the queue size.

The reaction of the algorithm to a packet burst can be observed very well, too, in Fig. 5.2. One can see that the queues are filled up in one node after the other until all the nodes transmit at the fastest rate, rate 0. After

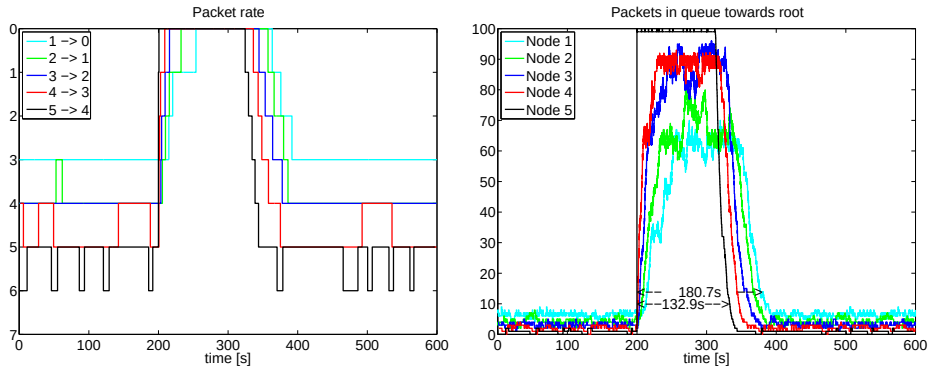

Figure 5.2: Test results of the first set using four frequencies. The left plot shows the rate that is used for each of the five links towards the root and the right plot shows the fill size of the queues towards the root.

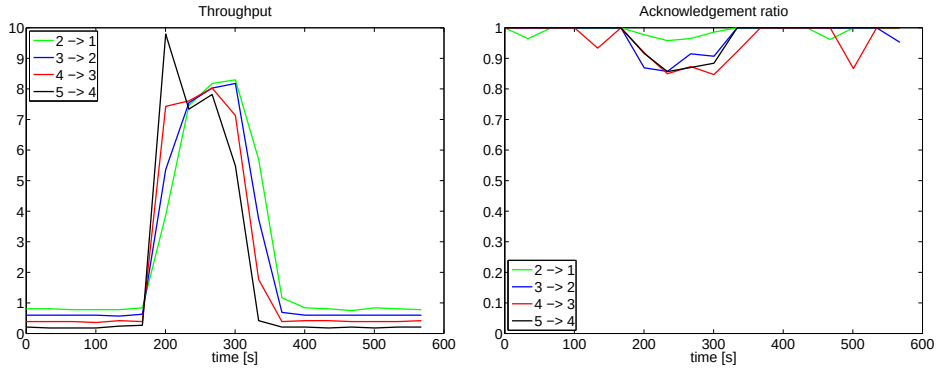

Figure 5.3: Test results of the first set using four frequencies. The left plot shows the throughput over each of the four links in packets per second and the right plot shows the ratio of successfully received acknowledgements.

the burst, it takes some time to clear out all queues because the nodes send slower and slower while the fill levels of the queues shrink.

Fig. 5.4 and Fig. 5.5 show that the algorithm adapts the rate in a reasonable way. We can see that the nodes didn't send many empty messages and that no packets had to be rejected because of memory problems. Empty messages are sent if the nodes have scheduled a slot when they have no data to send. This did only occur in the test case for node five because the periodic packets are generated at a rate just a bit slower than the slowest possible rate of the algorithm.

5.2.2 Interference and Local Conflicts

The transmission quality is shown and explained in Fig. 5.4. One can see that there are two principal reasons for failed transmissions where the sender receives no acknowledgement. The first and main reason is packet loss due to interference. There is a high probability that two links will transmit at the same time if all nodes send at the fastest rate. The second reason is given by local conflicts that cannot be resolved. The nodes try to resolve the local conflicts by informing their neighbor that one ore more slots should be skipped. This is not possible if the packet is lost due to interference. The probability to lose an acknowledgement was very low compared to the probability to lose a data packet in all test cases.

Fig. 5.3 shows the throughput and the ratio of successfully received acknowledgements between each communicating node pair over time. One can see for example that the throughput between node 5 and 4 is higher in the first measurement after the peak than after because there are less local conflicts with the communication between node 4 and 3 at that time because the communication between node 4 and 3 is still on a slower rate.

5.2.3 Multiple Frequencies

Using multiple frequencies works actually quite well and offers a significant benefit in the evaluated test cases. The plots in Fig. 5.4 show that the interference can be reduced significantly by using multiple frequencies. Further, the local conflicts are reduced, too. This is the case because the nodes are able to resolve more of their local conflicts by informing the neighbor to skip one or more slots.

5.2.4 Slow Down on Congestion

Another approach to reduce packet loss due to interference would be to lower the data rate on congestion. This was not yet implemented in the protocol but it was tested with the second set of tests where the fastest possible rate was set to rate 1 instead of rate 0. The results are shown the appendix in Fig. A.5 to Fig. A.8.

5.2.5 Summary

The test results show that the use of multiple frequencies reduces the interference which leads to a higher throughput and reduces the waste of energy. The results show also that transmitting at a slower rate in the presence of congestion reduces the interference and energy waste, too, but leads to a smaller throughput.

To summarise the result, the acknowledgement ratio and the time needed

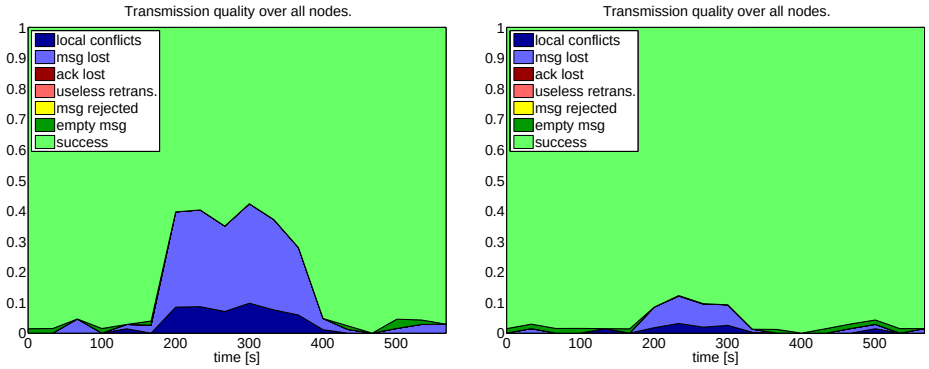

Figure 5.4: Test results of the first set using one frequency (left) and four frequencies (right). These plots show the transmission quality over all nodes (as seen from the sender). The dark blue part shows the fraction of the local conflicts that couldn't be resolved because the neighbor could not be informed that one or more slots should be skipped. The blue part shows the fraction of the messages that are lost in transmission, mostly due to interference. The dark red part shows the fraction of lost acknowledgements which is very low. The light red part shows the fraction of packets that were unnecessary retransmitted due to a lost acknowledgement. The yellow part shows the fraction of messages that had to be rejected by the receiver because it was completely out of memory which did never occur. The dark green part are the empty control messages that were sent successfully if there are no data messages to send. Lastly, the light green part shows the fraction of the successfully transmitted and acknowledged data messages.

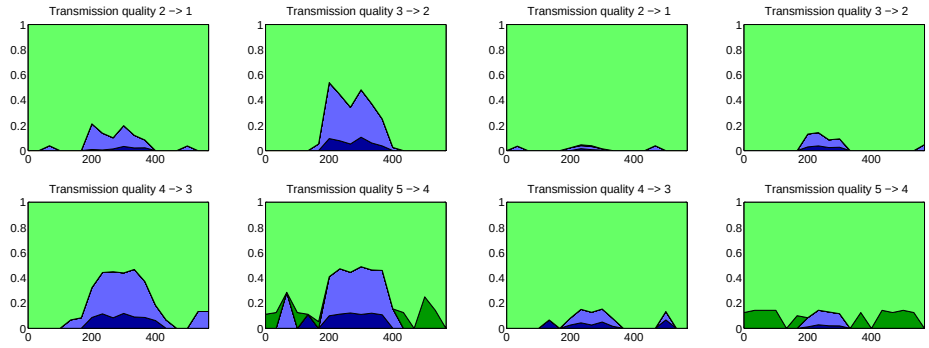

Figure 5.5: Test results of the first set using one frequency (left) and four frequencies (right). These plots shows the same results as Fig 5.4 but individually for each of the four links.

to send the packet burst is listed for all of the eight test cases. The acknowledgement ratio is computed over the hole evaluated 600s of each test case:

	rate 0-7	rate 1-8
1 freq.	67.3%	90.3%
2 freq.	83.4%	94.8%
4 freq.	92.1%	96.0%
8 freq.	95.6%	98.2%

The use of multiple frequencies raised the acknowledgement ratio from 67.3% up to 95.6%. In combination with the transmission at the next slower rate, an acknowledgement ratio of 98.2% could even be reached.

The following table shows the time needed to send the burst of a thousand packets from node five to the computer connected to node one. The values are scaled by the time needed with one frequency and rate 0-7:

	rate 0-7	rate 1-8
1 freq.	1.00	1.33
2 freq.	0.82	1.23
4 freq.	0.74	1.21
8 freq.	0.69	1.17

A speed up of 31% could be reached by using eight frequencies. Transmitting at a slower rate in combination with eight frequencies was shown to be only 17% slower but using clearly less energy because of the higher probability for successful transmissions. The transmission is not half as fast with the next slower rate as one might first expect. This is so because there are more than twice as many local conflicts at the rate 0 than rate 1 which results in skipped slots.

The evaluation showed that the proposed protocol can adjust well to load variations and that the usage of multiple frequencies reduces interference and allows a higher throughput.

Chapter 6

Conclusion

A MAC-Protocol for wireless sensor networks was proposed, implemented and tested in this thesis. The protocol schedules slots to receive and send packets between each pair of neighboring nodes. The radio is used in an energy efficient manner. It has to be turned on only during the assigned slots where actually a packet should be transmitted or received. The slots are selected adaptively at a rate corresponding to the current traffic conditions which allows to transmit energy efficient for variable loads. In the evaluation was shown that the protocol adapts well to dynamic traffic conditions. The protocol allows further to use a frequency-hopping technique where a frequency is chosen randomly for the communication in every slot. The evaluation has shown that the use of multiple frequencies can significantly reduce interference between the nodes if they transmit at high data rates. And the protocol was shown to work reliably.

The current implementation of the protocol is done to give a proof of concept and not to offer a full implementation ready to use in an arbitrary wireless sensor network. Various things are left to be done.

6.1 Future Work

There are several things that could not be done during this semester thesis. The protocol should be analyzed more formally and it should be compared to other MAC protocols that exist for wireless sensor network in terms of energy consumption, throughput and handling of traffic bursts or otherwise variable load. The test should also be done on other network topologies and other traffic conditions. And there are some optimizations that should be considered if the protocol is to be used in an actual implementation.

6.1.1 Possible Optimizations

Two optimizations were discussed during this thesis but not implemented: First, the time interval for a slot could be optimized. There are several time delays during one time slot that are currently not set as tight as possible. Sending multiple packets per slot is the second idea to optimize the algorithm. The overhead of computing the schedule, starting the radio at the appropriate frequency and waiting some delays in each slot is a little much if only one packet is transmitted in each slot.

6.1.2 Selection of the Appropriate Rate

The selection of the appropriate rate is another thing that should be adapted. The idea that the sender requests a rate that was computed as a function of the packets in the queue was shown to be useful. But one could use other functions than the one proposed in this thesis. And the same idea could be used for the receiver. The receiver could chose the fastest rate that it would accept as a function of the free memory that is still available.

In the current implementation, the receiver did take into account how fast it can forward the packets to the root in combination with a fixed slow down if it runs low on memory to choose which rate it would accept. This worked well on the list topology tested in this thesis but will not work well in other network topologies where one node has to forward traffic from multiple other nodes. Thoughts about the symmetry between the sender and the receiver lead to the idea that the same function could be used to calculate the rate, once based on the messages in the queue to send and once based on the free memory to receive messages.

Appendix A

Results

A.1 Test Cases - Complete Results

All plots of the test cases that were evaluated are included here for completeness. The test cases themselves are described in chapter 5.

A.2 Overview over the Source Code Written in this Thesis

Detailed documentation is contained in the files themselves. The bold files are the actual implementation of the protocol presented in this thesis.

- headers/FastListCmd.h — defines fast list commands
- headers/MacControl.h — defines settings of fast list
- headers/MultiTimer.h
- headers/MyAM.h — list of used am-types
- headers/MyTimeSchedule.h — settings for time-schedule
- headers/RadioChannel.h — frequencies for the available channels
- **headers/RandomSlot.h — protocol-settings**
- headers/Router.h — settings for router (buffer size)
- headers/SendRandom.h
- headers/TrafficGenerator.h
- interfaces/FastListCmd.nc — receive and send commands over fast list
- interfaces/FastListSchedule.nc — configure fast list
- interfaces/FastListState.nc — get state information from fast list
- interfaces/MacControl.nc
- interfaces/MultiTimer.nc
- interfaces/RadioChannel.nc — choose the frequency to use
- **interfaces/RandomSlotConf.nc — configuration/initialization**

- **interfaces/RandomSlot.nc — calculates the next slot for a schedule at a given rate**
- **interfaces/RandomSlotSchedule.nc — handles the parallel schedules between each neighbor**
- interfaces/Router.nc — send packets to any node in the network
- interfaces/RouterState.nc — query free message buffer
- interfaces/Routing.nc — maps each address to a next hop
- interfaces/RoutingPacket.nc
- **interfaces/SendPacket.nc — sends packets to a neighbor**
- interfaces/SendRandomConf.nc
- interfaces/TrafficGenerator.nc — produces periodic traffic or bursts
- **lib/FastList_RandomSlotConfC.nc — initialization**
- lib/FastListScheduleC.nc — implementation of the fast list
- lib/MacControlC.nc
- lib/MultiTimerC.nc — manage multiple timers in parallel
- lib/RadioChannelC.nc
- **lib/RandomSlotC.nc**
- **lib/RandomSlotScheduleC.nc**
- lib/RouterC.nc
- lib/Router_StaticList_Random_C.nc
- **lib/Router_StaticList_RandomSlotC.nc**
- lib/RoutingStaticListC.nc — routing for a list
- lib/SendRandomC.nc — send random, aloha
- **lib/SendRandomSlotC.nc — send and receive messages, negotiate rates**
- lib/TrafficGeneratorC.nc
- **impl/FastList_RandomSlotConfP.nc**
- impl/FastListScheduleP.nc
- impl/MacControlP.nc
- impl/MacSchedulerP.nc
- impl/MultiTimerP.nc
- impl/RadioChannelP.nc
- **impl/RandomSlotP.nc**
- **impl/RandomSlotScheduleP.nc**
- impl/RouterP.nc
- impl/RoutingStaticListP.nc
- impl/SendRandomP.nc
- **impl/SendRandomSlotP.nc**
- impl/TrafficGeneratorP.nc

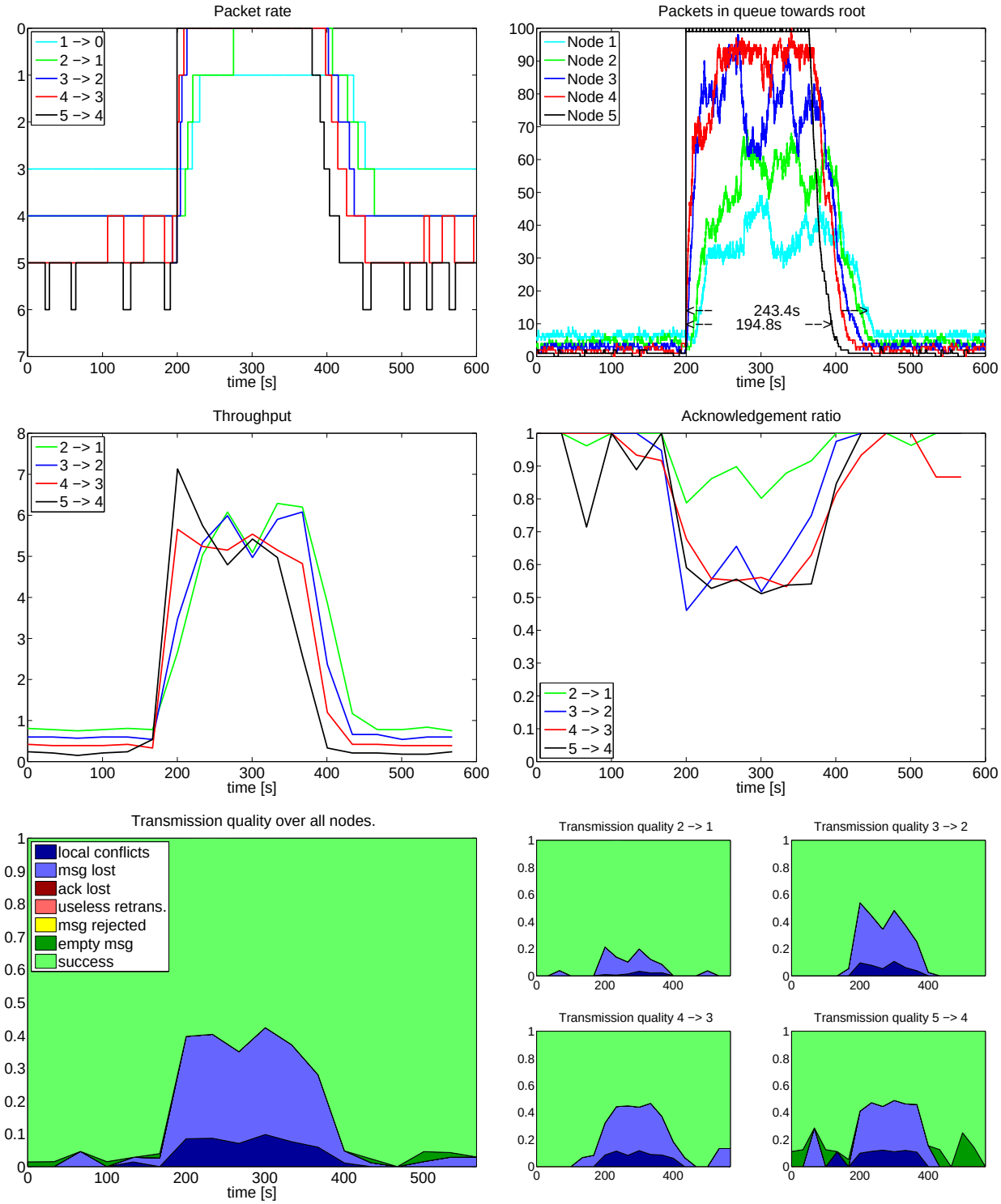

Figure A.1: 1 Frequency, Rates 0 - 7

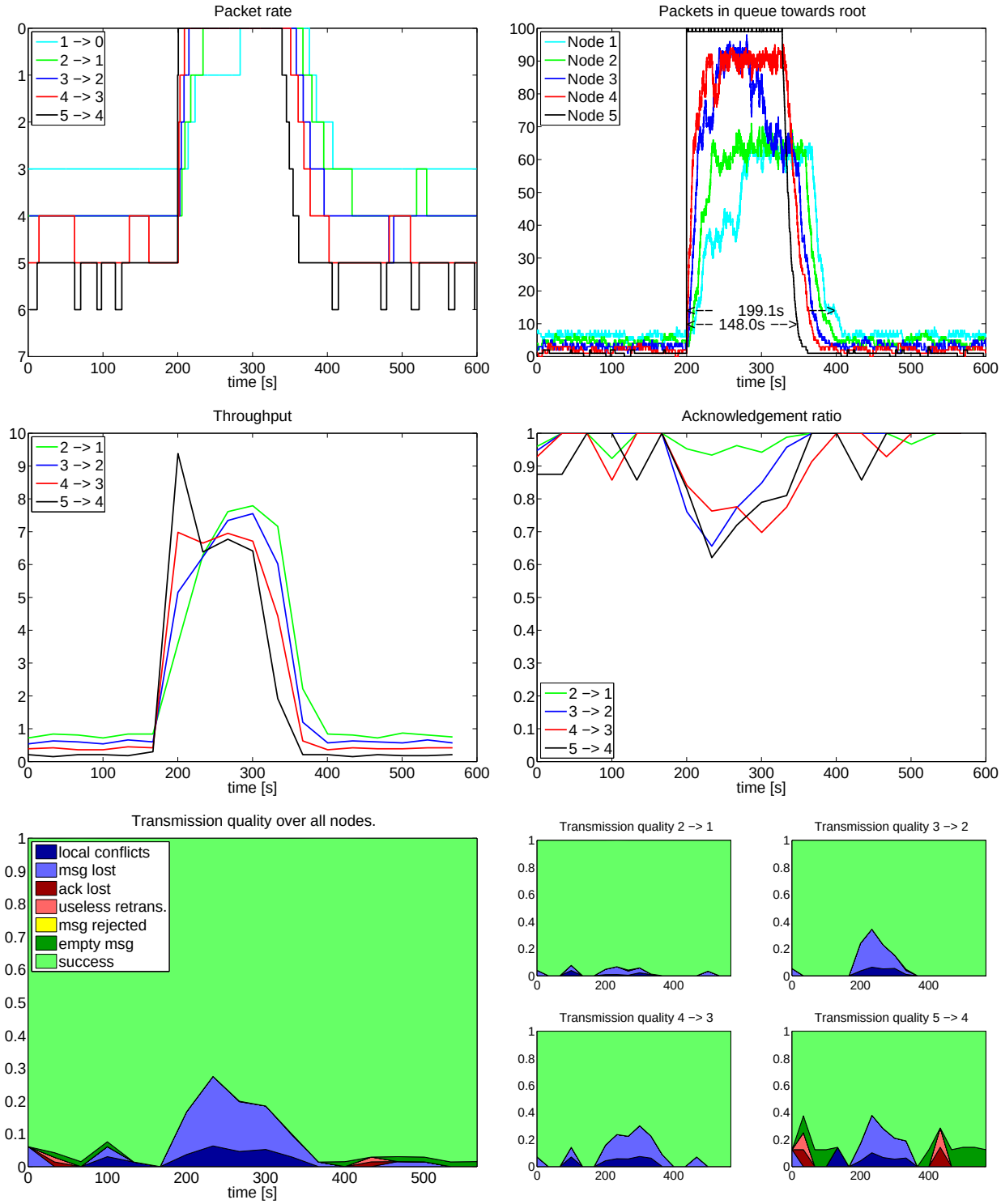

Figure A.2: 2 Frequencies, Rates 0 - 7

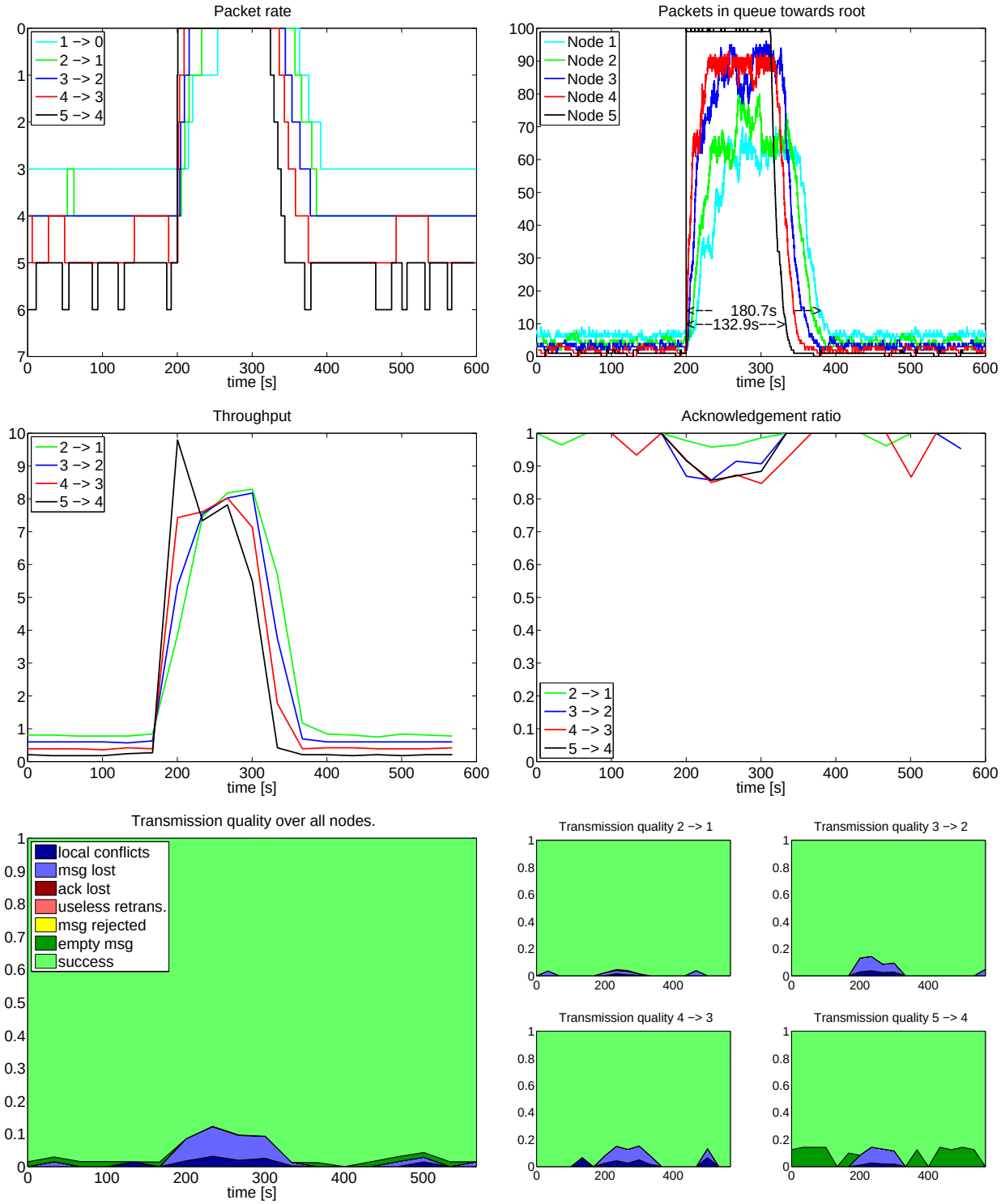

Figure A.3: 4 Frequencies, Rates 0 - 7

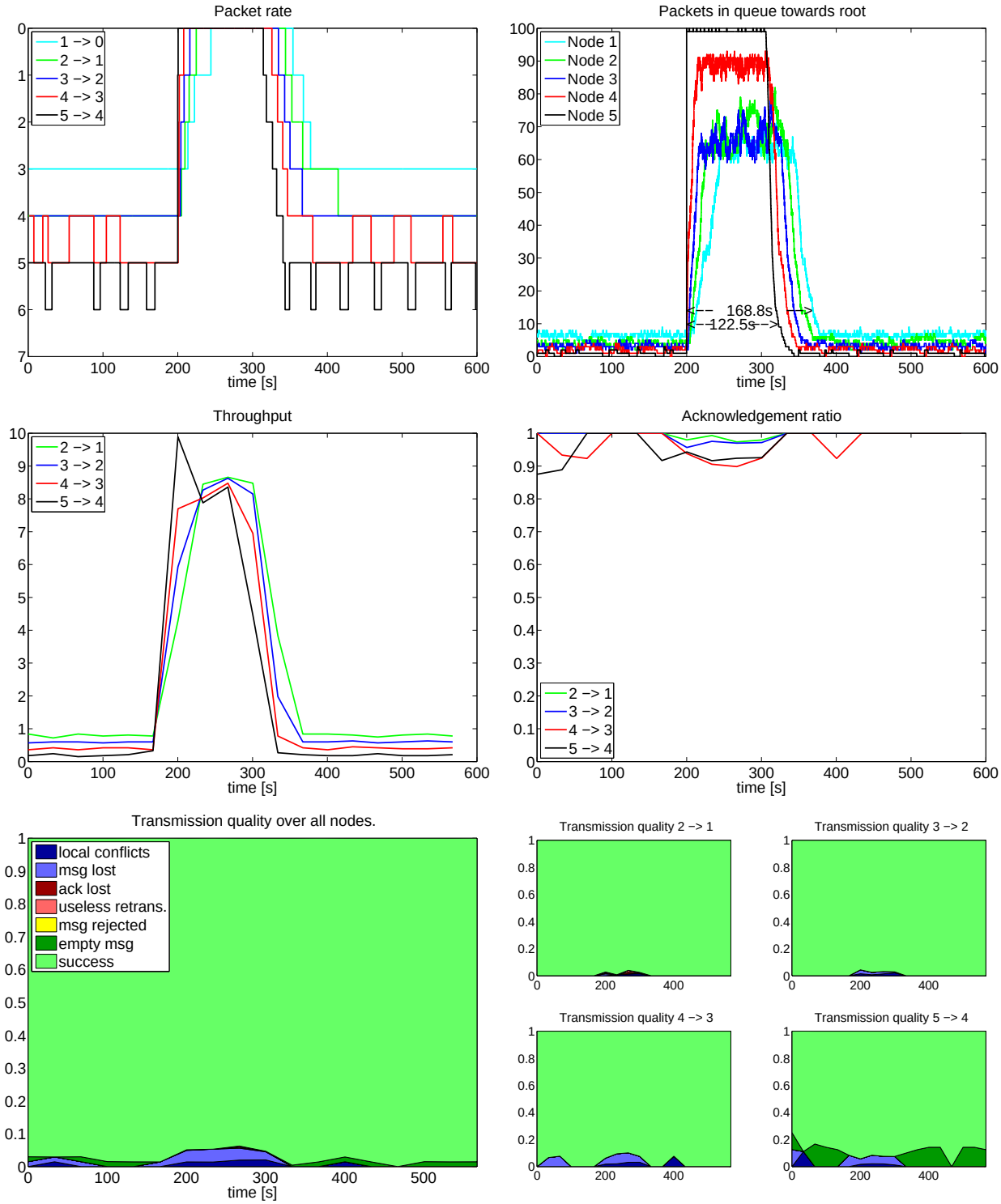

Figure A.4: 8 Frequencies, Rates 0 - 7

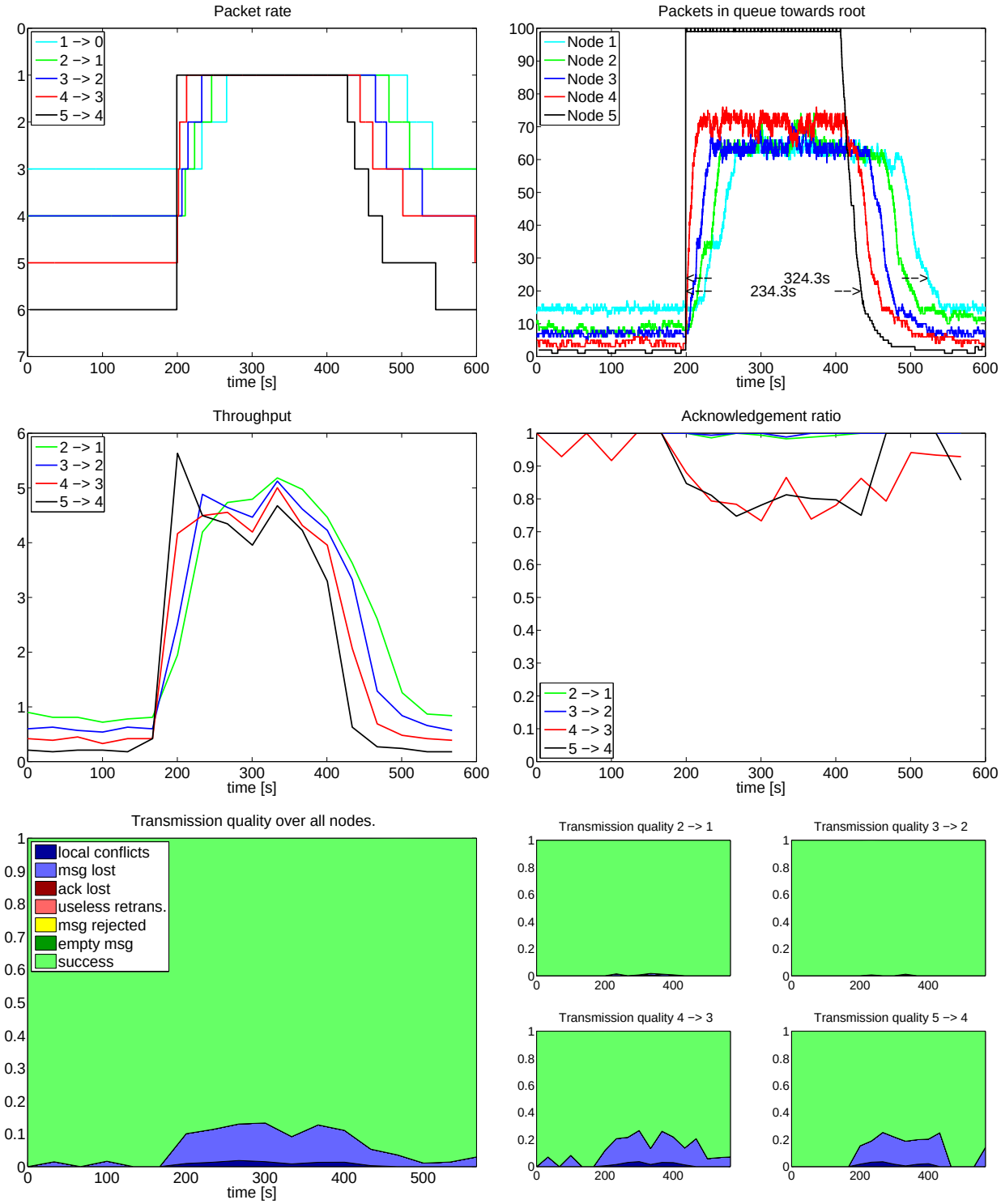

Figure A.5: 1 Frequency, Rates 1 - 8

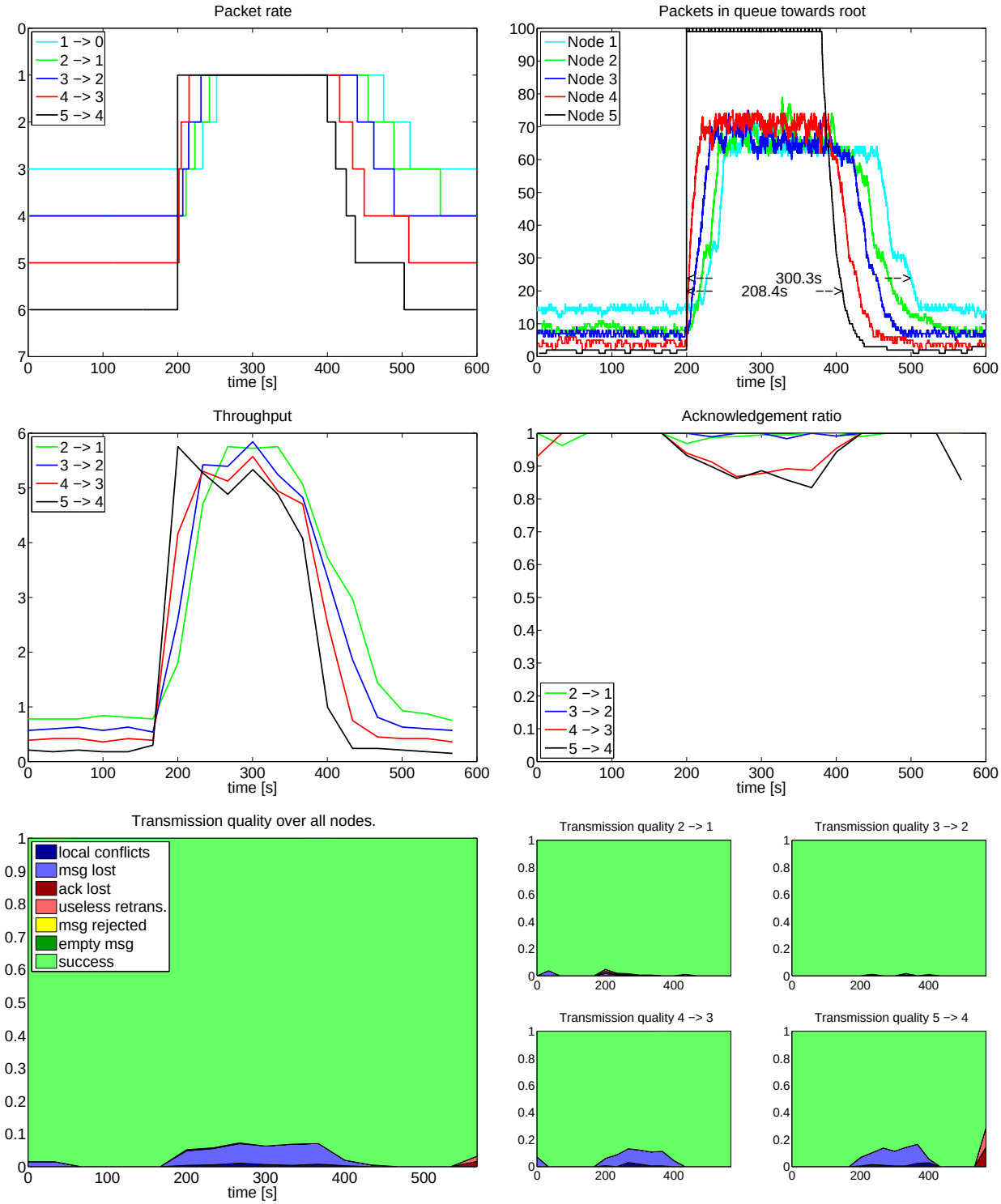

Figure A.6: 2 Frequencies, Rates 1 - 8

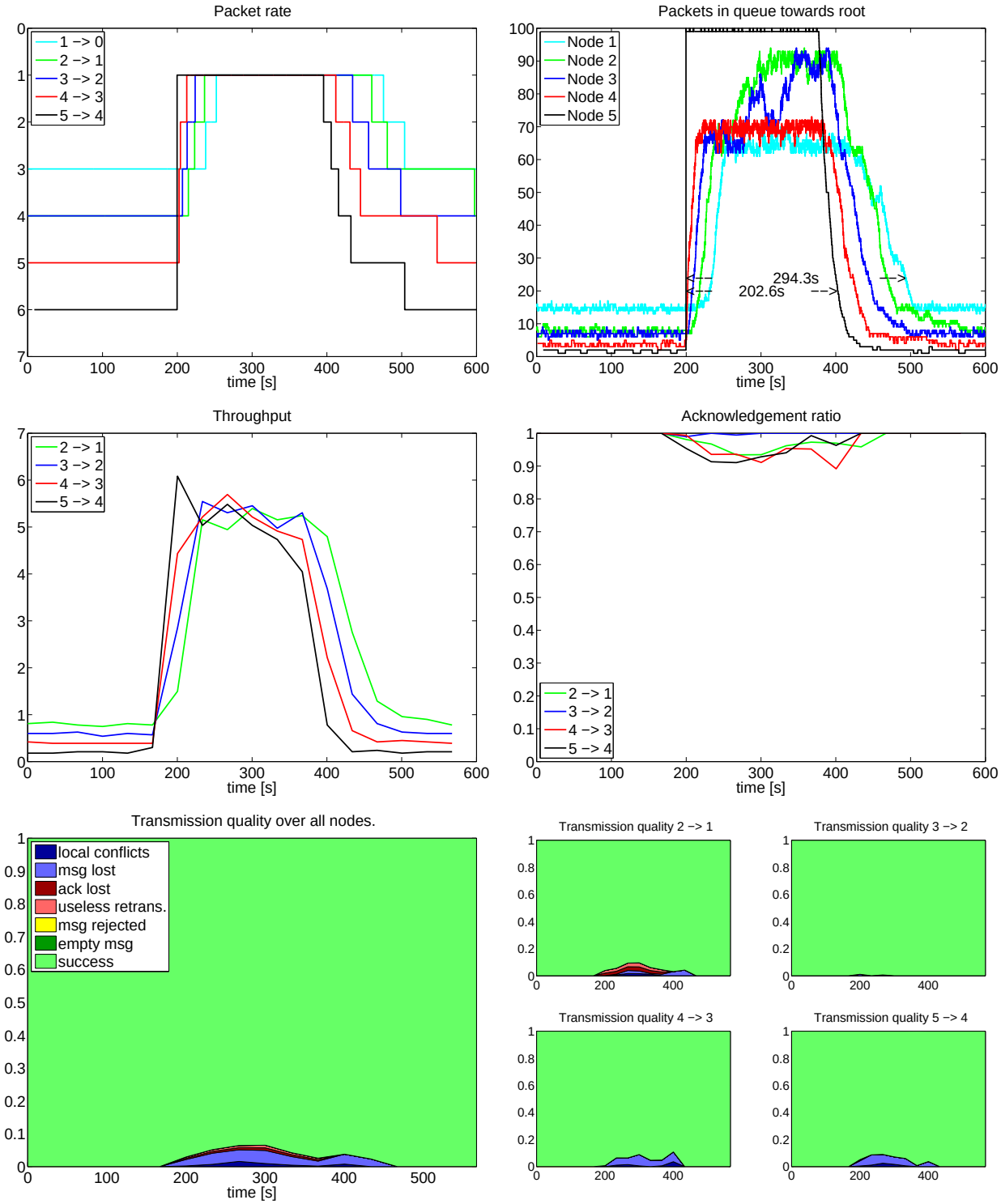

Figure A.7: 4 Frequencies, Rates 1 - 8

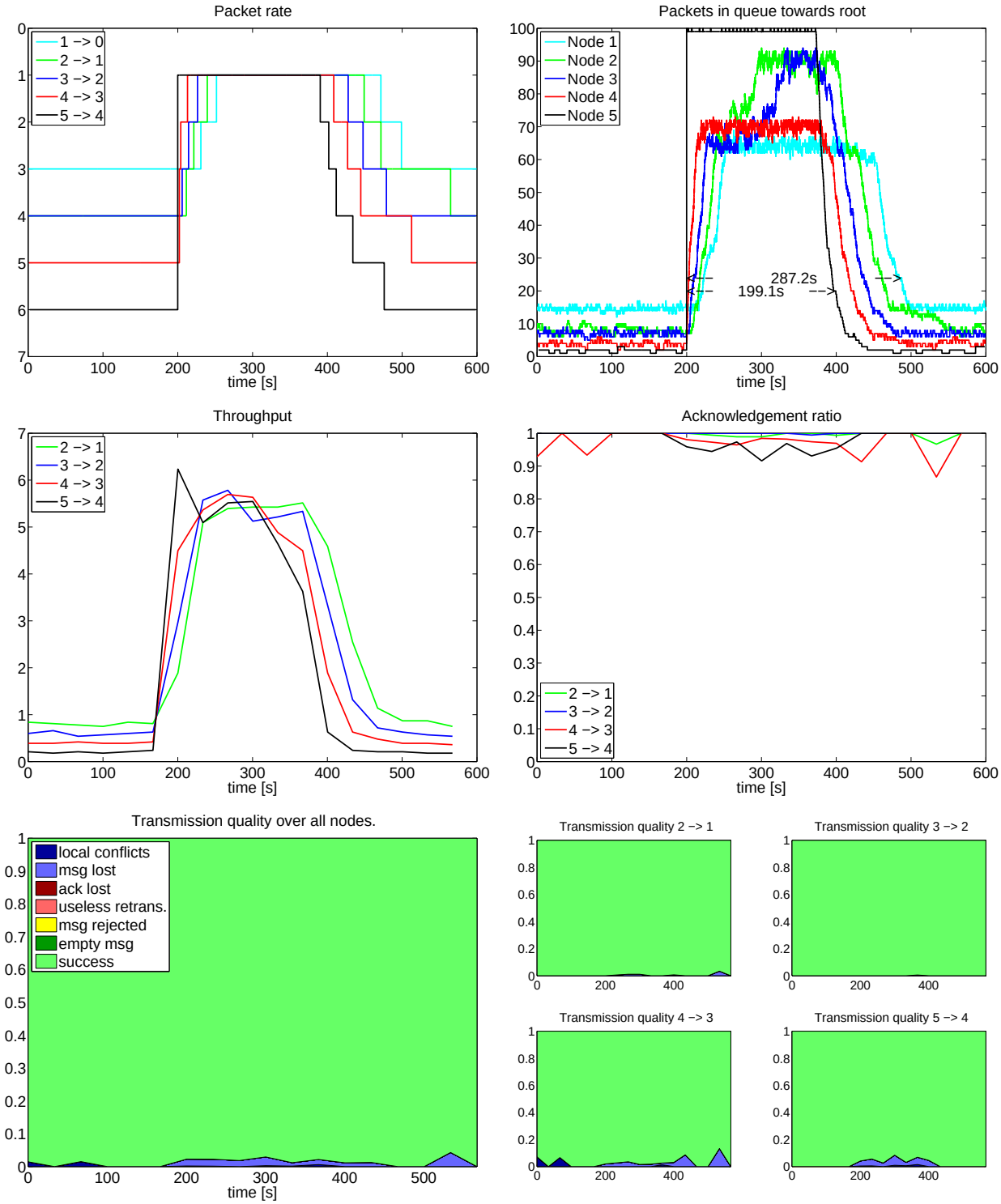

Figure A.8: 8 Frequencies, Rates 1 - 8

Bibliography

[1] B.P. Crow, I. Widjaja, L.G. Kim, and P.T. Sakai. Ieee 802.11 wireless local area networks. *Communications Magazine, IEEE*, 35(9):116–126, Sep 1997.

[2] I. Demirkol, C. Ersoy, and F. Alagoz. MAC Protocols for Wireless Sensor Networks: A Survey. *Communications Magazine, IEEE*, 44(4):115–121, April 2006.

[3] A. El-Hoiydi, J.-D. Decotignie, C. Enz, and E. Le Roux. Poster abstract: wisemac, an ultra low power mac protocol for the wisenet wireless sensor network. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 302–303, New York, NY, USA, 2003. ACM.

[4] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[5] NesC: A Programming Language for Deeply Networked Systems. http://nescc.sourceforge.net/, June 2009.

[6] J. Rousselot, A. El-Hoiydi, and J.-D. Decotignie. Low power medium access control protocols for wireless sensor networks. In *Wireless Conference, 2008. EW 2008. 14th European*, pages 1–5, June 2008.

[7] Slotted Programming for Sensor Nodes. Personal communication.

[8] TinyNode. http://www.tinynode.com/, June 2009.

[9] TinyNode 584 Embedded Wireless Network Node - Fact Sheet. http://www.tinynode.com/uploads/media/SH-TN584-103.pdf, June 2009.

[10] TinyOS Community Forum - An open-source OS for the networked sensor regime. http://www.tinyos.net/, June 2009.

[11] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the*

1st international conference on Embedded networked sensor systems, pages 171–180, New York, NY, USA, 2003. ACM.

[12] Wei Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576 vol.3, 2002.