# Debugging topology

In this simulation, I do not implement the wakeup phase.

I will simulate in the following topology for debugging.

Tier 1

Tier 2

Tier 3

6

Tier 0

A node listening, will receive all incoming packets. A node always listens until the end of the packet. That is, even if it expected a shorter packet, it will listen until the end of the transmission. It will process a packet’s meaning at the end of the reception.

# OrneMAC

## Phases

* Transmission phase
* Receive packets (from children and from higher priority siblings).
  + If for a long time a child does not provide a message, then we need to poll them. The reason for not replying may be:
    - It is waiting for children’s data, which may be unresponsive. A node’s delay propagates to all its ancestors.
    - it did not receive the Relay Offer Packet due to a collision. If this is the If it still does not reply, I eliminate it. Not that sending this message makes the simulation more complex, so instead of simulating it properly, I will just remove it from all the involved nodes.

## Aggregation

Cristescu's model: A packet's volume is R if it is not merged with another one, and it is r if it is merged with another one. This essentially penalizes leaves.

Packets contain an ID.

At the end of the day, the sink should receive a packet with all the contributing nodes.

n = 1:12;x=log2(n);stem(n,x+1),title('log2(n+1)'),xlabel('n')

# Timing

## Timestamp

The two functions I wrote are different in that DMAC is prepare to work during the quiescent period while ORNEMAC is not. ORNEMAC has four phases actually: steady, massive wakeup, tree building, and transmission. I have not implemented the two first, but there is no need. A bit of mathematical analysis will do.

My next move is to downgrade DMAC so that it only performs the transmission phase. When the subtree of the root is empty, I will stop the simulation and return the time it took to execute and the delay.

I expect to be making many calls to these two functions and compare their delay and energy. The topics I will cover will be:

* Number of hops
* Number of sources
* Number of packets

Timing problem

If a node has children but no grandchildren, it can easily compute the time it will take for them to transmit, assuming there are no collisions and no back off due to other nodes. It can compute the time it will take it to transmit to its parent and tell him. In this way, the time to get the data propagates through the tree. This could not work too well if a node has many cousins, uncles, etc.

The approach in (Solis & Obraczka 2004) considers periodic aggregation. I have made different assumptions. We want to ship the data as soon as possible.

For how long should a node wait before aggregating?

One possibility is that during the setup phase, each node reports to its parents for how long it expects to be receiving packets. When a node receives the information from all its children, it can compute the time it will take to gather all their data as the summation of the times. This estimate is not good.

Factors ignored that reduce the time:

* Concurrent transmissions are feasible.

Factors ignored that increase the time:

* Collisions
* Backoff times due to transmissions from nodes in upper tiers.

Therefore, a node will only fire its timer when it has sensed the channel idle for that amount of time.

Proposed technique

A node will only transmit to its parent when it has something to transmit and either it has received the data from ALL its children or It has sensed the channel idle for a time equal to the time to transmit through its subtree.

I know the setup period at the deepest tier started at t=0. Therefore the setup phase at the deepest period started at

### Simple simulation

When a node receives an acknowledgement of its parent about a data transmission, it sets a bit of all its children indicating that it is ready to receive the next packet. This is cheating, but it helps reduce the complexity. With this, we guarantee that every node will receive all the packets from all the nodes in each round.

Timing: A node enters the transmit state once the setup phase four tiers upwards has finished.

A node starts transmitting when the parentTransmitted flag is set to one and it has sensed the channel idle for a random time. I will forget about giving priority to siblings.

I want to model the time

When a node schedules a transmission, it also schedules the next retransmission. This is true only for packets that expect an acknowledgement. If the acknowledgement does not come,

# Timing

Let be the time sleeping time of the system.

Tx

Rx

Tx

Rx

Tx

Tier 3

Tier 2

Tier 1

Tx

Rx

Tx

Rx

Tx

Slot 1

Slot 2

Slot 3

Let be the number of tiers excluding the gateway. The period system will be.

The system

I will choose:

* . Therefore, a node picks a random back-off period between 0 and 4.
* DATA = 6
* ACK = 2

After a node successfully received a packet, it will transmit an acknowledgement and will relay a packet to the gateway (except if the node is in tier 0, i.e. it is the gateway).

It will transmit a new packet at the end of the current period.ssing

## Initialization

* Nodes with packets schedule StartCt events
  + For nodes in tier with data to report, the transmitting time is
* Schedule the StartLi event of all the nodes.
  + For nodes in tier, the first listening time is.
  + If

## Processing events

* Execution of a StartCt event
  + Set status to BO
  + Schedule a StartTx event a random time between 0 and CW later.
* Execution of a StartLi event
  + Set status to Li
  + Schedule a EndLi event later
* Execution of a StartTx event:
  + For all nodes in the same tier within the interference range
  + For all nodes within interference range
    - if it is in Li
      * It cancels its EndLi event
      * It sets the new StartLi
      * If it is not the intended destination
        + enter UNRESP state
      * If it is the intended destination
        + it enters into RX state
        + it schedules a SendAck event
    - If they are in BO
      * Cancel their StartTx event
      * Create a StartCt event
      * Set their state to UNRESP
    - If it is in RX % Collision
      * it cancels its SendAck
      * It enters into UNRESP state
      * Schedule a StartLi event
  + If the destination
    - is in Li:
    - is in RX: %Collision
* Execution of a EndRx event
  + The node
* Execution of a EndLi event
  + Set status to UNRESP
  + Schedule a StartLi in the next period

## Analyzing results

# DMAC's algorithm

Receive

Transmit

# Transmission of a packet

## Beginning of the transmission

### Impact on the intended recipient

When the leading edge of a packet reaches its destination:

* The destination sets the receiving flag.
* The destination schedules an "end-of-reception" event.

### Impact on interfering nodes

A transmission from A to B has a different effect in a node C within A's interference range:

### If C is backing off

C will delete its scheduled event.

C will schedule a transmission for the next period.

#### If C is sleeping and it has not scheduled to wake up before the end of A's transmission

We do nothing.

#### If C is receiving a packet from another node D

C will receive none of the transmissions from A and D. C reschedules an "end-of-reception" to

#### If C is transmitting a packet and it will finish transmitting before the end of A's transmission

C will record as ongoing interference the following:

* + Interference source
  + Time of end of interference.

Tier 1

Tier 2

Tier 0

5

Tier 3

### Impact on the transmitter

The transmitter schedules a Wait-For-ACK event.

## End of the transmission

# Transmission discussion

The transmission of a data packet and the transmission of the acknowledgement have a similar effect on all the nodes except the destination. Therefore, it makes sense not to duplicate the code and write a function for the packet transmission. The source of the packet will only have to set its new state, schedule a new event at the end of the transmission, and call this function.

## Transmit packet

* Input parameters
  + The packet
    - Source
    - destination
    - Packet type

A node will keep on receiving packets from the children with a high duty cycle if the last

# More to send packet (MTS packet)

There are three mechanisms to increase the duty cycle temporarily to transmit more data:

* The more data flag indicates the recipient that there are more packets to transmit.
* The data prediction mechanism is the following: nodes that received something always schedule the next reception. I think this approach is not robust. If one node fails to transmit a packet, its parent will not be listening in the next slot. Note that if we are applying data prediction there is no need to use the more data flag.
* The MTS packet with the MTS=1 is sent:
  + A node does not manage to transmit and does not receive the acknowledgement.

## Simulation platform

I have introduced a lot of complexity in my file.

I believe that I have created a good simulation platform:

* I define all the states that take part in the system
* I define all the events that change the state of the system

Introducing a new model would consist of:

* Writing the new state names
* Generating associated events
* Handle appropriately the new events
* Handle the reception of a packet in the new state
* Set the energy consumption of the new state.

## Simulation evolution (unrealistic)

We could represent each state could be represented by a new class. The class would have some properties such as:

* Duration of the state if nothing happens
* Handling of receiving packets.
* Energy consumption in that state

The advantage of this approach would be that each state would be defined at the same time, and we would have modularity.

## Scheduling

It will probably take me four more days to finish the DMAC simulation, that is, until 15 January. On 16 January I will see Kin. Therefore, it would be very good to have some results to show him and comment my progress.

Then I would have two weeks to simulate my algorithm.

# Description

## Simulation topology

5

4

3

1

2

%% Set timing parameters

% mu = FixedBackoffT + ContentionT + DataT + WaitB4TxAckT + AckT + MTS\_T;

% We need FixedBackoff > WaitB4Ack

I have decided that simulating the MTS packets is too much trouble. I would need a more systematic design, rather than my approach of start coding and gradually add features.

I will assume that the mechanism just works and that it involves little overhead. According to this:

## Receiver part

A node will always listen during the complete listening period, even after finishing acknowledging the reception of a packet. This is so because it must wait for the MTS packages. In my simulations, I do not use MTS, but the energy consumption should be similar. Therefore, I leave the radio transceiver on.

At the beginning of a listening period, a node will check if any of the nodes in its subtree have packets in their input buffer. If they do, the node set OneMore = 2. If they do not, the node decrements OneMore.

After the end of a listening period:

If (OneMore > 0) it will schedule an extended listening period 4µ later.

Otherwise, it will schedule the next natural listening.

## Transmitter part

A node will listen during the complete listening period, except when it is transmitting. This is so because a node has to be able to receive any packets. For example, in (Lu, Krishnamachari, & Raghavendra 2004), a node that wanted to transmit but lost the contention will not send an MTS if it overheard an ACK packet from its parent.

If a node has data, it will schedule transmission in the next increased transmission period. If a node has no

# MTS functioning

Backing-Off state

If when the back-off period expires the channel is idle:

If there is enough time to transmit a packet

It enters into data transmission state.

It sends the packet.

If there is not enough time to transmit the packet it simply transmits a MTS packet.

If when the backoff period the channel is busy initi

Extend the backoff period.

If while backing off it receives a packet (the first packet it receives) :

Put it into the input buffer.

Cancel the transmission event for the node.

Schedule end of reception event.

At the end of the reception:

If the reception was successful

Check the content of the packet.

If it is an acknowledgement from the parent, set TransmitMTS to false.

Remove the packet from the input buffer.

Clean reception parameters.

Receiving state

When a packet arrives:

Put it into the buffer

Do NOT remove the end of listening event.

At the end of the transmission

If it is unsuccessful, remove the packet.

If it is successful,

If it is a Data packet for itself:

put it into the output buffer

Reply with an Ack

If it is a MTS packet for itself:

If it is a MTS request packet, set MTS to initialMTSvalue

If it is a MTS clear packet, set MTS to 0.

Go on listening.

Data transmission state

Simplification

I have decided that simulating the MTS packets is too much trouble. I would need a more systematic design, rather than my approach of start coding and gradually add features.

I will assume that the mechanism just works and that it involves little overhead. According to this:

Receiver part

A node will always listen during the complete listening period, even after finishing acknowledging the reception of a packet. This is so because it must wait for the MTS packages. In my simulations, I do not use MTS, but the energy consumption should be similar. Therefore, I leave the radio transceiver on.

At the beginning of a listening period, a node will check if any of the nodes in its subtree have packets in their input buffer. If they do, the node set OneMore = 2. If they do not, the node decrements OneMore.

After the end of a listening period:

If (OneMore > 0) it will schedule an extended listening period 4µ later.

Otherwise, it will schedule the next natural listening.

Transmitter part

A node will listen during the complete listening period, except when it is transmitting. This is so because a node has to be able to receive any packets. For example, in (Lu, Krishnamachari, and Raghavendra), a node that wanted to transmit but lost the contention will not send an MTS if it overheard an ACK packet from its parent.

If a node has data, it will schedule transmission in the next increased transmission period. If a node has no

Sequence

Listening

When startWaiting4Data:

Check if any of the nodes in its subtree have packets in their input buffer.

If they do, the node set OneMore = 2. If they do not, the node decrements OneMore.

When packet arrives:

enter Receiving Data State, and stay in that state for

If while receiving, interference arrives, set ruinedReception = 1;

When endDataRx:

If the node received a data packet correctly:

Enter into idle state for WaitB4TxAckT and schedule a startAckTx event.

If the node did not receive a data packet correctly

Enter Idle State, in which the node does not transmit or receive,

but its energy consumption is that of Idle. Do not schedule anything,

as the endWaiting4Data is still in the event list.

When endWaiting4Data, a node will

Schedule a startContending if its output buffer is full.

Schedule a startWaiting4Data in the next natural listening time if

Transmitting

startContending

Schedule a startDataTx

Schedule an endContending event

If data arrives while contending

Enter idle state and stay that way until the endContending event.

startDataTx

Enter into TransmittingDataState and schedule an startWaiting4Ack event

For all nodes within interference range

...

startWaiting4Ack

Enter Waiting4AckState

If a packet arrives while Waiting4AckState

Enter into ReceivingAckState and schedule an endAckRx event

endContending

Reference List

Lu, G., B. Krishnamachari, and C. S. Raghavendra. "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks." Proceedings of the IEEE 18th International Parallel and Distributed Processing Symposium (IPDPS '04). 2004. 224.

# Future work

The topology module

# Comparison with my algorithm

My algorithm has more states:

* There are three phases in the algorithm.
* In each phase

Reference List

Lu, G., B. Krishnamachari, and C. S. Raghavendra. "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks." Proceedings of the IEEE 18th International Parallel and Distributed Processing Symposium (IPDPS '04). 2004. 224.

References

Lu, G., Krishnamachari, B., & Raghavendra, C. S. 2004, "An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks," in *Proceedings of the IEEE 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, p. 224.

Solis, I. & Obraczka, K. 2004, "The impact of timing in data aggregation for sensor networks," in *Communications, 2004 IEEE International Conference on*, 6 edn, pp. 3640-3645.