# 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

# Flow control

When a node transmits a packet, it schedules the retransmission some time afterwards. If it receives the acknowledgement, it cancels the retransmission, and it requests all its children the next round.

Note that the sink does not receive acknowledgements from its parents. Therefore, no one requests the next round. To solve this problem, I propose the following:

When the sink has aggregated all packets from its children, instead of sending an acknowledgement it executes the request(pktNr). That makes all the children prepare for transmitting the next round. They may need to

# Algorithm realistic

* When a node receives DATA
  + If pktNr == expected, send ACK
  + If pktNr < expected, send them a Next Packet Request (NPR)
* When a node receives an ACK
  + AckRx code:
    - Remove packet from output buffer and the corresponding event (we know for sure there will be both).
    - If it does not have any children but it has some packets in BOwn, place the packet in the output buffer and schedule transmission.
    - If it has children, send them a NPR.
    - I schedule a resend NPR event
* When a node receives a NPR from parent with demanded
  + If in output buffer pktNr < expected, execute AckRx code
    - Note that if several children had unacknowledged packets, they will collide.
  + Else, I have necessarily already executed the AckRx code, therefore I do not need to do anything else

# Simplest algorithm

* When x receives a packet from y
  + Remove packet from the source’s output buffer
  + Remove the event from the source that retransmits the packet.
  + Acknowledged(y)++
  + If all the packets are there
    - Aggregate them and place them in the output buffer
    - Expected(x) ++
    - For z = x or chld{x}
      * If expected(prnt(z)) > acknowledged(x) && tier(z)
        + If (no inPktTx && not sleeping && no transmitting) && ~isempty(BOut{z})

Schedule inPktTx event

# To do plans

The main things to do are:

* Compare the performance of
  + fat + txPhase.m
  + dijkstra + txPhase.m
  + dijkstra + dmac.m (note that there is no setup phase)
  + steiner.m + txPhase.m
* Compare my algorithm to the once recently published (Zhu et al. 2005). Scalability was a design objective of this protocol, therefore a fair comparison would require simulating many nodes. My Matlab code is probably not fast enough to simulate that. I am concerned that the performance of my implementation of this protocol would be below the one from

I have decided not to force the generated set of nodes to be a connected set. This is not necessary if I put data in connected nodes. In my code, I make sure that there are at least as many connected nodes as the desired number of sources. When I build the tree using Dijkstra’s algorithm or my approximation of Steiner tree, I grow the tree from the root to the leaves. This process will never include a disconnected node.

My algorithm has taken shape. The optimal algorithm will be similar, but with the following differences:

* It does not involve tree construction overhead.
* The tree constructed with the pseudo-optimal algorithm should perform better.

The question arises whether as existing work I should use my transmission algorithm. It would not be fair that those algorithms already assumed that the nodes knew without cost from how many children they have to wait. It would not be fair because I would be giving a considerable advantage to those algorithms over mine. I am not aware of how existing algorithms to gather this knowledge. I think it is best to assume that SPT involves a similar overhead in making each node aware of the children as my algorithm in constructing the tree. This way we make sure that my algorithm outperforms the one I label as SPT.

Alternatively, I could use SPT as follows. Each node waits for a time equal to the maximum number of children in the tree if all nodes had data. This method makes use of timers. Note that it is possible that due to collisions a node will not aggregate the packets from all its children and then will relay the remaining packets with less aggregation than possible. This is not difficult to implement. I would just need some timers. If my protocol does not perform well enough I could use this technique as a benchmark. Although which technique is better depends may depend on the number of packets. With this technique, all nodes would have to stay awake therefore for networks with many more nodes than those that detect the event, the energy consumption would be much higher.

When I compare FAT with the protocol based on the Steiner tree, I must decide carefully which nodes remain active.

FAT incorporates a setup phase. This entails overhead, but it also means that each node knows how many children and siblings it has. If a node has neither data nor children, it can go to sleep. If there are many such nodes, this can save a significant amount of energy. Note that in DMAC such nodes enter fast into sleep mode, so FAT does not offer any advantage in this respect.

Let us recapitulate what we plan to do. Now I need to plan the actions over the next two weeks. Every action has to be explained in that frame.

* I would like to turn the packets into objects.

# Main drawbacks of current paper

1. There is no comparison with protocols using aggregation. Actions:
   1. Compare with Shortest Path Tree
   2. Compare with an approximation of the optimal solution: Steiner tree
2. The energy figures look terrible. If there are going to be two figures in the paper, is a very bad idea that one of them reflects that my algorithm is much worse than existing ones. Actions:
   1. Create new graphs for them
3. It is difficult to sell my algorithm as something that works fine for transients but does not work well when we need to send more than a couple of packets. Therefore, I should present them otherwise. I will present my algorithm as a decentralized mechanism that provides a good approximation.

# Temporal planning

I have two weeks. I will reserve the last week in case an emergency occurs. That leaves me with a week. By Sunday 17 February I should have some interesting graphs comparing the quality of the tree obtained with the quality of my pseudo-optimal.

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.

Zhu, Y., Ramanuja, V., Park, S. J., & Raghupathy, S. 2005, "A scalable correlation aware aggregation strategy for wireless sensor networks," in *Proceedings of the First International Conference on Wireless Internet*, pp. 122-129.