EMBS

[redacted]

January 2019

1 Theory

1.1

The first step is to build the topology matrix Γ representing each actor on each column and each connection on each row.

$$\begin{array}{c} \begin{matrix} A & B & C & D & E & F & G & H & I \end{matrix} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & X \\ 0 & 0 & 0 & 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 & -2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \\ c7 \\ c8 \\ c9 \end{matrix}$$

Secondly we need to multiply Γ by the column vector $\mathbf{q}$ containing the fire frequencies of each actor. By setting the resulting equations equal to zero we can determine the fire frequencies values.

$$\begin{pmatrix} q_a \\ q_b \\ q_c \\ q_d \\ q_e \\ q_f \\ q_g \\ q_h \\ q_i \end{pmatrix}$$

We obtain the following system of equations and we need to find solutions for the smallest values of the firing frequencies.

$$\begin{pmatrix} q_a - q_b = 0 \\ -q_b + q_c = 0 \\ q_b - 4q_d = 0 \\ q_c + Xq_i = 0 \\ 2q_d - q_f = 0 \\ q_d - 2q_i = 0 \\ -2q_e + 2q_f = 0 \\ -q_g + 3q_i \\ -q_g + h_h = 0 \end{pmatrix}$$

We can derive the following equalities.

$$\begin{pmatrix} q_a = q_b = q_c \\ q_d = \frac{q_a}{4} = \frac{q_b}{4} = \frac{q_c}{4} \\ q_e = q_f = 2q_d \\ q_i = \frac{q_d}{2} = \frac{q_a}{8} = \frac{q_b}{8} = \frac{q_c}{8} \\ q_g = q_h = 3q_i \end{pmatrix}$$

The following assignments can be inferred from the previous equalities and we can prove they are a solution to the set of equations.

$$\begin{pmatrix} q_a = 8 \\ q_b = 8 \\ q_c = 8 \\ q_d = 2 \\ q_e = 4 \\ q_f = 4 \\ q_g = 3 \\ q_h = 3 \\ q_i = 1 \end{pmatrix}$$

We can use the assignments above to solve $q_c + Xq_i = 8 + X1$.
Therefore $X = 8$.

1.2

Since actor I needs 8 tokens from c4 to fire, if a PASS schedule exists, we must assume that the FIFO buffer with more than 4 positions is on c4.
The following is a PASS schedule.
a.fire(4); c.fire(4); b.fire(4); d.fire(1); f.fire(2); e.fire(2); a.fire(4); c.fire(4); b.fire(4); d.fire(1); f.fire(2); e.fire(2); i.fire(1); h.fire(3); g.fire(3);

1.3

A schedule that minimizes context switches is the single appearance schedule, where each actor fires consecutively as many time as possible.
If we ignore any buffer restriction the following is a single appearance schedule.
a.fire(8); c.fire(8); b.fire(8); d.fire(2); f.fire(4); e.fire(4); i.fire(1); h.fire(3); g.fire(3);

2 Design exercise, WSN MAC layer protocol

2.1 Ptolemy

My solution consists of 3 three actors.

A MainActor receives incoming beacons and uses time arrival and beacon value to estimate the parameters T and N of the sinks (MainActor, fire, 97 - 174).

A SendActor is responsible for sending frames to sinks (SendActor, fire, 45 - 65) and schedule such sending events (SendActor , scheduleCompletely, 91 - 102).

A ChannelActor is responsible for switching between channels (ChannelActor, fire, 75 – 85) for the purpose of receiving beacons (ChannelActor, switchChannelForReception, 108 - 111) and for the purpose of sending frames (ChannelActor, switchChannelForSending, 120 – 124).

The MainActor is substantially a state machine that follows the trasitiond illustrated in Figure 1 , the simplest way it has of finding the value of T is observing two beacons in the same SYNC phase (MainActor, calculateTWithSuccessiveTokens, 234 -237). It can also determine it by observing a beacon of value one and then the first beacon of the next SYNC phase (MainActor, calculateTWithDivisionBy12, 245 -247). Finding N requires observing the first beacon of the SYNC phase (MainActor, fire, 118 -119), it determines this fact by knowing how long it has been waiting for a beacon after switching to the channel (MainActor, fire, 118).

The SendActor keeps track to what channel it needs to send a frame by using a map of channels and time instants (SendActor, timeToChannels, 25), it calls the ChannelActor to switch channel if the current channel is not the target one (SendActor, fire, 52 – 60). In order to switch back to the previous channel the ChannelActor fire itself some instant after the SendActor has performed the send (ChannelActor , switchChannelForSending, 124). Switching back is performed in order to continue listening to a channel for which parameters have not been estimated yet.

2.2 Mote Runner

My solution includes a Source class which is the main class, a Scheduler and a MyUtils class. Whereas in Ptolemy I tried to calculate both T and N for a sink and schedule everything upfront, here in Moterunner I adopted a different and

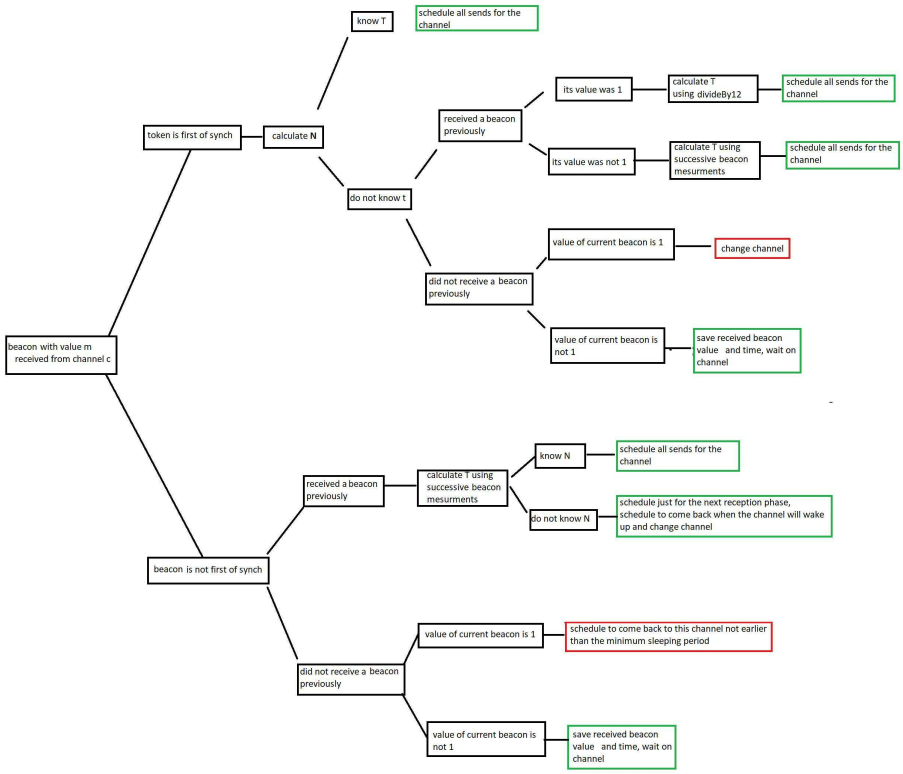

Figure 1: From left to right, the transitions taken by MainActor

simpler approach. This is motivated by the the disrupting effects that clock drift has on time calculations too far into the future. Also the fact that the minimum value of N is now 2 has eliminated much of the complicated logic for edge cases. Compare Figure 2 describing the transitions in the Mote Runner code with Figure 1.

All Source is concerned with is estimating T (Source, onReceiveUnkownT, 79 - 87) and then scheduling the upcoming reception phase using any beacon value with the formula below (Source, scheduleAndIterate, 106 - 107).

$$Middle of Receptionphase = (BeaconValue * T) + \frac{T}{2}$$

Once T is known, Source will schedule the next reception phase as soon as it gets a beacon (Source, onReceiveKnownT, 99 -100). The method iterateChannel switches to a new channel after we have scheduled the next reception phase. This is done by taking energy efficiency into account: if all channels have been scheduled we know we will not need to read any beacon for a certain period,

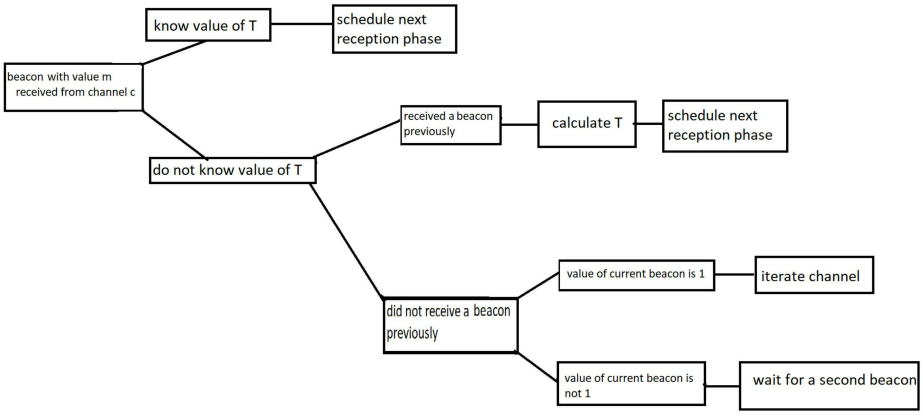

Figure 2: Transitions in the Moterunner code, read left to right

therefore we can turn off receiving (Source, iterateChannel, 180) which will spare energy. The Moterunner documentaion states: "The OS keeps devices in states with the lowest energy consumption without applications having to care about the involved state changes. [...] Applications specify tasks, optionally with a time of desired execution, and the OS keeps the device in the lowest state and only ramps up it up in order to run the tasks." Therefore by making explicit that we do not need the radio in the reception phase and by specifying, similar to how my switch channel mechanism does, at what time we want the radio to turn on reception (Source, scheduleChannelSwitch, 138), we put the OS in a good position to operate the radio at the lowest power consumption mode possible.

The Source schedule a switch to a channel roughly when the channels enters the SYNC phase (Source, scheduleChannelSwitch, 121 - 125). When it is determined that we can not schedule a channel switch (Scheduler, canChannelSwitchBeScheduled, 25 -26) and (Scheduler, canBeScheduled, 36 -43) we add an offset based on T and try again, if we fail we do not schedule the channel switch (Source, scheduleChannelSwitch, 128 - 134).

In respect to this, an improvement could have been keeping track of the minimum value of N for each channel, (i.e. the maximum beacon value). With that information we can determine how many T offset we can safely add to a wake-up time that is not schedulable. Figure 3 illustrates this; the scenario at he bottom has more T offsets that can be safely added and guarantee that we will see a beacon before reception.

Another improvement could have been estimating the clock drift and taking it into account. If the clock drift was constant I could have attempted scheduling into the future by also calculating N. This could potentially have lead to not needing listening to the sinks anymore, because all the reception phases would have been known once calculated the parameters. This would be a great advantage in terms of energy efficiency.

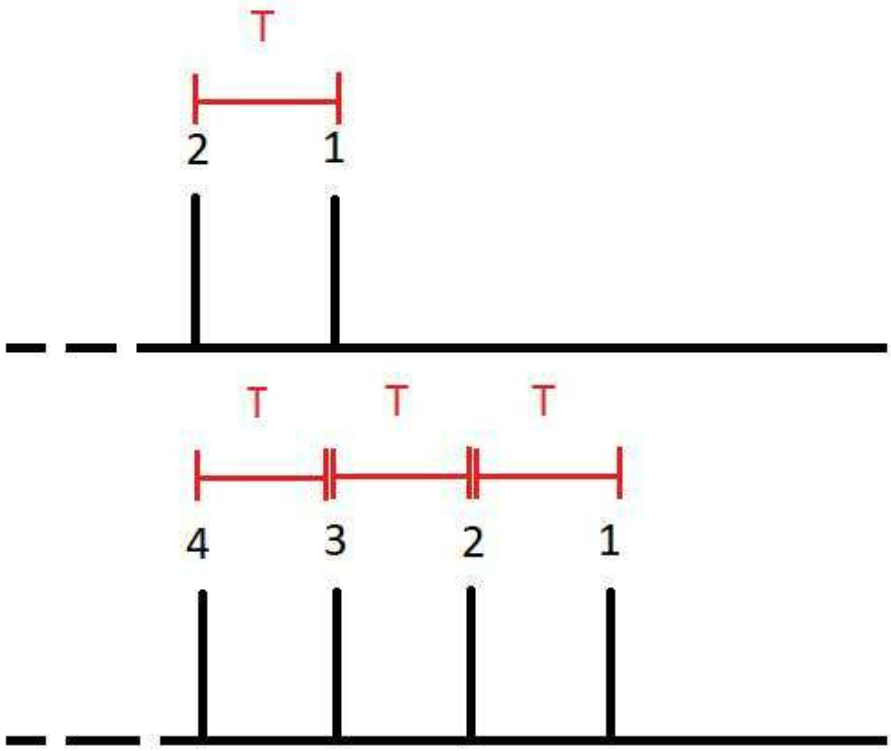

Figure 3: channel with higher N has a higher "schedulability slack"

3 Design Exercise, Embedded platform modelling

3.1

In the first table no write instruction generate a cache hit, but when a write instruction is executed it also writes the data in the cache, effectively making it available to other instructions. The second table assumes write-back and write-allocate. We can see how the number of write-hit between the second and first table is the same. The third table assumes write-through and write-around. Here the values are slightly different. Write-hit policies: Write-through writes to both main-memory and cache, slower but more consistent. Write-back consists in writing only to cache, faster but requires persisting the data before it is overwritten.

Write policies are important for data consistency: Write-miss policies: Write-allocate loads the missing address into cache and writes to both cache and main memory. Write-around writes to the missing address only in main memory.

Another assumption is that the local memory sizes do not take into account the memory needed to store the address. If I did not make such assumption, I would have considered each size for each scenario as halved, this is because the

No write policies, a wrtie just loads the address to cache

	128	512	1024
PE1	72	90	90
PE2	18	154	154
PE3	104	117	117
PE4	290	309	310

Write back and write allocate

		128	512	1024
PE1	read hit	72	90	90
	write hit	24	30	30
PE2	read hit	18	154	154
	write hit	5	35	35
PE3	read hit	104	117	117
	write hit	40	45	45
PE4	read hit	290	309	311
	write hit	103	110	110

Write through / around

		128	512	1024
PE1	read hit	72	90	90
	write hit	0	0	0
PE2	read hit	18	154	154
	write hit	0	0	0
PE3	read hit	104	117	117
	write hit	0	0	0
PE4	read hit	289	306	308
	write hit	1	3	0

Figure 4: cache-hits tables ,from the top: no write-policy, write-back/allocate, write through/around

address size is 16 bits.

In local memory, dynamic energy is dissipated every time the cache transistors open or close, it is therefore proportional to the number of cache hits.

Even when the cache is not used, static energy dissipation occurs as a result of current leakage. The effect of adding local memory units should not be consider alone but in the context of the whole system, static dissipation of other components could change. Charging and discharging capacitance in relation to neighbour wires should be taken into account. Local memory could also reduce the time the PEs are driving the bus wires with a request signal; a cache hit would eliminate the need of using the bus arbitration system.

The advantages of having local memory is a potential increase in performance and a potential reduction of energy utilization. It would depend on the application that is being run and on the energy-dissipation of the cache compared to using the bus and accessing the memory.