Practicum 2 Synchronisatie bij vuurvliegjes

Peter van Dijk & Elizabeth Schermerhorn

1 juli 2014

Inhoudsopgave

1 Inleiding

In de natuur zijn er verschillende vormen van synchronisatie. Aangezien synchronisatie een belangrijk punt is in de werking tussen verschillende nodes in een netwerk is het van belang om een goed algoritme hiervoor te hebben. Een bekende uit de natuur zijn vuurvliegjes die met elkaar synchroniseren wat zich uit in samen knipperen. Dit principe dat vuurvliegjes synchroon kunnen knipperen wordt geïmplementeerd in een multihop netwerk. Door een algoritme te ontwikkelen moet het mogelijk zijn om nodes tegelijk te laten knipperen net zoals een groep vuurvliegjes.

2 Probleemstelling

De probleemstelling die gesteld wordt is: *"het synchroniseren van nodes in een multihop netwerk met als doel ze tegelijk te laten knipperen."*. Om dit netwerk te ontwikkelen moet er gekeken worden naar de eisen wat betreft omgeving en precieze werking. Een aantal eisen waar het netwerk aan moet voldoen:

- Wanneer twee gesynchroniseerde netwerken worden samengevoegd dienen ze te synchroniseren.
- Wanneer een node uitvalt dient de synchronisatie nog te werken.
- Wanneer nodes toegevoegd worden aan het netwerk gaan deze met dezelfde frequentie knipperen als de andere nodes in het netwerk.
- Wanneer nodes uit sync raken moeten ze opnieuw synchroniseren.

Nu is vastgesteld aan welke eisen het netwerk moet voldoen kan er een hypothese opgesteld worden. Om ervoor te zorgen dat nieuwe nodes met dezelfde frequentie gaan knipperen als nodes die al langer in het netwerk verkeren zal er om de zoveel tijd opnieuw met alle nodes gesynchroniseerd moeten worden. Door dit in de implementatie toe te voegen zou dit ook moeten werken. Dit geldt ook voor de drie resterende bovengenoemde punten.

3 Gerelateerd werk

Er zijn verschillende onderzoeken gedaan naar het onderwerp synchronisatie in een multihop netwerk. Een paar papers welke dit onderwerp aansnijden zijn:

- Clock synchronization for wireless sensor networks: a survey
- Wireless sensor network survey
- Academic Press Library in Signal Processing, Chapter 2 Synchronization

De bovenstaande drie papers gaan over synchronisatie in draadloze netwerken. De eerste paper - Clock synchronization for wireless sensor networks: a survey - evalueert bestaande kloksynchronisatie algoritmen gebaseerd op factoren zoals precisie, complexiteit, nauwkeurigheid en kosten. De tweede paper, Wireless sensor network, geeft een overzicht van bestaande draadloze netwerk implementaties op verschillende niveaus en als laatste geeft het boek Academic Press Library in Signal Processing een kort overzicht van tijd synchronisatie in een netwerk.

In Sectie 3.3 van de paper "Clock synchronization for wireless sensor networks: a survey"wordt gesproken over de verschillende mogelijkheden bij de implementatie en hoe deze in hun werk gaan. Wanneer er een draadloos sensor netwerk ontwikkeld wordt zijn er een aantal factoren waar rekening mee gehouden moet worden. Hieronder staat een lijst weergegeven met de verschillende design-keuzes die gemaakt moeten worden.

- Master-slave protocol versus peer-to-peer synchronisatie
- Interne synchronisatie versus externe synchronisatie
- Single-hop netwerk versus multihop netwerk
- probabilistische versus deterministische synchronisatie

In Sectie 3.3 van de paper "Clock synchronization for wireless sensor networks: a survey"staan nog meer keuzemogelijkheden, deze zijn niet van belang voor dit experiment en zijn voor het gemak weggelaten. In Appendix B staat kort beschreven wat elke implementatie mogelijkheid inhoudt.

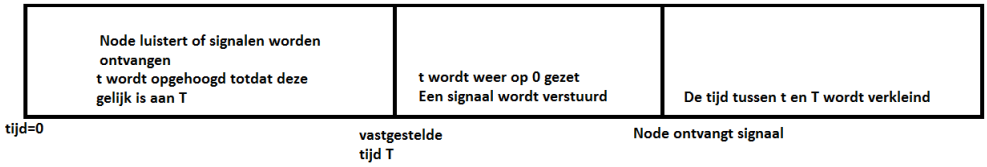
Afhankelijk van de eisen die aan het netwerk gesteld worden moet er voor bepaalde onderdelen gekozen worden. Eisen aan het netwerk zouden kunnen zijn: Er is geen server beschikbaar, het moet zo min mogelijk energie verbruiken zodat de nodes lang meegaan of de nodes staan ver uit elkaar. Dit zijn slechts een aantal voorbeelden.

In de tweede genoemde paper, Wireless sensor network survey, worden in sectie 6.2 meerdere synchronisatie algoritmen uitgelegd. De volgende algoritmen worden beschreven:

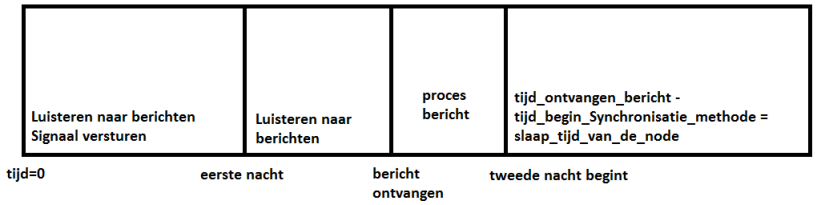
- Onzekerheid gedreven benadering

- Lucarelli's algoritme
- Vuurvlieg algoritme
- Tijd synchronisatie protocol voor sensor netwerken
- klok-opname gezamenlijke netwerk synchronisatie
- Tijd synchronisatie
- globale synchronisatie synchronisatie

Het geïmplementeerde algoritme wat in deze paper besproken wordt is gebaseerd op het vuurvlieg algoritme. Hoewel het geïmplementeerde algoritme erop gebaseerd is, is het op sommige punten aangepast. Het beschreven algoritme in de paper werkt als volgt: Elke node in het netwerk werkt als een oscillator met een vastgestelde tijd T. Elke node heeft een interne tijd t die opgehoogd wordt totdat deze gelijk is aan T. Op tijd T zal de node een signaal sturen en zijn interne tijd t weer op 0 zetten. Buren van deze node die dit signaal ontvangen zullen de tijd tussen t en T verkleinen. Dit wordt bepaald door middel van een functie en een kleine constante. Na verloop van tijd zullen de nodes gelijk hun signaal versturen dus gesynchroniseerd zijn. in figuur 1 is dit algoritme schematisch weergegeven. De bedoeling is dat alle nodes tegelijk een signaal gaan versturen na verloop van tijd.



Figuur 1: Het vuurvliegalgoritme zoals beschreven in de paper



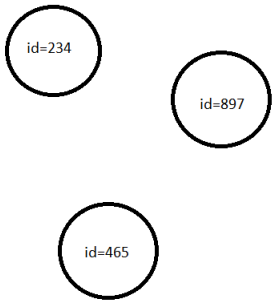
Figuur 2: Het geïmplementeerde algoritme

Het grote verschil tussen bovenstaande implementatie en de experimentele implementatie van deze paper is dat de t niet naar 0 wordt gezet maar het verschil tussen het luisteren naar berichten en het uitlezen van de berichten

wordt bijgehouden. Op basis van deze tijd wordt er voor een bepaalde tijd geslapen waardoor er een vertraging optreedt. Door dit fenomeen synchroniseren na verloop van tijd de nodes in het netwerk. 2

4 Ontwerp synchronisatiealgoritme

Het synchronisatie algoritme is gebaseerd op een master-slave structuur. In afbeelding 3 staat een schematische weergave van het netwerk zoals deze is gebruikt. Er zijn ID's meegegeven aan de nodes zodat er te zien is dat er één master is met één of meerdere slaves. De master van het netwerk wordt bepaald



Figuur 3: Schematische weergave van de testopstelling

aan de hand van een random nummer tussen 0 en 1000. Er is gekozen voor deze structuur aangezien er gesynchroniseerd moet worden. Het doel is om de nodes tegelijk te laten knipperen, om dit te laten werken moet de interne klok van de nodes exact gelijk lopen of in dit geval stuurt de master naar alle andere nodes zijn tijd zodat daarop gecorrigeerd kan worden. Binnen deze master-slave structuur is de implementatie verdeeld in twee alternerende delen. In het eerste deel wordt vijf maal achtereen alternerend de methode delay() aangeroepen en vervolgens een bericht verstuurd(De struct die hiervoor gebruikt is staat in de bijlage). In het tweede deel wordt er gesynchroniseerd, eerst wordt er voor 1000 milliseconden geluisterd of er berichten zijn ontvangen. Als er berichten zijn ontvangen worden deze verwerkt. Als laatste wordt er voor 1000 milliseconden geslapen en als laatste wordt er een bericht gestuurd.

Eerst zullen de gebruikte instellingen samen met het verstuurde signaal besproken worden. Daarna zullen de twee delen waaruit de implementatie bestaat toegelicht worden.

4.1 Gebruikte instellingen

Omdat er gebruik wordt gemaakt van de RF24 radio om berichten te versturen zijn hier instellingen voor gebruikt. Instellingen gebruikt voor de RF24 radio:

- radio.setRetries(0,0);
- radio.setPayloadSize(sizeof(MESSAGE));
- radio.setAutoAck(false);

In de radio.setPayloadSize(sizeof(MESSAGE)) methode wordt er gebruik gemaakt van de grootte van MESSAGE. Dit is een zelf gedefinieerd struct wat in de header file is toegevoegd. Door dit struct te versturen kan er bij het ontvangen van een bericht dit snel uitgelezen worden zonder delimiters te moeten gebruiken. Deze struct bevat drie delen: ID van de zendernode, id van de tot dan toe hoogst bekende node en het aantal verstuurde berichten. Er is gekozen om het aantal berichten bij te houden zodat wanneer er oude berichten worden ontvangen, die bijvoorbeeld een delay hebben opgelopen, niet gebruikt worden om op te synchroniseren. De hoogste ID is nodig om te beslissen of een bepaalde node de master node is en of daarnaar geluisterd moet worden. Als laatste is de ID van de Sender toegevoegd omdat dit de hoogste ID zou kunnen zijn in het netwerk. De instelling radio.setAutoAck(false); zorgt ervoor dat er geen acknowledgement wordt verstuurd op het moment dat er een bericht binnen komt. Dit is gedaan omdat er anders niet gesynchroniseerd kan worden, er zou op willekeurige momenten een bericht verstuurd worden. De laatste instelling, radio.setRetries(0,0);, houdt in dat wanneer het niet lukt op de vastgestelde tijd een bericht te sturen er geen hertransmissie plaats vindt. Anders zou een andere node die dit bericht ontvangt op het verkeerde moment gaan synchroniseren. Wat betreft de overige instellingen, is er gebruik gemaakt van de standaard instellingen van de arduino UNO en RF24.

4.2 Het eerste deel, slapen en versturen

Het eerste deel bestaat uit alternerend slapen en berichten versturen.

```
int sendMax = 5;
int i = 0;
while(i<sendMax)
        sleepFor(1000)
        toggleLed();
        sendMessage();
        i++
```

De bovenstaande pseudo code wordt vijf maal herhaald voordat er door wordt gegaan met het tweede deel. Doordat er alleen berichten worden verstuurd over een lange periode is er een grote kans dat andere nodes de verzonden berichten ontvangen. Door voor vijf berichten te kiezen wordt deze kans alleen maar groter.

4.3 Het tweede deel, synchroniseren

Het tweede deel gaat over het synchroniseren van de nodes om ze tegelijk te laten knipperen. De synchronisatie gebeurt door de volgende acties:

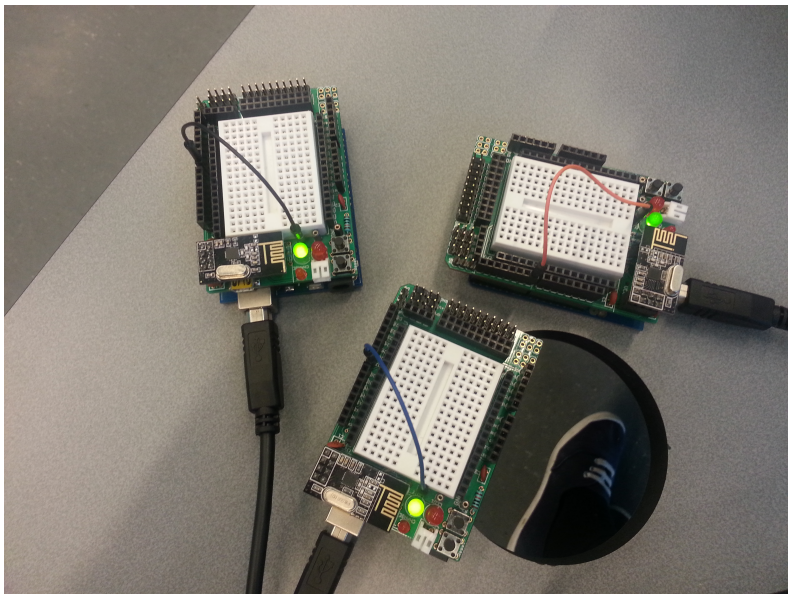
```
listenFor(deltT)       //deltaT = Tijd tussen luisteren en uitlezen bericht
if(message_received){
        toggleLed();
```

```
} else {
        toggleLed();
        sendMessage();
}
sleepFor(1000 milliseconds);
toggleLed();
sendMessage();
```

In bovenstaande code zal er eerst voor deltaT tijd geluisterd worden of er berichten worden ontvangen. De tijd deltaT is de essentie van de synchronisatie en is de tijd die nodig is tussen het luisteren voor berichten en het daadwerkelijk verwerken van de ontvangen berichten. door zo lang te luisteren zullen alle nodes uiteindelijk synchroniseren. De berichten die van de master node komen en een hoger bericht nummer dan het laatst bekende zullen alleen verwerkt worden. Op het moment dat er een bericht is ontvangen zal alleen het LEDje getoggled worden. Anders zal er ook een nieuw bericht uitgezonden worden. Door hierna voor 1000 milliseconden te slapen blijft het LEDje of aan voor 1000 milliseconden of blijft uit voor 1000 milliseconden. Hierna zal opnieuw het LEDje getoggled worden en zal er weer een nieuw bericht verstuurd worden. Na deze synchronisatie periode zal er nu weer 5 maal deel één herhaald worden voordat deel twee weer herhaald wordt.

5 Testopstelling en resultaten van de test

In deze paragraaf wordt het geïmplementeerde algoritme onderzocht op hoe snale de synchronisatie plaatsvind bij verschillende afstanden. De hypothese is dat naarmate de afstand groter wordt de tijd om te synchroniseren toenaamt. Er is meer tijd nodig om de berichten te versturen dus zal het langer duren voor het synchroon gaat. De schematische opstelling hiervan is te zien in tabel 1 en de reële opstelling zoals deze was is te zien in figuur 4.



Figuur 4: Testopstelling

Tabel 1: Testresultaten

afstand	**10 cm**	**25 cm**	**50 cm**	**100 cm**
1.	4,33 seconden	11,50 seconden	7,40 seconden	6,72 seconden
2.	4,72 seconden	5,58 seconden	17,46 seconden	6,18 seconden
3.	4,07 seconden	9,0 seconden	8,84 seconden	6,40 seconden
4.	3,82 seconden	9,85 seconden	12,54 seconden	5,71 seconden
5.	7,92 seconden	4,20 seconden	8,87 seconden	4,93 seconden
gemiddelde	4.972 seconden	8.026 seconden	11.22 seconden	5.988 seconden

In tabel 1 is te zien hoe de afstand van de nodes zich verhoudt tot de tijd die het neemt om te synchroniseren. Wat opmerkelijk is aan deze tabel is dat met een afstand van 50 centimeter de synchronisatie tijd soms meer dan twee keer zo lang neemt als wanneer er op een afstand van 100 centimeter wordt gesynchroniseerd. Twee mogelijke verklaringen voor dit fenomeen zijn:

- Door interferentie met wi-fi is er op bepaalde momenten soms meer of minder bereik van de radio's.
- door de gebruikte golflengtes van de radio kan het zijn dat er dode punten ontstaan zoals bij een satelliet. Er zijn bepaalde afstanden waar dit voor gebeurt en dit zou ook kunnen bij bijvoorbeeld 50 centimeter afstand.

6 Conclusie en aanbevelingen

De probleemstelling die gesteld werd is: *"het synchroniseren van nodes in een multihop netwerk met als doel ze tegelijk te laten knipperen."*. Na het testen van de implementatie kan er geconcludeerd worden dat het experiment geslaagd is en dus de hypothese waar is. In tabel 2 staan de eisen waar de implementatie aan voldoet. In het hoofdstuk :*Ontwerp synchronisatie algoritme* en in *Testopstellingen en resultaten* staan de implementatie en de testresultaten.

Tabel 2: Gestelde eisen aan het netwerk

nummer	**eis**
1.	Wanneer twee gesynchroniseerde netwerken worden samengevoegd dienen ze te synchroniseren.
2.	Wanneer een node uitvalt dient de synchronisatie nog te werken.
3.	Wanneer nodes toegevoegd worden aan het netwerk gaan deze met dezelfde frequentie knipperen als de andere nodes in het netwerk.
4.	Wanneer nodes uit sync raken moeten ze opnieuw synchroniseren.

Hoewel de code werkt en de tests zijn gelukt staat de implementatie open voor verbetering. Tijdens dit onderzoek was er slechts de beschikbaarheid over drie nodes. In de subsectie aanbevelingen worden verder testmethoden en uitbreidingen voorgesteld.

6.1 Aanbevelingen

Tijdens dit onderzoek was er slechts de beschikbaarheid over drie nodes. In een vervolgonderzoek kan er meer uitgebreid getest worden of het ook werkt met veel meer nodes. Vragen die hierbij gesteld kunnen worden zijn:

- Is er een maximum aan het aantal nodes in het netwerk waarop er nog gesynchroniseerd kan worden? Aangezien er gebruik wordt gemaakt van een random getal tussen 0 en 1000 is het van belang om te weten wanneer er vaak dubbele getallen voorkomen.
- veranderd de synchronisatie tijd die nodig is om te synchroniseren naarmate er meer nodes in het netwerk aanwezig zijn?
- De mogelijkheden onderzoeken om in plaats van een gedecentraliseerd netwerk te ontwikkelen een gecentraliseerd netwerk te synchroniseren. Gebruik makende van de code en hier enkele aanpassingen aan te maken.

7 Bronnen

A Code Sender

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

/****************************************************
 *                                                  *
 *                   CONFIGURATION                  *
 *                                                  *
 ****************************************************/
  RF24 radio(3, 9);
  const uint64_t pipes =  0xF0F0F0F0E1LL;

  int minID = 0;
  int maxID = 1000;
  int ID;

  const char DELIM = ':';
  const char* messageBuffer;

  //Previous synchronisation data needed to detect loops
  int prevHighestID; //Previous Highest found ID
  int prevMessageID; //Previous message ID

  int highestID;  //Highest found ID
  int senderID;   //Sender of highest ID
  int messageID;  //MessageID
  long timestamp; //Timestamp of highestID
  long deltaT;
  boolean lightOn = false;
  int sendMax = 5;  //amount of messages sent before synchronisation
  int messageCount; //amount of sent messages
  int night = 1000; //sleep time in ms

  //Set up LED
  int led = 7;

void setup(void){
  Serial.begin(57600);

  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
  //create random ID for this device
  randomSeed(analogRead(0));
```

```
  ID = random(minID, maxID);

  printf_begin();
  printf("NODE ID: %i \n\r",ID);

  radio.begin();
  radio.setRetries(0,0);
  radio.setPayloadSize(sizeof(MESSAGE));
  radio.openWritingPipe(pipes);
  radio.openReadingPipe(1,pipes);
  radio.startListening();
  radio.setAutoAck(false);
  radio.printDetails();
  messageCount = 0;
  sendMax++;
  prevHighestID = -1;
  prevMessageID = -1;
}



//
//  Listen for messages and process
//
void listenFor(long time){
  long t = millis();
  MESSAGE ontvangen;
  while(millis()-t < time){
    if(radio.available()){
      radio.read( &ontvangen, sizeof(MESSAGE));
      processMessage(ontvangen);
    }
  }
}


void processMessage(MESSAGE received){
  processContent(received.identity, received.high_identity, received.number);
}

//
// process message and change highestID if necessary
//
void processContent(int sender, int high, int messNumber){
   printf("RECEIVED: %i, %i, %i\n\r", sender, high, messNumber);
   if((high >  highestID && !(prevMessageID == messNumber && prevHighestID == high))   //if
```

```
   || (high == highestID && sender == high)                //or if we receive previously rec
   ){                                                      //update highest
     deltaT   = millis()-timestamp;                        //Tijd die nodig is van luisteren
     printf("NEW HIGHEST!\n\r");
     highestID = high;
     senderID  = sender;
     messageID = messNumber;
   }
}

void sendMessage(){
  MESSAGE bericht = { ID, highestID, messageID };
  printf("SEND: %i, %i, %i: ", ID, highestID, messageID);
  radio.stopListening();
  bool ok = radio.write(&bericht, sizeof(MESSAGE));
  radio.startListening();
  if(ok){
    printf("SENT CORRECTLY\n\r");
  }else{
    printf("SENDING FAILED\n\r");
  }
  messageCount++;
}

void synchronize(){
  highestID = ID;           //reset data
  deltaT    = night;
  timestamp = millis();
  messageID = messageCount;

  listenFor(night);          //first night: find highest
  prevMessageID = messageID; //update prevMessageID
  prevHighestID = highestID; //update prevHighestID
  if(deltaT == night){       //if no new highest found
     sendSignal();
  }else{                     //toggle LED to indicate wake-up
    toggleLED();
  }
  printf("SLEEP FOR %ul", deltaT);
  sleepFor(deltaT);         //second night: synchronise
  sendSignal();
}

void toggleLED(){
  if(lightOn){
     lightOn = false;
```

```
    digitalWrite(led, LOW);  // turn the LED off (LOW is the voltage level)
  }else{
    lightOn = true;
    digitalWrite(led, HIGH); // turn the LED on by making the voltage HIGH
  }
}

void sendSignal(){
  toggleLED();
  sendMessage();
}

void loop(void){
  while((messageCount % sendMax) != 0){
    sleepFor(night);
    sendSignal();
  }
  synchronize();
}

void sleepFor(long time){
 delay(time);
}
```

B Terminologie

Tabel 3: Toelichting op de implementatie mogelijkheden

Onderdeel	**Uitleg**
Master-slave protocol	Er is een master node in het netwerk waar alle andere nodes naar luisteren, deze bepaald de synchronisatie
peer-to-peer protocol	Elke node in het netwerk heeft contact met elke andere node. Dit elimineert het risico dat wanneer een master node wegvalt het netwerk niet meer synchroniseert.
Interne synchronisatie	De reële tijd is niet beschikbaar. Het doel is het verschil in tijd tussen de lokale klok en het uitlezen van de sensor zo klein mogelijk te maken.
externe synchronisatie	Hier is de tijd wel beschikbaar zoals UTC. Er is een atomische klok die de ware tijd heeft.
single hop netwerk	Elke node in het netwerk heeft contact met elke andere node in het netwerk
multi hop netwerk	Niet elke node in het netwerk heeft contact met een andere node in het netwerk.
probabilistische synchronisatie	Geeft een probabilistische garantie van de maximum offset van de klok.
deterministische synchronisatie	Deze algoritme garanderen een bepaalde maximum offset van de klok.