Practicum 2 Synchronisatie bij vuurvliegjes

Peter van Dijk s1102109 & Elizabeth Schermerhorn s1223380

3 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 bekend voorbeeld uit de natuur zijn vuurvliegjes die met elkaar synchroniseren, wat zich uit in samen knipperen. Het algoritme dat vuurvliegjes gebruiken om synchroon te knipperen wordt geïmplementeerd in een multi-hop netwerk. Door het algoritme te gebruiken moet het mogelijk zijn om nodes tegelijk te laten knipperen net zoals een groep vuurvliegjes.

2 Probleemstelling

Het doel van dit onderzoek is het vinden van een algoritme om Arduino's te synchroniseren met behulp van radio's. Een aantal eisen waar dit algoritme aan moet voldoen zijn:

- 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 synchronisatie raken moeten ze opnieuw synchroniseren.

Nu is vastgesteld aan welke eisen het algoritme 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 multi-hop 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 [1]. De tweede paper, Wireless sensor network survey, geeft een overzicht van bestaande draadloze netwerk implementaties op verschillende niveaus [3] en als laatste geeft het boek Academic Press Library in Signal Processing een kort overzicht van tijd synchronisatie in een netwerk [2].

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 designkeuzes 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 implementatiemogelijkheid inhoudt.

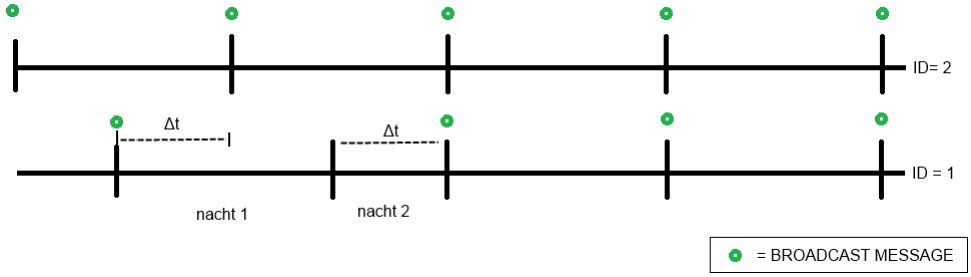
Afhankelijk van de eisen die aan het netwerk gesteld worden moet er voor bepaalde onderdelen gekozen worden. Eisen aan het netwerk zouden kunnen zijn: Er is geen server beschikbaar, het moet zo min mogelijk energie verbruiken zodat de nodes lang meegaan of de nodes moeten over grotere afstand kunnen communiceren. 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 dat in deze paper besproken wordt is gebaseerd op het vuurvliegalgoritme. Hoewel het geïmplementeerde algoritme erop gebaseerd is, is het op sommige punten aangepast. Het vuurvliegalgoritme 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. De bedoeling is dat alle nodes tegelijk een signaal gaan versturen na verloop van tijd. Het grote verschil tussen bovenstaande imple-

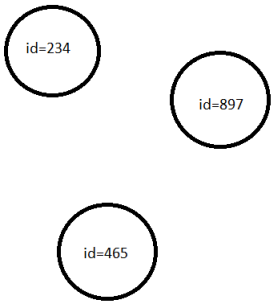


Figuur 1: Tijdlijnen van twee synchroniserende nodes

mentatie 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 het tijdsverschil afneemt. Door dit fenomeen synchroniseren na verloop van tijd de nodes in het netwerk. **??**

4 Ontwerp synchronisatie-algoritme

Het synchronisatie-algoritme is gebaseerd op een master-slave structuur. In Figuur 2 staat een schematische weergave van het netwerk zoals deze is getest. Er zijn ID's meegegeven aan de nodes zodat er te zien is dat er één master is met één of meerdere slaves. Het algoritme dat deze nodes gebruiken voor de synchronisatie staat in pseudocode in Figuur 3. Verder is de synchronisatie tussen twee nodes visueel weergegeven in Figuur **??**. Iedere node in het netwerk krijgt een willekeurig ID tussen de 0 en de 1000. De master van het netwerk wordt gekozen aan de hand van het hoogste ID. Er is gekozen voor deze structuur aangezien er gesynchroniseerd moet worden op een gekozen node. Het doel is om de nodes tegelijk te laten knipperen. Om dit te doen moeten de interne klokken van de nodes gelijk lopen. In onze implementatie detecteert een node het moment dat de master wakker is en zal zijn interne klok corrigeren naar dit



Figuur 2: Schematische weergave van de testopstelling

moment. Binnen deze master-slave structuur is de implementatie verdeeld in twee alternerende delen.

In het tweede deel wordt een aantal maal achtereen alternerend de Arduino en Radio in slaapstand gezet en vervolgens een bericht verstuurd. De lengte van deze slaapstand, of 'nacht' staat vast voor het hele netwerk.

In het eerste deel wordt er gesynchroniseerd, eerst wordt er voor één nacht geluisterd of er berichten zijn ontvangen. Als er berichten zijn ontvangen worden deze verwerkt. Dan wordt er voor een tijd, die afhankelijk is van de ontvangen data, geslapen en ten slotte 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

De RF24-radio wordt gebruikt met de volgende instellingen:

- geen hertransmissies
- pakketgrootte: grootte van bericht
- geen automatische bevestiging bij ontvangen bericht

Verder wordt er gebruik gemaakt van de standaardinstellingen. Het kanaal waarop gezonden wordt is hetzelfde kanaal als waarop geluisterd wordt, waardoor een node dus óf aan het luisteren is óf aan het zenden, aangezien de radio niet beiden tegelijk kan op het zelfde kanaal.

Het uitzetten van hertransmissies zorgt ervoor dat er geen berichten worden gestuurd op momenten die niet het wakker worden van de node aanduiden. Als deze hertransmissies wel gestuurd zouden worden, dan zou er gesynchroniseerd

```
time nacht    = 1000 ms;
int ID        = random(0,1000);
int berichten = 0;
int hoogsteID, berichtNummer, vorigHoogsteID, vorigBerichtNummer;

while(true){
    //Eerste deel: synchronisatie
        hoogsteID   = ID;
        time deltaT = nacht;
        timestamp t = nu();
        vorigHoogsteID = hoogsteID;
        vorigBerichtNummer = berichtNummer;
        //eerste nacht
            for(nacht){
                if(radio.ontvang(){
                    if(radio.bericht.hoogsteID > hoogsteID &&
                        !(radio.bericht.hoogsteID == vorigHoogsteID
                           && radio.bericht.berichtNummer == vorigBerichtNummer
                        )
                    ){
                        deltaT = t - nu();
                        hoogsteID = radio.bericht.hoogsteID;
                        berichtNummer = radio.bericht.berichtNummer;
                    }
                }
            }
        //tweede nacht
            //Als node niet master node is
            if(hoogsteID != ID){
                slaap(deltaT);
                zendBericht;
            }
    //Tweede deel: slapen
        for(int i=0;i<5;i++){
            slaap(nacht);
            zendBericht();
        }
}

func zendBericht(){
    radio.zend([ID, hoogsteID, berichten]);
    berichten++;
}
```

Figuur 3: Pseudoalgoritme

kunnen worden op hertransmissies, wat niet op het juiste tijdstip zou zijn. Het uitzetten van de automatische bevestiging zorgt er voor dat er geen onnodige berichten worden gestuurd, immers zal er niets gedaan worden met de bevestiging, aangezien berichten niet opnieuw gestuurd worden.

4.2 Berichten

Berichten die verstuurd worden bevatten de volgende data:

1. ID van de zender
2. hoogst door zender ontvangen ID
3. ID van bericht

De hoogste ID is nodig om te beslissen of een bepaalde node de master node is en of daarnaar geluisterd moet worden. De ID van de zender toegevoegd om te kunnen achterhalen welke node het bericht heeft gestuurd.

Er is voor gekozen berichten een ID te geven, waarbij de combinatie tussen hoogst ontvangen ID en ID van het bericht uniek is binnen het netwerk. Hierdoor kunnen veroudere berichten gedetecteerd worden, zodat deze niet gebruikt worden om op te synchroniseren. Dit voorkomt dat nodes een ID van een uitgevallen node blijven versturen als hoogst gevonden ID.

4.3 Het eerste deel, synchroniseren

Het eerste deel bestaat uit alternerend slapen en berichten versturen. In het eerste deel van het programma, zoals aangegeven in Figuur 3, wordt voor een nacht gekeken welke andere nodes er in het netwerk aanwezig zijn.

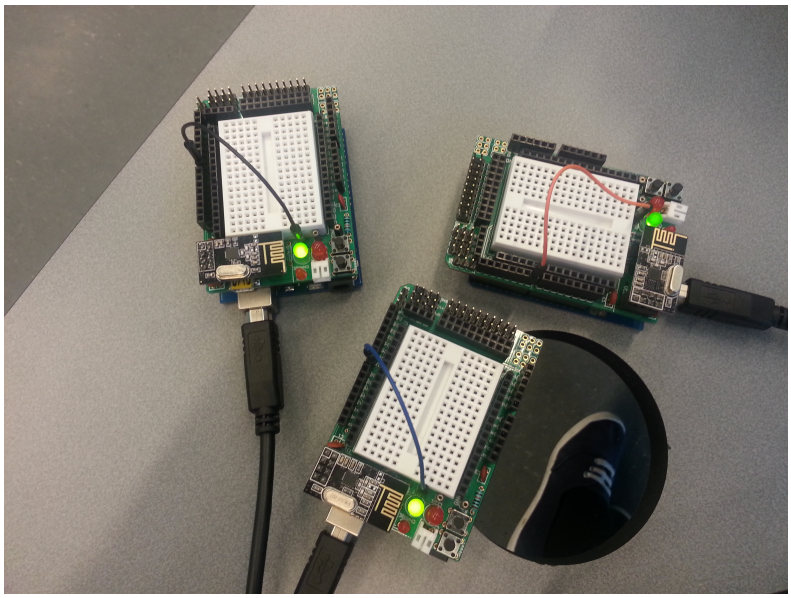
Op het moment dat er begonnen wordt met luisteren wordt er een tijdstip t_0 geklokt. Op het moment dat er een bericht van de node met het hoogst gedetecteerde ID wordt ontvangen wordt er een tweede tijdstip t_1 geklokt. Δt, oftewel $t_1 - t_0$, is het tijdsverschil tussen de interne klokken van de Arduino's. Daarom wordt de tweede nacht van de synchronisatie voor Δt geslapen. Na deze nacht zijn de nodes gesynchroniseerd en zal de node een bericht sturen met het grootst gevonden ID. Na de eerste nacht wordt geen bericht gestuurd, omdat de node dan nog niet gesynchroniseerd is.

4.4 Het tweede deel, slapen en versturen

Als de nodes gesynchroniseerd zijn zullen ze alternerend een nacht slapen en een bericht sturen. Deze berichten worden gestuurd zodat ongesynchroniseerde nodes kunnen synchroniseren op de ontvangst van deze berichten.

5 Testopstelling en resultaten van de test

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



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

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 synchronisatietijd soms meer dan twee keer zo groot is 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 dit ook het geval is bij satellieten. 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 vinden van een algoritme om Arduino's te synchroniseren met behulp van radio's"*. Na het testen van de implementatie kan er geconcludeerd worden dat deze de Arduino's laat synchroniseren en het voorgestelde algoritme dus werkt. 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 willekeurig ID tussen 0 en 1000 is het van belang om te weten wanneer er vaak dubbele ID voorkomen.
- Verandert de synchronisatietijd 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 een gecentraliseerd netwerk te synchroniseren, gebruik makende van de code met enkele aanpassingen.

Referenties

[1] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization for wireless sensor networks: A survey. *Ad Hoc Networks (Elsevier*, 3:281–323, 2005.

[2] S. Theodoridis and R. Chellappa. *Academic Press Library in Signal Processing: Signal Processing Theory and Machine Learning*. Academic Press Library in Signal Processing. Elsevier Science, 2013.

[3] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, Aug. 2008.

A Code Node

```
#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 we receive a higher ID than our highest yet received
```

```
   //or if we receive previously received highest from highest node itself
   if((high >  highestID && !(prevMessageID == messNumber && prevHighestID == high))
   || (high == highestID && sender == high)
   ){ //update highest
     deltaT   = millis()-timestamp;
     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.