Betrouwbare end-to-end communicatie

Patrick van Looy & Bram Leenders

2 juli 2014

1 Inleiding

Bij het opzetten van sensornetwerken, kan het zijn dat twee nodes niet in elkaars zendbereik vallen. In een dergelijk geval kan een tussenliggende node helpen door berichten door te sturen. Zo kunnen nodes als schakels in een ketting gebruikt worden om een groter bereik mogelijk te maken.

Een dergelijke constructie is een stuk gecompliceerder dan directe communicatie tussen twee nodes, omdat voor betrouwbare communicatie het zenden van ontvangstbevestigingen nodig is. Het communiceren binnen een groep nodes is daarom ook geen onderdeel van de standaard meegeleverde libraries. Het doel van dit onderzoek was het mogelijk maken van communicatie via nodes, gebruikmakend van de functionaliteit die wel door Arduino's geleverd wordt.

2 Probleemstelling

Om het probleem overzichtelijk te houden, is het aantal nodes in deze proef beperkt tot drie. Er is een zender, een ontvanger en een doorsturende node. Omdat het aantal nodes beperkt is, is er maar één mogelijk pad. Hierdoor kan het pad vooraf vastgelegd worden, dit heet ook wel statische routering.

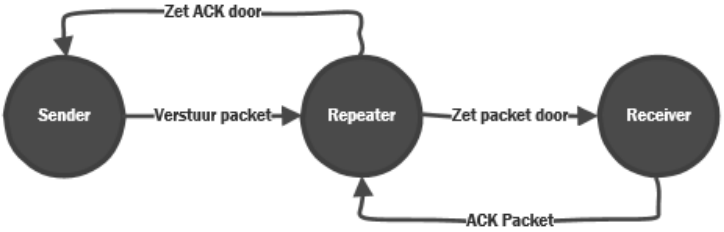
De uiteindelijke opstelling moet de zender de garantie geven dat een bericht uiteindelijk ontvangen wordt door de ontvanger via de tussenliggende node.

We willen nu weten of het mogelijk is om via een multihopnetwerk betrouwbare end-to-end communicatie te garanderen. Specifiek verwoord: dit onderzoek heeft als doel om een protocol te ontwikkelen dat de juiste packetvolgorde en de volledigheid (er ontbreken geen packets) garandeert. Van dit protocol wordt tevens een implementatie gegeven.

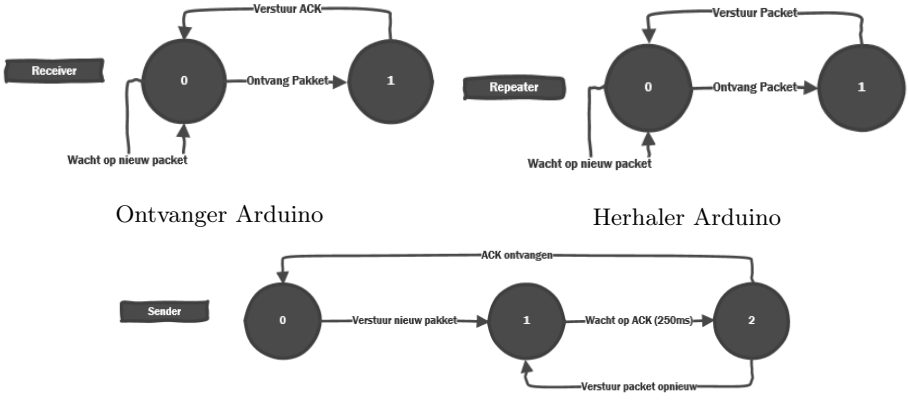
3 Protocol

Zoals gezegd is dient het protocol te garanderen dat er geen packets wegvallen tijdens het verzenden en packets de juiste volgorde te behouden. Om dit te garanderen, kan gebruik worden gemaakt van het alternating bit protocol (ABP). Dit protocol stuurt een nummer -een bit- mee met een pakket, wanneer de ontvanger het ontvangt stuurt deze bevestiging met datzelfde nummer terug. Indien de zender een bevestiging met het correcte (laatst verzonden) nummer ontvangt, hoogt deze een counter met een op en stuurt deze het volgende pakket.

Figuur 1 illustreert het pad dat een pakket met bijbehorende bevestiging aflegt. Figuur 2 toont de verschillende staten waarin een zender/herhaler/ontvanger zich kan bevinden en de transities tussen deze staten.



Figuur 1



Figuur 2: State-diagram Alternating Bit Protocol voor de drie communicerende Arduino's.

4 Implementatie

Onze implementatie gebruikt niet een enkele bit als identificatie van een pakket, maar een getal. Conceptueel verandert dit niets aan het protocol, maar het biedt wel de mogelijkheid voor later uitbreiding. Zo geldt niet langer de restrictie dat er slechts één oud pakket in het netwerk mag zitten. Onze implementatie werkt dus ook als oudere berichten (e.g. tien berichten terug) nog ronddolen in het netwerk.

Tevens stuurt de ontvanger een ACK terug die niet de identificatiecode bevat, maar de negatieve waarde hiervan. Hierdoor is het direct duidelijk of een pakket een bevestiging is (indien de code kleiner dan 0 is), of echte data bevat. Dit helpt om ook in grote netwerken ervoor te zorgen dat een zender niet zijn zelf verzonden bericht als ACK kan lezen.

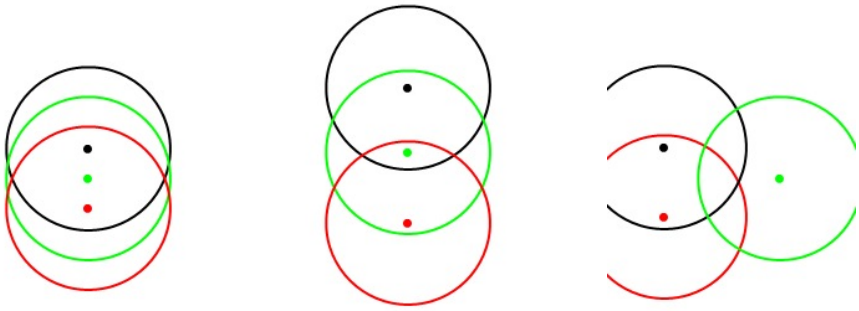
Verder moet de communcatie op zowel korte als lange afstand betrouwbaar zijn, ook dit moet getest worden.

5 Test methodologie en resultaten

De implementatie, waarvan de code is bijgevoegd in appendix A, is in verschillende vormen getest:

- De zender, herhaler en ontvanger allemaal binnen elkaars ontvangstbereik.
- De zender en herhaler respectievelijk de herhaler en ontvanger binnen ontvangstbereik, maar zender en ontvanger buiten ontvangstbereik.
- Alleen zender en ontvanger binnen ontvangstbereik; herhaler buiten bereik.

Omdat er gebruik gemaakt is van statistische routering, is het geen optie voor de zender en ontvanger om direct te communiceren. In de eerste twee gevallen verloopt de communicatie via de herhaler; in het derde geval kan geen verbinding worden opgezet omdat de herhaler niet bereikbaar is.



Alles direct bereikbaar | Ontvanger bereikbaar via herhaler | Herhaler onbereikbaar

Figuur 3: Drie mogelijke scenario's; zwart is de verzender, groen is de herhaler en rood is de ontvanger.

Op zowel korte als lange afstand (tussen zender en ontvanger zo'n veertig meter, overbrugd via herhalers) bleken de nodes in staat te zijn alle packets succesvol af te leveren. Eventueel packet loss wordt door het algoritme gedetecteerd, en de ontbrekende packets worden opnieuw gestuurd totdat deze ontvangen zijn. In deze gevallen werkt het protocol dus zoals gewensd, in het derde geval werkt het protocol niet maar werd dit ook niet verwacht. Immers, dit scenario valt buiten de oorspronkelijke eisen.

6 Conclusie

Concluderend, het is wel degelijk mogelijk om via een multihopnetwerk betrouwbare end-to-end communicatie te garanderen. Bij het uitgevoerde onderzoek zijn de resultaten verassend goed. Over een grote afstand waarbij veel mogelijke obstakels en stoorzenders aanwezig waren, bleken de nodes in staat te zijn

nagenoeg alle individuele packets succesvol af te leveren bij de desbetreffende ontvanger.

In gedachten nemend dat een (sensor)netwerk op een veel grotere schaal nog steeds betrouwbaar moet zijn, is het wel verstandig ervoor te zorgen dat er altijd twee andere nodes bereikt kunnen worden. Op deze manier is het veiliger om te zeggen dat een packet altijd bij zijn bestemming aankomt, omdat je er dan rekening mee houdt dat er eventueel een node kapot zou kunnen gaan of uitvalt. Mocht dit gebeuren, dan kan de andere node die binnen bereik is de taak van de kapotte node overnemen. Dit valt echter buiten de scope van dit onderzoek, en kan in een volgend onderzoek verder uitgewerkt worden.

Verder onderzoek zou gedaan kunnen worden naar het dynamisch maken van het netwerk. Op die manier is het veel gemakkelijker om nodes te verwijderen of toe te voegen in een netwerk en gaat het netwerk dus ook beter om met een node die kapot gaat of uitvalt. Om deze functionaliteit te kunnen bieden moet het protocol nog veel aangepast worden en is het nodig om dynamische routering te implementeren.

A Bijlage 1 - Code

```
/* Copyright (C) 2011 J. Coliz <maniacbug@ymail.com> */

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

RF24 radio(3, 9);

// sets the role of this unit in hardware.  Connect to GND to be the 'pong' receiver
// Leave open to be the 'ping' transmitter
const int role_pin_sender = 7;
const int role_repeat = 6;

// Radio pipe addresses for the 3 nodes to communicate.
const uint64_t pipes[3] = { 0x123456789aLL, 0x987654321bLL, 0x384654f2cbLL };

const int sendValue = 170; // binary; 10101010
const int numberOfPackets = 1000;
const int RESETVAL = 42;

typedef enum { role_sender = 1, role_receiver = 2, role_repeater = 3 } role_e;
// The debug-friendly names of those roles
const char* role_friendly_name[] = { "invalid", "Sender", "Receiver", "Repeater"};

// The role of the current running sketch
role_e role;

void setup(void) {
    // set up the role pin
    pinMode(role_pin_sender, INPUT);
    digitalWrite(role_pin_sender,HIGH);
    delay(20); // Just to get a solid reading on the role pin
    // read the address pin, establish our role
    if (!digitalRead(role_pin_sender)) {
        role = role_sender; // sender
    }
    else {
        // set up the role pin
        pinMode(role_repeat, INPUT);
        digitalWrite(role_repeat,HIGH);
        delay(20); // Just to get a solid reading on the role pin

        if (!digitalRead(role_repeat)) {
            role = role_repeater; // repeater
        }
        else {
            role = role_receiver; // receiver
        }
    }

    Serial.begin(57600);
    printf_begin();

```

```
55      radio.begin();
56      radio.setRetries(0,0);
57
58      radio.setDataRate(RF24_250KBPS);
59      radio.setPALevel(RF24_PA_MAX);
60      radio.setChannel(0);
61
62      // optionally, reduce the payload size.  seems to
63      // improve reliability
64      radio.setPayloadSize(8);
65
66      if ( role == role_sender ) { // Sender
67          radio.openWritingPipe(pipes[1]); // Write to repeater
68          radio.openReadingPipe(1,pipes[0]); // Read from repeater
69      }
70      else if (role == role_repeater){ // Repeater
71          radio.openReadingPipe(1,pipes[1]);
72      }
73      else { // Receiver
74          radio.openWritingPipe(pipes[1]);
75          radio.openReadingPipe(1,pipes[2]);
76      }
77
78      radio.startListening();
79      radio.printDetails();
80  }
81
82  // Bevat het nummer dat nu gestuurd moet worden
83  int currentNumber = 1;
84
85  void resetRadioReads(void) {
86      radio.stopListening();
87      radio.openWritingPipe(0x111111110aLL);
88  }
89
90  void loop(void) {
91      if (role == role_sender) {
92          radio.stopListening();
93          radio.openWritingPipe(pipes[1]);
94          bool ok = radio.write( &currentNumber, sizeof(int) );
95          resetRadioReads();
96          radio.startListening();
97
98          // Wait here until we get a response, or timeout (250ms)
99          unsigned long started_waiting_at = millis();
100         bool timeout = false;
101         while ( ! radio.available() && ! timeout )
102             if (millis() - started_waiting_at > 250 )
103                 timeout = true;
104
105         // Describe the results
106         if ( timeout ) {
107           // Zend opnieuw!
108           // currentNumber wordt opnieuw verzonden in de volgende↩
                  iteratie van loop()
109           printf("Timeout occurred at sender; no ACK received. ↩
                  Packet #: %i\n", currentNumber);
110         }
111         else {
112             int receivedValue;
113             radio.read( &receivedValue, sizeof(int) );
114
```

```
115             // ACK our value! Increase our success counter.
116             if(receivedValue == - currentNumber) {
117                 // Success!
118                 printf("ACK succesfully received for packet #%i\n↩
                        ", currentNumber);
119                 currentNumber++;
120             }
121             else {
122               // Received double ACK
123               printf("Received double ACK. Packet #: %i\n", ↩
                      currentNumber);
124             }
125         }
126         delay(50);
127     }
128
129     if ( role == role_receiver ) {
130         // if there is data ready
131         if ( radio.available() ) {
132
133             // Dump the payloads until we've gotten everything
134             int v; bool done = false;
135             while (!done) {
136                 done = radio.read( &v, sizeof(int) );
137                 delay(10);
138             }
139
140             printf("Received packet # %i\n", v);
141
142             v = -v; // ACK waarde is negatief
143
144             radio.stopListening();
145             radio.openWritingPipe(pipes[1]);
146             radio.write( &v, sizeof(int) );
147             resetRadioReads();
148             radio.startListening();
149         }
150     }
151     if (role==role_repeater) {
152         if ( radio.available() ) {
153             int v; bool done = false;
154             while (!done) {
155                 done = radio.read( &v, sizeof(int) );
156                 delay(10);
157             }
158             radio.stopListening();
159
160             if (v > 0) { // Dit is het originele bericht; stuur ↩
                    naar receiver
161                 radio.openWritingPipe(pipes[2]);
162             }
163             else { // Dit is de ACK; stuur naar sender
164                 radio.openWritingPipe(pipes[0]);
165             }
166
167             radio.write( &v, sizeof(int) );
168             resetRadioReads();
169             radio.startListening();
170         }
171     }
172 }
```