# Betrouwbare end-to-end communicatie

Patrick van Looy & Bram Leenders

9 mei 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.

## 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.

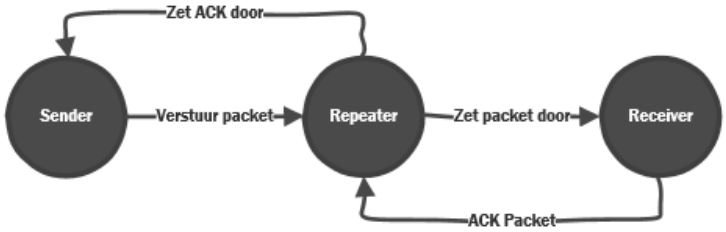
De uiteindelijke opstelling moet de zender de garantie geven dat een bericht uiteindelijk ontvangen wordt door de ontvanger via de tussenliggende node.

We willen nu weten of het mogelijk is om via een multihopnetwerk betrouwbare end-to-end communicatie te garanderen.

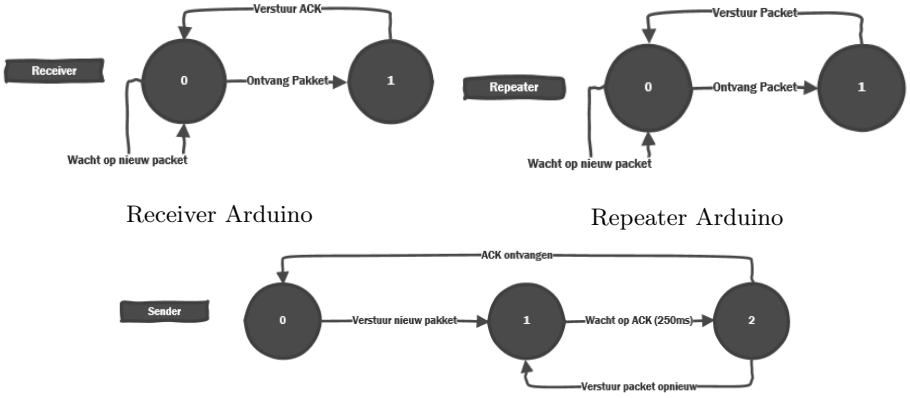
## 3 Protocol

Om deze garantie te bieden, kan gebruik worden gemaakt van het alternating bit protocol. 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/repeater/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 Methodologie

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.

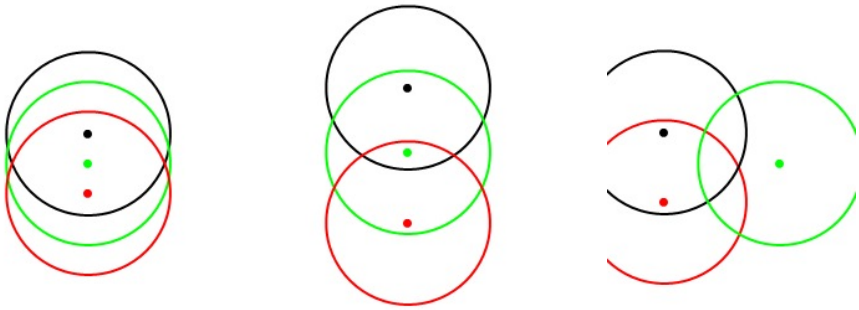
Verder moet de communcatie op zowel korte als lange afstand betrouwbaar zijn, ook dit moet getest worden.

# 5 Resultaten

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

- De zender, repeater en ontvanger allemaal binnen elkaars ontvangstbereik.
- De zender en repeater respectievelijk de repeater en ontvanger binnen ontvangstbereik, maar zender en ontvanger buiten ontvangstbereik.
- Alleen zender en ontvanger binnen ontvangstbereik; repeater 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 repeater; in het derde geval kan geen verbinding worden opgezet omdat de repeater niet bereikbaar is.



Alles bereikbaar | Repeater bereikbaar | Repeater onbereikbaar

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

Op zowel korte als lange afstand (tussen sender en receiver zo'n veertig meter) bleken de nodes in staat te zijn nagenoeg alle packets succesvol af te leveren.

# 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.

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();

    radio.begin();
    radio.setRetries(0,0);
```

```
    radio.setDataRate(RF24_250KBPS);
    radio.setPALevel(RF24_PA_MAX);
    radio.setChannel(0);

    // optionally, reduce the payload size.  seems to
    // improve reliability
    radio.setPayloadSize(8);

    if ( role == role_sender ) { // Sender
        radio.openWritingPipe(pipes[1]); // Write to repeater
        radio.openReadingPipe(1,pipes[0]); // Read from repeater
    }
    else if (role == role_repeater){ // Repeater
        radio.openReadingPipe(1,pipes[1]);
    }
    else { // Receiver
        radio.openWritingPipe(pipes[1]);
        radio.openReadingPipe(1,pipes[2]);
    }

    radio.startListening();
    radio.printDetails();
}

// Bevat het nummer dat nu gestuurd moet worden
int currentNumber = 1;

void resetRadioReads(void) {
    radio.stopListening();
    radio.openWritingPipe(0x111111110aLL);
}

void loop(void) {
    if (role == role_sender) {
        radio.stopListening();
        radio.openWritingPipe(pipes[1]);
        bool ok = radio.write( &currentNumber, sizeof(int) );
        resetRadioReads();
        radio.startListening();

        // Wait here until we get a response, or timeout (250ms)
        unsigned long started_waiting_at = millis();
        bool timeout = false;
        while ( ! radio.available() && ! timeout )
            if (millis() - started_waiting_at > 250 )
                timeout = true;

        // Describe the results
        if ( timeout ) {
          // Zend opnieuw!
          // currentNumber wordt opnieuw verzonden in de volgende
              iteratie van loop()
          printf("Timeout occurred at sender; no ACK received.
              Packet #: %i\n", currentNumber);
        }
        else {
            int receivedValue;
            radio.read( &receivedValue, sizeof(int) );

            // ACK our value! Increase our success counter.
            if(receivedValue == - currentNumber) {
```

```
                // Success!
                printf("ACK succesfully received for packet
                    #%i\n", currentNumber);
                currentNumber++;
            }
            else {
              // Received double ACK
              printf("Received double ACK. Packet #: %i\n",
                  currentNumber);
            }
        }
        delay(50);
    }

    if ( role == role_receiver ) {
        // if there is data ready
        if ( radio.available() ) {

            // Dump the payloads until we've gotten everything
            int v; bool done = false;
            while (!done) {
                done = radio.read( &v, sizeof(int) );
                delay(10);
            }

            printf("Received packet #%i\n", v);

            v = -v; // ACK waarde is negatief

            radio.stopListening();
            radio.openWritingPipe(pipes[1]);
            radio.write( &v, sizeof(int) );
            resetRadioReads();
            radio.startListening();
        }
    }
    if (role==role_repeater) {
        if ( radio.available() ) {
            int v; bool done = false;
            while (!done) {
                done = radio.read( &v, sizeof(int) );
                delay(10);
            }
            radio.stopListening();

            if (v > 0) { // Dit is het originele bericht; stuur
                naar receiver
                radio.openWritingPipe(pipes[2]);
            }
            else { // Dit is de ACK; stuur naar sender
                radio.openWritingPipe(pipes[0]);
            }

            radio.write( &v, sizeof(int) );
            resetRadioReads();
            radio.startListening();
        }
    }
}
```