Anti Bicycle Theft

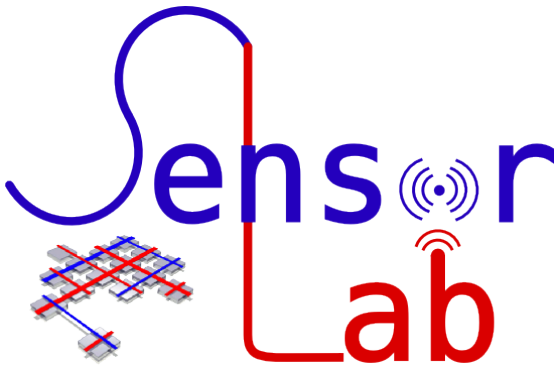
Documentation

Kevin Freeman (TODO1)
Martin Schwarzmaier (TODO2)
Georg-August-Universität Göttingen

January 25, 2016

Practical Course on Wireless Sensor Networks

Lab Advisor: Dr. Omar Alfandi
Lab Assistants: Arne Bochem, M.Sc.



Contents

1 Introduction

... bicycle theft, current problem (goettingen e.g.), pro contra to GSM/Bluetooth e.g.
Neues Vorgehen: also 1. intro, 2. used motes + sensorboard + mongodb + nodejs etc und 3 walkthrough =¿ wie man es benutzt und 4 dann alle bestandteile und protokolle etc erklären

2 Used Sensorboards and Motes

We used the standard IRIS motes for everything.... `http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf`
sensorboard mts-420cc
gps antenna

3 Walkthrough

This section demonstrates the project without showing technical details, as they will be explained in Section %TODO refhere.

3.1 Flashing the motes

In order to make this project work, at least one IRIS mote as a base station as well as one iris mote equipped with a mts420-cc board including a gps antenna are needed. Each additional bicycle will need the same sensorboard and gps antenna. Additionally, it is possible to extend the network range with extra motes. This brings us to the following amount of needed motes:

a) one base station `[./nesC/base_station]`

b) one mts420-cc sensorboard with gps antenna per bicycle `[./nesC/bike]`

c) any amount of network node motes `[./nesC/nodes]`

3.2 Starting MongoDB, NodeJS and SerialForwarder

As the whole project is supposed to be user friendly, a webserver (Node.js) is implemented including a database (MongoDB). The webserver can be started by running the following command and will be available under `localhost:8080` afterwards:

```
$ node ./webapp/app.js
```

Additionally, the MongoDB daemon has to be started:

```
# mongod
```

Now the user is able to register to the webservice and mark his bicycles as stolen them already as shown in Figure **??**. Once gps data is collected, he is able to review this on the same webpage, as seen in Figure **??**. %TODO screenshot example

On the other side, the base station establishes a connection to the database via python, which is calling the java classes `net.tinyos.tools.Send` and `net.tinyos.tools.Listen`. These are available after starting the `SerialForwarder` with:

```
$ java net.tinyos.sf.SerialForwader
                                    -comm serial@/dev/ttyUSB1:iris
```

Afterwards, only the `listen.py` has to be run, which imports our `./python-api/env/bikeDB/bikeDB.py` script. This will enable automatic gathering of stolen bicycle `IDs` from the database and push them into the network, but will also collect gps coordinates from bicycles and save them in the database.

```
$ python2 ./python-api/env/bikeDB/listen.py
```

Our test environment used the following Linux packages:

1. nodejs (version 5.4.1)
2. python2 (version 2.7.11)
3. mongodb (version 3.2.0)
4. jdk8-openjdk (version 8.u66)
5. python2-pymongo (version 3.2)

%TODO: brauch man npm? Packages needed node, mongo, python2, java, pymongo

3.3 Registration and Tracking

%TODO Screenshot of Views
is this section still necessary?

4 network protocols and mote insight %todo: clever sectionname here

As the walkthrough only showed how to use the project, this section will show the programs and protocols used for each mote in detail. As all motes are supposed to talk with each other, one superior header file is implemented. It provides easy changeable variables for the maximal amount of bicycles, that can be stolen at once as well as how many gps coordinates are gathered in a single packet as seen in Listing 1. The gathering process uses the Collection %TODO quelle protocol and the propagating of stolen bicycle `IDs` is done via the Dissemination protocol.

```
1 //...
2 #define MAXBIKES 10
3 #define COORDS_PER_PACKET 2
4 typedef nx_struct EasyDisseminationMsg
5 {
6     nx_uint16_t bikes[MAXBIKES];
```

```
7  } EasyDisseminationMsg;
8  //...
9  typedef nx_struct EasyCollectionMsg
10 {
11     nx_uint16_t nodeid;
12     nx_uint32_t current_time;
13     nx_uint32_t time[COORDS_PER_PACKET];
14     nx_uint32_t lat[COORDS_PER_PACKET];
15     nx_uint32_t lon[COORDS_PER_PACKET];
16 } EasyCollectionMsg;
```

Listing 1: DataMsg.h, content of packets

Obviously, the content and variable types can be changed easily, too.

4.1 Base Station

Our base station `[./nesC/base_station]` is connected to a computer via USB. On this computer, a `SerialForwarder` is run in order to establish a possibility to send and receive data to and from the base station. For this project, on the one hand, we have to push `IDs` of stolen bikes to the base station in order to disseminate them through the network. On the other hand, the collection protocol is implemented to gather information from stolen bikes, like coordinates. This is done via a python script, which can be seen in Listing 2. In line it imports our bikeDB class, which is simply providing methods to read and write to the database.

```
1 todo: push the new one -.-
```

Listing 2: listen.py, ...

%TODO several picutres here (CLI, serialforwarder)

4.2 Network Node

The network nodes are completely omittable as they only enlarge the network. More network nodes are needed if a great availability of the network is wanted. These are connected with other network nodes and disseminate and collect the data mentioned already to and from the base station and bicycle motes. Currently, they are doing nothing but disseminating and collecting. However, functionality can be easily added in the `./nesC/nodes/NodeC.nc`.

4.3 Bicycle Mote

Bicycle motes are attached to each bicycle and equipped with a mts420-cc sensorboard including GPS-antenna.
%TODO picture of stuff
Each of them has a unique identifier (`ID`), which is used in order to link a mote to a bicycle. On our webview the user is then able to mark his bicycle as stolen. Afterwards, the `ID` is disseminated through the whole network. If the bicycle approaches a network node, it receives a dissemination packet. These are called

"pings". As the bicycle now knows that the network is in range and available, it will check, if its own `ID` is marked as stolen. If so, the GPS antenna is powered on and starts approximate 90 seconds later to save data (current runtime, latitude, longitude).
After recording data successfully and receiving another ping, the bicycle mote will dump the data into the network using the collection protocol.
%TODO: kann weg? The amount of data per packet can be easily changed within the `./nesC/DataMsg.h`, as seen in Listing 1.
Each mote has a RAM of 8kB. Therefore, it is possible to store 600 tuples on the bicycle mote RAM. As we are approximately storing one tuple every 3 minutes, it is possible to store the coordinates of the last 30h. This can be extended by saving onto the flash itself, which we left out for future work, as 30h is enough for all our testcases as well as the battery time is limited due to usage of GPS as well. %TODO batterien haben 3.4k mAh oder so, gps brauch 70 mAh

4.4 Topology

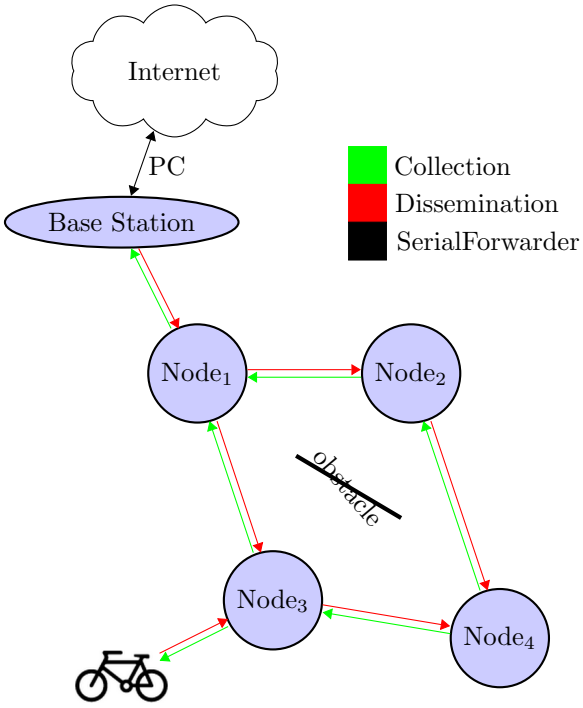


Figure 1: Example network showing the protocols used

- extensible by "nodes"

5 Conclusion

6 Future Aspects

1. PRIVACY → GPS TRACKING ??
2. BATTERY → LOAD WHILE CYCLING
3. BIKES DISSEMINATE STOLEN BIKES
4. ENCRYPTED TRAFFIC
5. flash instead of RAM
6. hidden sensor (in bike e.g., but still have connectivity)
7. use wifi instead of GPS to track location

7 Relevant codes passages... just a dump

7.1 base_station

Receiving stolen bicycle IDs from PC

```
1 //...
2 for (i=0;i<MAXBIKES;i++)
3 {
4     pkt.bikes[i]=msg->data[i*2]*256+msg->data[i*2+1];
5 }
6 //...
```

Listing 3: DataMsg.h, content of packets

7.2 base_station

Receiving stolen bicycle IDs from PC

```
1  #define MAXPOSITIONS 100
2  //...
3  uint32_t lons[MAXPOSITIONS];
4  uint32_t lats[MAXPOSITIONS];
5  uint32_t times[MAXPOSITIONS];
6  //reading
7  atomic
8  {
9      for (i=current_reading_pos;i!=current_writing_pos;i++)
10     {
11         msg->time[j] = times[i];
12         msg->lat[j] = lats[i];
13         msg->lon[j] = lons[i];
14         times[i]=0;
15         lats[i]=0;
```

```
16          lons[i]=0;
17          current_reading_pos++; //we read the value
18          if (current_reading_pos==MAXPOSITIONS)
19              current_reading_pos=0;
20
21          j++;
22          if (j==COORDS_PER_PACKET)
23              break;
24      }
25  }
26  //writing
27  atomic
28  {
29      lats[current_writing_pos]=(uint32_t)(lat*1000000);
30      lons[current_writing_pos]=(uint32_t)(lon*1000000);
31      times[current_writing_pos]=(uint32_t)((call LocalTimeMicro.get())/1000); //
32      current_writing_pos++;
33      if (current_writing_pos==MAXPOSITIONS)
34          current_writing_pos=0;
35      call Leds.led0Toggle();
36  }
37  //receiving IDs and starting gps if stolen
38  event void Value.changed()
39  {
40  uint8_t i;
41  const EasyDisseminationMsg* newVal = call Value.get();
42  bool found=FALSE;
43  pkt = *newVal;
44  for (i=0;i<MAXBIKES;i++)
45  {
46      if (pkt.bikes[i]==secret())
47      {
48          stolen=0x01;
49          found=TRUE;
50          call Leds.led1On();
51          if (gps_started==0)
52          {
53              gps_started=1;
54              call Timer.startOneShot(90000); //wait X/1000 secs
55              call GpsControl.start();
56          }
57          else if (gps_started==2) //startDone for gps
58          {
59              call Leds.led2Toggle();
60              //it is stolen AND received a broadcast
61              //=> DUMP ONE PACKET
62              sendMessage();
63          }
64      }
65  }
```

```
66 if (found==FALSE)
67 {
68     call Leds.led1Off();
69     stolen=0x00;
70     if (gps_started>1)
71     {
72         call GpsControl.stop();
73         gps_started=0;
74     }
75 }
```

Listing 4: DataMsg.h, content of packets

8 References

Dissemination and collection protocols for TinyOS: http://tinyos.stanford.edu/tinyos-wiki/index.php/Network_Protocols