Anti Bicycle Theft

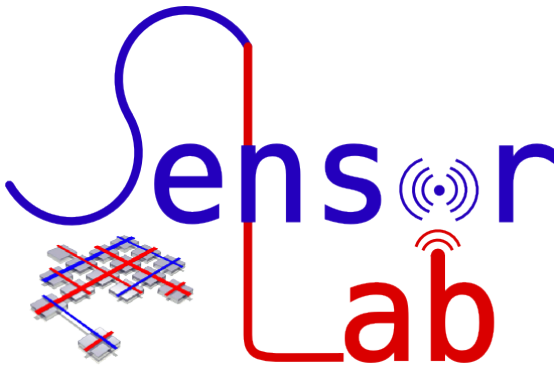
Documentation

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

January 24, 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`

3 Walkthrough

This section demonstrates the project in order to understand explanations given lateron.

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

TODO:java+python2
Collect+send data via ./python-api/env/bikeDB/listen.py
Packages needed node, mongo, python2, java(...)

3.3 Registration and Tracking

%TODO Screenshot of Views

4 Used Sensorboards and Motes

We used the standard IRIS motes for everything.... `http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf`

4.0.1 Base Station

Our 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 `ID`s of stolen bikes to the base station in order to disseminate the `ID`s through the network. On the other hand, a collection protocol is implemented to gather information from stolen bikes, like coordinates.
%TODO several picutres here (CLI, serialforwarder)

4.0.2 Network Node

The network nodes are completely omittable as they only enlarge the network. So if one wants a great availability of the network more network nodes are needed. These are connected with other network nodes and disseminate and collect the mentioned data to and from the base station.

4.0.3 Bicycle Mote

Bicycle motes are connected 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 on our webview lateron. Here, the user is able to mark his bicycle as stolen. Afterwards, the `ID` is disseminated through the whole network. If the bicycle comes in range of a network node, it receives a dissemination packets. These are called "pings", as the bicycle knows that the network is in range and available. It will therefore check, if its own `ID` is marked as stolen. If so, the GPS antenna is powered on and starts collecting data (current runtime, latitude, longitude).
After collecting data successfully and receiving another ping, the bicycle mote will dump the data into the network using the collection protocol. The amount of data per packet can be easily changed within the `./nesC/DataMsg.h`, as seen in Listing 1.

```
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

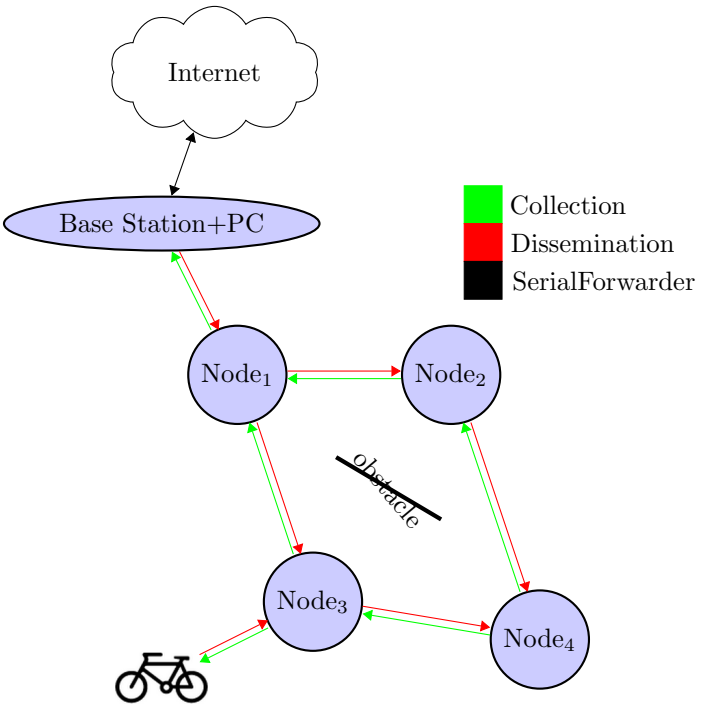
Each mote has a RAM of 8kB. Therefore, it is possible to store 600 tuples on the bicycle mote RAM. As we are approximately storing one tuple every 3 minutes, it is possible to store the coordinates of the last 30h. This can be extended by saving onto the flash itself, which we left out for future work, 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

5 Network

5.1 Protocols

5.2 Base Station and Computer Communication

5.3 Topology



- extensible by ”nodes”

5.4 Our Example Topology

insert picture here

6 Conclusion

7 Future Aspects

8 Relevant codes passages... just a dump

8.1 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 2: DataMsg.h, content of packets

8.2 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 3: DataMsg.h, content of packets

9 References

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