Anti Bicycle Theft

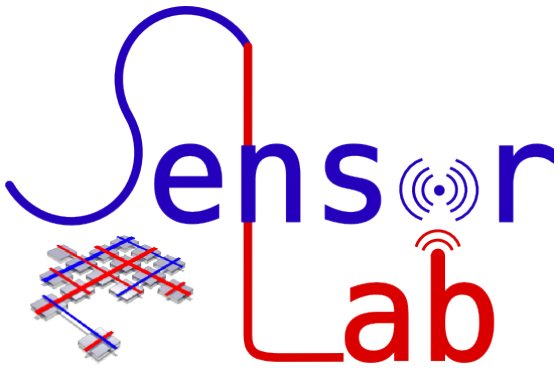
Documentation

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

January 26, 2016

Practical Course on Wireless Sensor Networks

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



Contents

1 Introduction

Bicycle theft is a major concern in many cities. Especially areas that have a high demand for bikes, like university towns, are severly afflicted. In the year 2014 more than 300.000 bikes have been registered as stolen in Germany with many more being stolen but unreported (1).
Over the last decade many systems have been developed to prevent bicycle theft or to locate stolen bikes. Most of these systems work by using a GPS module for tracking a bike and a GSM module to communicate the data via the mobile phone network (2,3). Another approach is using a Bluetooth transmitter in combination with a bluetooth and GPS equipped smartphone (4). The idea is to detect if a stolen bike is close to the mobile phone of any user of the system and use the phones positioning capability to log and communicate its approximate position to the owner.
In this %TODO paper? paper a new approach is proposed. The system envisioned by the authors is based on sensor beacons equipped with a GPS module and a close range wireless interface. This wireless interface connects though a system of wireless relay nodes to a base station that is connected to the internet. A user of this system will be able to mark a bike as stolen online. This information will then be disseminated from the base station to all relay stations, possible across several hops. As soon as a bike marked as stolen comes in reach of a relay station it will be informed that is has been stolen. This bike will start to continuously capture location information combined with a timestamp and store this information locally. From now on each time the bike comes in reach of a relay station again it will use this station to transmit its stored location data to the base station. From there the tracked positions are made available to the owner via the web interface.
This system combines advantages of the existing solutions. By using a GPS module exact locations are obtained making it more exact than using the Bluetooth approach. Also, by storing many locations and dumping them via a short range but high speed wireless connection, a very exact movement history can be created. By providing a relaying infrastructure within a certain area (e.g. a small city) the user is not depending on other users using the system (avoiding a hen-egg problem). At the same time there will be no dependency on mobile network providers and therefore fees imposed by them.

%TODO: SOURCES
Quellen: 1: `https://www.bka.de/nn_196810/SharedDocs/Downloads/DE/Publikationen/PolizeilicheKriminalstatistik/2014/pks2014ImkBericht.html?__nnn=true`
2: `http://www.spybike.com/`
3: `http://www.golem.de/news/bike-spike-gps-modul-gegen-fahrradklau-1303-98306.html` (offizielle website nicht erreichbar)
4: `https://www.cycleleash.com.au/`

2 Environment

Before presenting the project we firstly show what the whole environment looks like. Firstly, IRIS motes[1] are used (see Figure 1). Programs for the motes are written in nesC[2], compiled and flashed with tinyOS. Additionally, the computer used provides packages mentionend in Section 3 and is connected to a base station via USB Gateway, which can be seen in Figure 2. Some of the motes are connected to a MTS420CC sensorboard (see Figure 3). A GPS antenna is attached to the sensorboards.[3]

Figure 1: Standard IRIS mote without sensorboard.

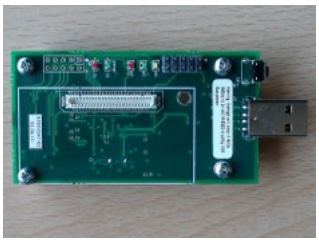


Figure 2: USB Gateway, used to forward packets via USB.

Figure 3: MTS420CC sensorboard, able to process GPS signals.

Our test environment used the following Linux packages:

1. nodejs (version 5.5.0)
2. python2 (version 2.7.11)
3. mongodb (version 3.2.0)
4. jdk7-openjdk (version 7.u95_2.6.4)

[1] A full description can be found at http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf

[2] http://www.tinyos.net/api/nesc/doc/ref.pdf

[3] Figure 1, 2 and 3 have been take from https://user.informatik.uni-goettingen.de/~sensorlab/Hardware.php

5. python2-pymongo (version 3.2)

%TODO: brauch man npm?

3 Walkthrough

This section demonstrates the project without showing technical details, as they will be explained in Section 4.

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 MTS420CC 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 MTS420CC 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 4 and 5. Once gps data is collected, he is able to review this on the same webpage, as seen in Figure 6.
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
```

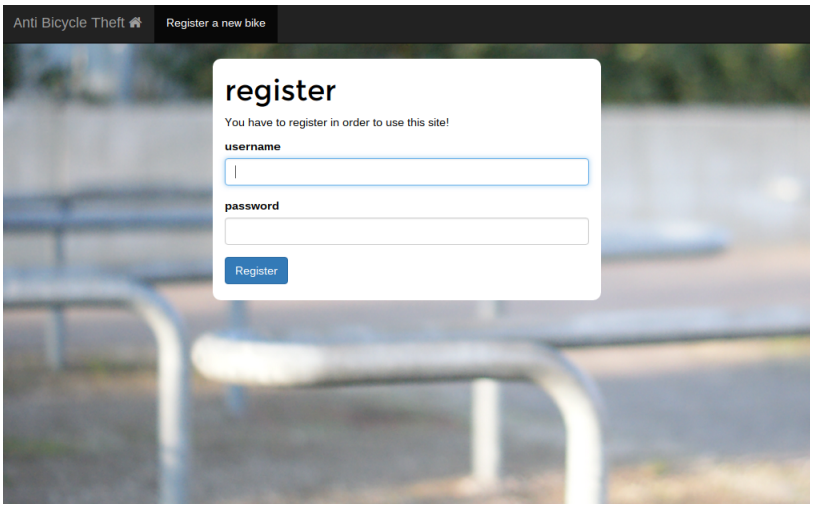


Figure 4: Registration website

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. Contrary, the propagation of stolen bicycle `IDs` is done via the Dissemination protocol. %TODO quelle again

```
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

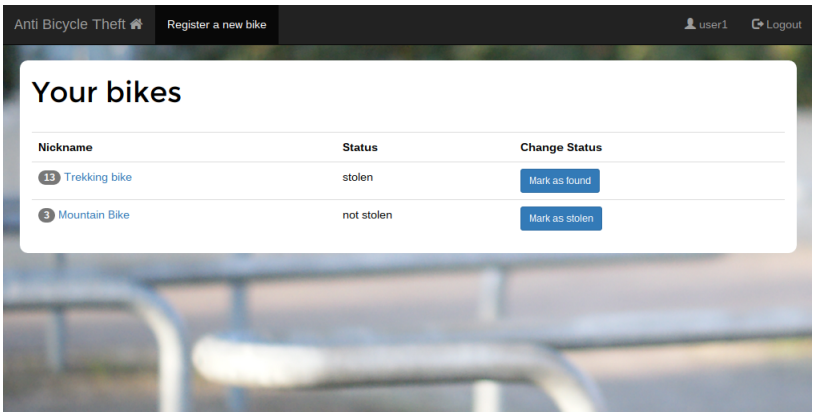


Figure 5: Viewing registered bikes

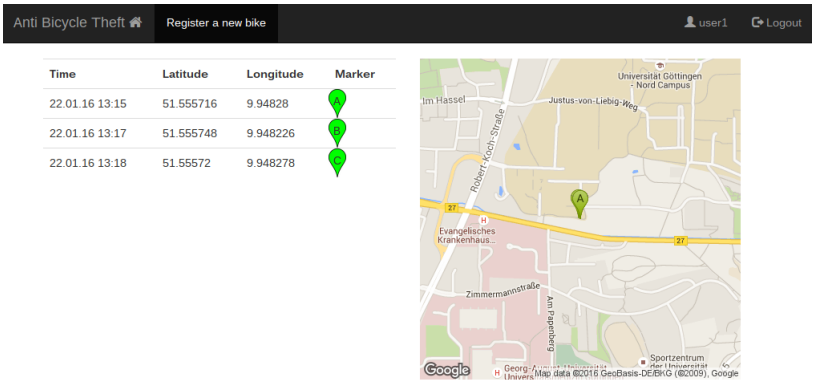


Figure 6: Viewing gathered gps coordinates

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 4. In line %TODO it imports our bikeDB class, which is simply providing methods to read and write to the database.

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 MTS420CC 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 coordinate-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 measurement 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

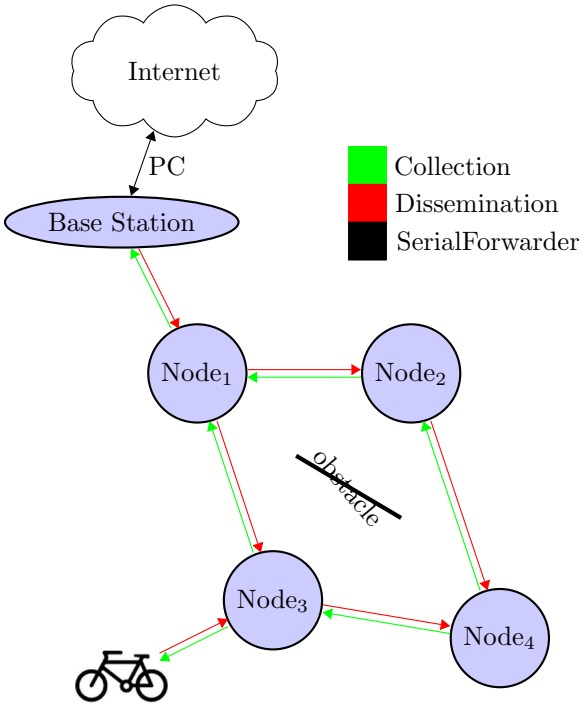


Figure 7: Example network showing the protocols used

- extensible by "nodes"

5 Conclusion

6 Future Aspects

The projects objective is to point out the possibility of a new tracking system using IRIS motes. Therefore, some additions have been left out, as they would not change the procedure, but increase the security, for example. All of these points are explained in this section.

1. **Privacy and authentication**. Obviously, admins are able to review data of users and mark specific bicycles as stolen, to start the gatherin process. Therefore, the gps data gathered could be encrypted using asymmetric cryptography. The user will receive encrypted data and decrypt it using his private key.

2. **Encrypted traffic**. Relating to the previous point, the whole traffic of the network, including dissemination of stolen bicycle IDs, should be encrypted to ensure privacy and integrity of packets.

3. **Further dissemination of stolen IDs**. Bicycle motes, regardless of their state, can save and disseminate the IDs of stolen bicycles. This way, a bicycle is able to start gathering data without connecting to the network, but to another bicycle. Obviously, one has to take care of synchronicity.

4. **Measurement flash**. Instead of just keeping the gathered gps data in the memory, one could save them to the flash of the mote and load them again when receiving a ping of the network. This provides the possibility to store a huge amount of coordinates.

5. **Battery life**. When implementing the previous mentioned point, the battery life has to be extended as well as GPS needs a lot of battery. However, one could use two approaches:

   (a) A stolen bicycle is going to be used. Therefore, one could power the mote using the energy of a dynamo attached to the bicycle.

   (b) At some places, for example a cellar, the gps antenna is not able to maintain a connection. Hence, the gps antenna could be powered off after e.g. 1 minute without connection. Afterwards, start the gps again after a small amount of time and check for connection again.

6. **Hiding of the mote**. The thief may not see and be able to remove the mote from the bicycle. So the mote has to be hidden somewhere in the frame of the bicycle without losing signal.

7. **GPS exchangeability**. As GPS drains a lot of battery, one could think of implementing a wifi scanner instead of GPS. As wifi is widely used, nearly everywhere wifi networks can be found. Using the SSIDs only (without connecting to them) one is able to specify the current location. This obviously consumes less power than gps and will even work, when a bicycle is e.g. in a garage, where a gps antenna will not be able to establish a connection.

%TODO gps VS GPS

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 2: 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 3: DataMsg.h, content of packets

```
1  import subprocess
2  import time
3  import bikeDB
4  import datetime
5  import thread
6  import os
7
8  maxBikes=10
9
10 def unsecret(secret):
11     return (secret-9)/7
12
13 def send(bDB):
14     while True:
15         stolen=bDB.getIdsOfStolen()
16         cmd="java net.tinyos.tools.Send "
17         pkt="00 FF FF 00 04 %02x FE 2A "
18         packetlength=2
19         stolen_str=""
```

```
20          for b in stolen:
21              stolen_str+="%02x_%02x_"%(int(b)/256,int(b)
                    %256)
22              packetlength+=2
23          for i in range(maxBikes-len(stolen)):
24              stolen_str+="00_00_"
25              packetlength+=2
26          if packetlength>2:
27              pkt = pkt % packetlength
28              pkt = pkt + stolen_str
29              cmd = cmd + pkt.upper()
30              print("send:_{}".format(pkt))
31              print(cmd)
32              os.system(cmd)
33
34          time.sleep(3);
35
36
37  def recv(bDB):
38      #p = subprocess.Popen(["java net.tinyos.tools.Listen
            -comm serial@/dev/ttyUSB1:iris &"], stdout=
            subprocess.PIPE, shell=True)
39      p = subprocess.Popen(["java_net.tinyos.tools.Listen_&
            "], stdout=subprocess.PIPE, shell=True)  # use
            this WITH serialForwarder
40      COORDS_PER_PACKET=2
41      LENGTH_OF_PACKET=(2+4+4+4)*2 #in nibbles
42      while True:
43          pkt=p.stdout.readline().strip("_\n\r\t").lower()
                #reads until \n
44          pkt=pkt.replace("_","")
45          print("recv:_{}".format(pkt))
46          if (pkt[30:32]=="ee"): #dissemination ID
47              nodeid=pkt[32:36]
48              print "sent_by:",unsecret(int(nodeid,16)),"(%
                    s)"%nodeid
49              runtime=int(pkt[36:44],16)/1000.0
50              print "runtime:",runtime
51              offset=44
52              for i in range(0,COORDS_PER_PACKET):
53                  t=int(pkt[offset+i*8:offset+i*8+8],16)
                        /1000.0
54                  lat=int(pkt[offset+i*8+COORDS_PER_PACKET
                        *8:offset+i*8+COORDS_PER_PACKET
                        *8+8],16)/1000000.0
55                  lon=int(pkt[offset+i*8+COORDS_PER_PACKET
                        *8+COORDS_PER_PACKET*8:offset+i*8+
                        COORDS_PER_PACKET*8+COORDS_PER_PACKET
                        *8+8],16)/1000000.0
56
```

```
57                      print "time:",t
58                      print "lat:",lat
59                      print "lon:",lon
60                      timestamp=datetime.datetime.fromtimestamp
                            (time.time()-int(runtime-t)).isoformat
                            ()
61                      print "timestamp:",timestamp
62                      print "insert:","%d"%int(nodeid,16),
63                      print lat,lon,timestamp
64                      if (lat<99 and lon<99 and lat!=0.0 and
                            lon!=0.0):
65                          bDB.insertPosition("%d"%int(nodeid
                                ,16),str(lat),str(lon),timestamp)
66                      #lat = pkt[
67
68
69  if __name__ == "__main__":
70      bDB = bikeDB.BikeDB()
71      thread.start_new_thread(send,(bDB,))
72      recv(bDB) # blocking
```

Listing 4: listen.py, ...

8 References

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