Materialien für

LoRaWAN – Workshops
- Sensorbau
- GPS-Tracker

Technologiestiftung Berlin, 17.01.2019 // 23.07.2019

Dr. Christian Hammel, Technologiestiftung Berlin, 2019

THE THINGS NETWORK

TECHNOLOGIE STIFTUNG BERLIN

Inhalt

Dieser Foliensatz enthält Material für die Durchführung von Workshops zum Thema

LoRaWAN mit dem Arduino für Neueinsteiger

Für Workshopanbieter*innen und Selbstlerner*innen: https://github.com/technologiestiftung/workshops

Workshop 1 (Dauer: 2,5 bis 3 Stunden (Selbstlernen: 2 Stunden mehr), Seiten 4 - 20)
Bau eines Temperatursensors, Anschluss an Thethingsnetwork (TTN), Senden der Temperatur
Zielgruppe: Neueinsteiger*innen Arduino und LoRaWAN
Lerninhalte:
Funktion TTN und Internet of Things, Basiswissen Arduino, Sensoren und Breadboards, Basiswissen Airtime, Modifizieren/Hochladen von Code, Ansehen der Daten / Abruf der Daten mit MQTT

Workshop 2 (Dauer 2,5 – 3 Stunden, Seiten 21 ff.)
Bau eines Trackers, der GPS-Daten über seine Position per LoRaWAN sendet
Zielgruppe: Neueinsteiger*innen GPS und LoRaWAN
Lerninhalte:
GPS, serielle Kommunikation, Funktion LoRaWAN-Library, Timing auf dem Arduino, Stromspar-Technik „Senden bei Bedarf“

Detail-Erläuterungen stehen als Kommentar im Code selbst!

Danke für Beiträge aus den Erprobungsphasen an C. Clausnitzer, L. Lakritz und B. Seibel
Code-Quellen im Code

Hinweise für Veranstalter

Für die Workshops

LoRaWAN mit dem Arduino für Neueinsteiger

benötigt Ihr folgende Voraussetzungen am Veranstaltungsort:

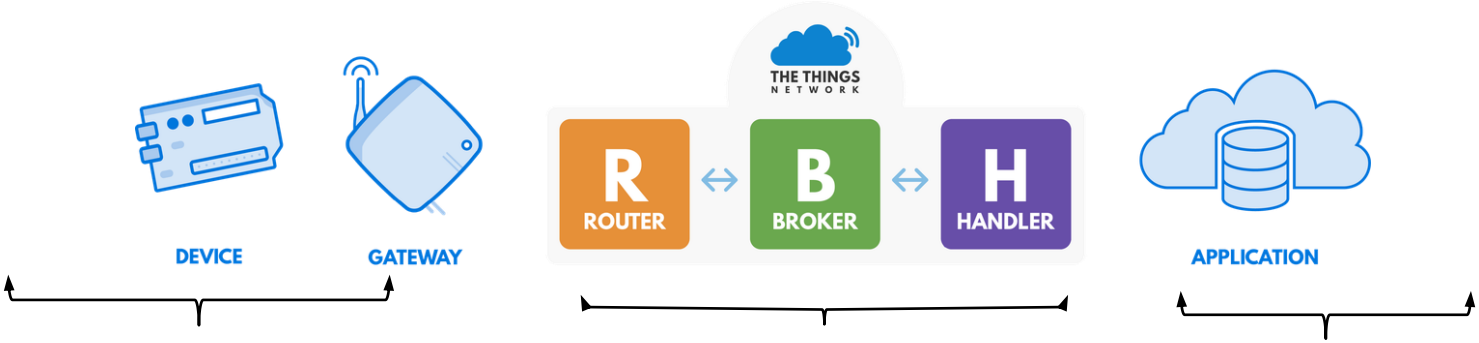
Teilnehmer*innenseitig
Laptop (älteres Modell reicht aus) mit einer Arduino-IDE (Version >1.7), die die Teilnehmenden vorab installiert haben (Installation zu Veranstaltungsbeginn dauert zu lange).
Teilnehmende sollten sich vor der Veranstaltung einen TTN-Zugang beschafft haben.

Material
Arduino, Dragino LoRaWAN-Shield (für Workshop2 mit GPS), Breadboard, Kabel, TMP36 Sensor
Alles auch mit der Hackingbox entleihbar..

Bauseitig
WiFi zum Herunterladen des Codes. Verfügbarkeit von TheThingsNetwork (Gateway entleihbar, benötigt am Veranstaltungsort Internet (Ethernet, DHCP, offener Port 1.700/UDP)).
WS 2: GPS muss empfangbar sein. Tief indoor oder hinter metallisierten Fensterscheiben wird der Workshop wird keinen Spaß machen.
Sitzgelegenheiten, Beamer und das Übliche wisst ihr selbst.

TECHNOLOGIE STIFTUNG BERLIN

Komponenten im TTN-Netzwerk



Die zentrale Struktur des TTN-Netzwerkes kann jeder nach Anmeldung kostenlos nutzen. Sie bietet Geräte-Keys, Nutzerverwaltung, Paketrouting, End-zu-End-Verschlüsselung (AES-128), API (HTTP, MQTT) und einige fertige Integrationen für andere Softwaren.
„Fair use“ ist: 30s Uplink pro Tag und Node und 10 Messages Downlink pro Tag und Node.

Zusätzlich gibt es kostenpflichtige Leistungen der TTN Industries wie private Server, Integration in Kundensysteme,..

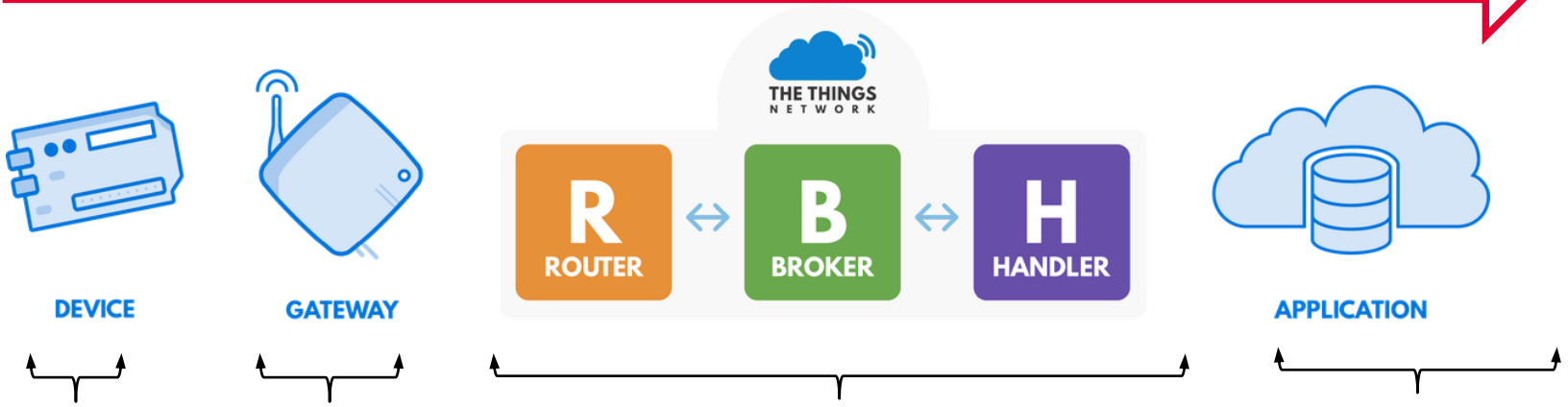
LoRaWAN-Sprech: Device = Mote = Node = Endgerät; Gateway = Basisstation (= Router)

Die Funkregulierung erlaubt LoRaWAN genau 1% Airtime!... Also lange Sende-Intervalle wählen!

Bild: Johan Stocking (TTN) / eigene Darstellung

Worum geht es heute? Was machen wir?

Wir übertragen selbst gemessene Daten vom Arduino ins Netz (uplink)



Unser Arduino!

vorhanden

Alle Teilnehmer sollten einen Account haben, damit sie ihr node anmelden und auf ihre Daten zugreifen können.

Wenn wir Zeit haben, lesen wir unsere Daten mit einem MQTT-Viewer.

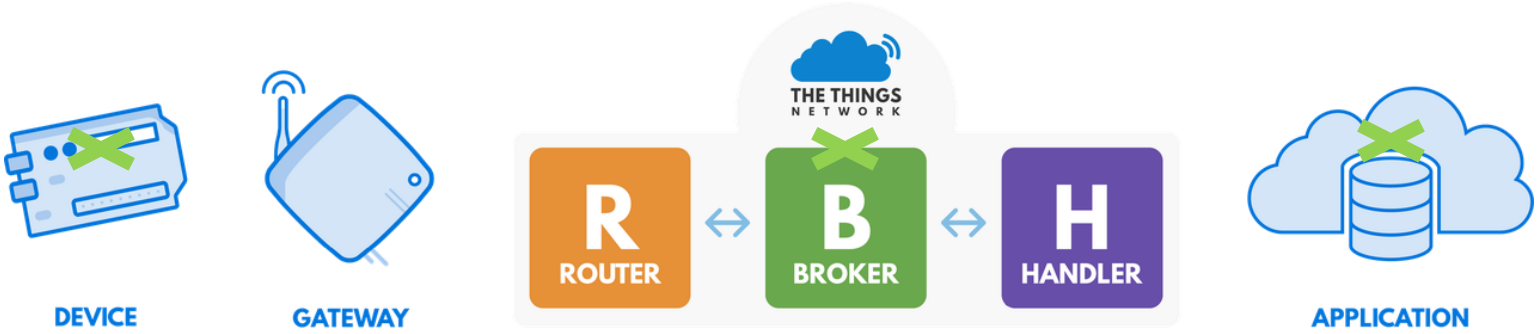
Zuerst klären: Wie melde ich den Arduino im TTN Netzwerk an? Wo sehe ich meine Daten?

Workshop-Ziele heute: Wir schließen einen Sensor an. Wir messen eine Temperatur. Wir melden das Gerät bei TTN an. Wir senden die Temperatur ins TTN. Wir nutzen Beispielcode.

Bild: Johan Stocking (TTN) / eigene Darstellung

Was genau soll unser Device machen?

Wir übertragen selbst gemessene Daten vom Arduino ins Netz (uplink)



Unser Arduino
- Misst die aktuelle Temperatur mit einem Sensor
- Überträgt die Temperatur (genauer: Bytes) ins TTN (genauer: zum TTN-Broker)
- Optional: Blinkt mit einer Kontroll-LED, wenn die Temperatur über 30° ist, damit wir direkt sehen können, ob unsere Schaltung und unser Code funktionieren.

TTN
- Zeigt uns in der Ansicht „Console“ die von uns gemessene Temperatur an, nachdem wir übertragene Bytes in brauchbare Daten zurückverwandelt (decodiert) haben.

Application MQTT-Viewer
- Zeigt die Daten im Browser oder auf dem Smartphone an, wenn noch Zeit ist

Bild: Johan Stocking (TTN) / eigene Darstellung

Board zusammenbauen

1. **LoraRaWAN-Shield auf den Arduino montieren**
   (vorsichtig so aufstecken, dass er passt und dass die Anschlüsse auf Arduino und Shield in Deckung sind)

2. **ANTENNE ANSCHRAUBEN !**
   (NIE ohne Antenne betreiben, das kann das Board zerstören)
   (NIE ohne Antenne betreiben, das kann das Board zerstören)

Der Arduino führt reinen Maschinencode aus, den man mit der Arduino-IDE aus menschenlesbarem Code erzeugt (=> Folgeseiten). Er hat viele Anschlüsse für digitale Signale und zum Auslesen analoger Spannungen.

Einstieg in die Arduino-Plattform:
https://www.arduino.cc/en/Tutorial/HomePage

Arduino-Befehlsreferenz:
https://www.arduino.cc/reference/de/

Mehr über das LoRaWAN-Shield: http://wiki.dragino.com/index.php?title=Lora_Shield
und in der GPS-Version: http://wiki.dragino.com/index.php?title=Lora/GPS_Shield

Arduino-Code

1. **Arduino-Code:** Die Programmieroberfläche (IDE) kompiliert den c-ähnlichen Code in Maschinencode, der dann auf den Arduino geladen wird. Deshalb kann man Code auch nicht vom Arduino herunterkopieren.

2. **Programmaufbau:** Es gibt immer mindestens drei Teile,
   - einen Header, in dem Includes, Variablen u.ä. definiert werden,
   - einen Teil void setup (){}, der genau einmal abgearbeitet wird und
   - einen Teil void loop (){}, der ständig wiederholt wird.

   Man kann an beliebiger Stelle eigene Funktionen ergänzen, die sich aus anderen Programmteilen heraus aufrufen lassen.

3. **Syntax**
   Funktionen: void FUNKTIONSNAME (Rückgaben) {Code der Funktion}
   Befehlszeilen: enden immer mit **;**
   Kommentar: wird mit // eingeleitet und nicht kompiliert
   Konstanten und Variablen: Sind immer 2 Byte (16 Bit), wenn man nichts anders definiert.

4. **Prográmmchen (Sketches)**
   Konvention: enden auf .ino
   stehen in einem Ordner namens Arduino-Sketchbook in eurem Userverzeichnis.

5. **Beispiele für heute:** https://github.com/technologiestiftung/workshops

TECHNOLOGIE STIFTUNG BERLIN

TEST Kommunikation zwischen Laptop und Arduino

1. **Arduino-IDE starten**

2. **Arduino über USB-Kabel anschließen**
   (NIE ohne Antenne!)

3. **Kommunikationstest**
   „Werkzeuge | Board | Mega“, „Werkzeuge | Prozessor | Mega2560“; „Werkz. | Port | $richtigerPort“;
   „Werkzeuge | Boardinformationen holen“
   Wenn etwas kommt: alles richtig angeschlossen,
   Wenn nicht oder Fehler: richtiger Port? Strom am Arduino? USB-Anschluss freigegeben (Linux)?

4. **LMIC (LoRaWAN-Bibliothek einbinden)**
   „Sketch | Bibliothek einbinden | IBM LMIC framework“
   wenn das nicht funktioniert: ZIP-file von: https://github.com/matthijskooijman/arduino-lmic

5. **Testcode hochladen**
   Testcode der TSB (kommunikationstest_lappi_arduino.ino) in die IDE laden; „Sketch | hochladen“
   Keine Fehlermeldungen? Alles prima!
   Dragino mit GPS und Fehlermeldung in diesem Stil:
   ```
   avrdude: stk500v2_ReceiveMessage(): timeout
   avrdude: stk500v2_getsync(): timeout communicating with programmer
   ```
   GPS-Reset-Knöpfchen während Hochladen gedrückt halten

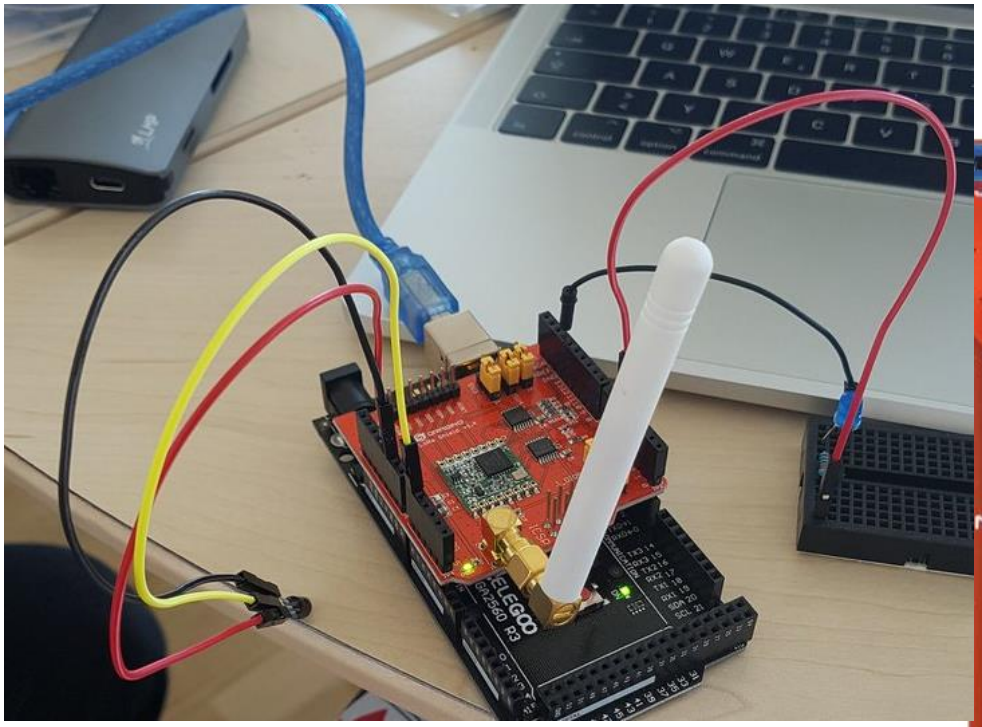
6. **Test**
   „Werkzeuge | serieller Monitor“:
   Wenn bei 115200 baud sinnvoller Output kommt, ist alles prima.

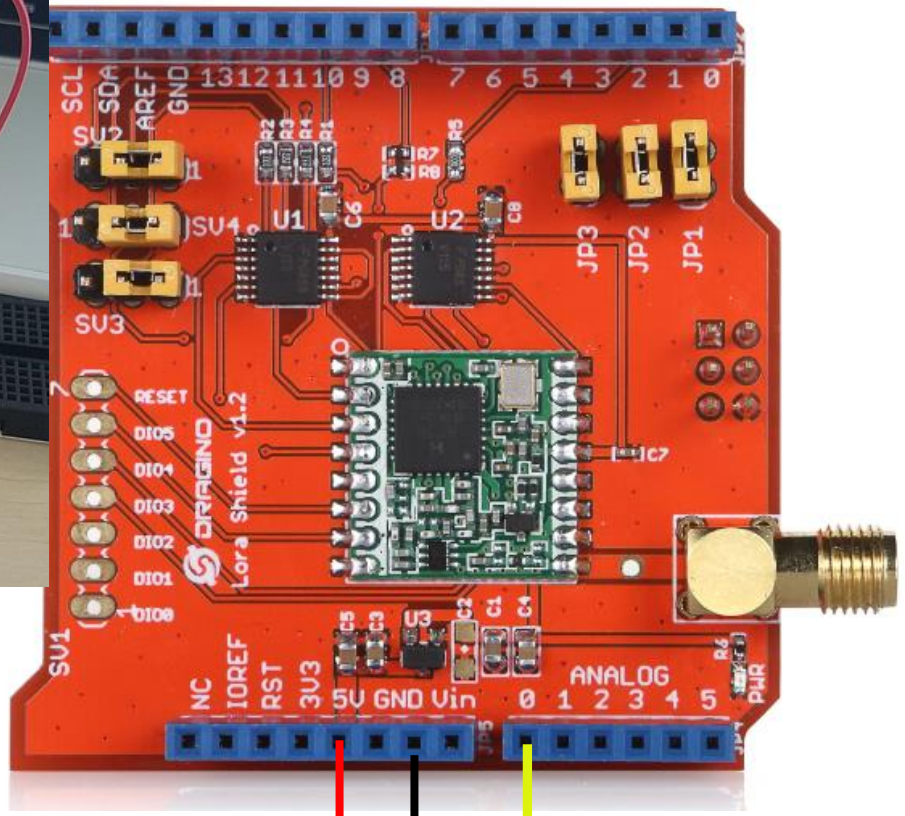
TECHNOLOGIE STIFTUNG BERLIN

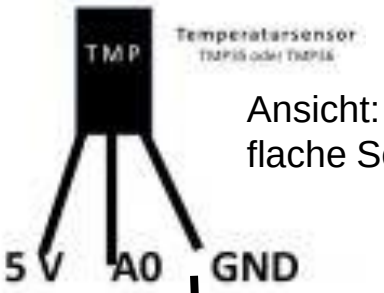
Schaltung aufbauen: Sensor

Sensor am Arduino anschließen:

- GND an GND (Analogseite)
- 5V an 5V (Analogseite), Konvention: rot an +
- Messausgang auf Analogeingang 0







Ansicht:
flache Seite oben

Bild: Technologiestiftung, Dragino

TECHNOLOGIE STIFTUNG BERLIN

Arduino bei TTN anmelden, Daten senden und decodieren

1. **Bei TTN einloggen (wer keinen Account hat, muss sich im Web bei TTN einen anlegen)**
   TTN-Konsole:
   Anwendung anlegen,
   device anlegen,
   Settings, ABP, generiert Schlüssel und Device-ID

2. **Arduino IDE starten**
   TSB-Mustercode (temperatur_basteltreff.ino) in die IDE **aber noch nicht auf den Arduino** laden
   (die Schlüssel im Beispiel sind fake als Syntaxbeispiel. Sie produzieren Fehler.).
   Die Schlüssel Network Session Key, Appskey (msb) und Device ID (hex) (≠ EUI !) in den Mustercode einpflegen (Schlüsselformat mit <-> umschalten, ganz rechts: Kopier-Button).
   Eine beliebige interne ID könnt ihr einbauen, wenn ihr mehrere Nodes auseinanderhalten wollt.

3. **Code hochladen**
   Wenn eure Schlüssel richtig eingetragen sind, dann seht ihr im seriellen Monitor Daten, die ihr erwartet und in eurer Konsole Daten, die noch nicht sinnvoll sind.
   Wenn die optionale Kontroll-LED angeschlossen ist und der Sensor >30° warm ist, geht die LED an.

4. **Payload-Decoder einrichten**
   Damit die an TTN gesendeten Daten wieder in eine sinnvolle Form kommen, legt ihr über die TTN-Konsole einen payload-decoder an. Code dafür (javascript) steht als Kommentar im Arduino-Testcode.

5. **Code verstehen**
   versuchen wir direkt im Code, dafür sind Kommentare im Code.

Daten ansehen und prüfen

1. **Direkt bei TTN**
   TTN-Konsole: in eurer Application / eurem Device könnt ihr die Daten live sehen.
   Wenn der Decoder funktioniert, werden sie auch entschlüsselt.

2. **Von TTN auslesen**
   über http: geht, sprengt hier aber schnell den Rahmen
   über MQTT: für diesen Standard gibt es einige fertige Viewer.

3. **MQTT**
   liefert Datenfelder, die ein Viewer „abonniert“ und anzeigt, nutzt üblicherweise tcp auf Port 1883.
   Ein Datenfeld hat Name und Wert und heißt in MQTT-Sprech „Topic“.
   Da wir eine End-zu-End-Verschlüsselung haben und auf das Application-Ende nicht jeder Zugriff haben soll, braucht der Viewer einen Schlüssel/password zum Zugriff auf die Daten.
   Unsere Topics sehen so aus: **AppID/devices/deviceID/up (/EUERDATENFELD**)
   Wenn ihr viele devices habt, sollte das so aussehen: AppID/devices/+/up (/EUERDATENFELD)
   Zugang: Host: eu.thethings.network ; Port: 1183 ; Username; App-ID ; Password: App-Access Key

4. **MQTT mit Android-Apps: MQTT Dash**, **Linear MQTT Dashboard**, MyMQTT
   (Tipp: App-Access-Key abtippen geht immer schief!
   Lösung: Browser oder die App TTN Mobile können copy und paste.)

5. **MQTT mit Chrome:** Plugin MQTT-Lens

Daten nutzen: Live-Graph mit LinearMQTT (Android)

1. **Sicherheit und Privacy**
   Ihr installiert eine App und gewährt ihr Rechte. Der Hersteller „ravendmaster“ ist lt. Impressum in Russland .
   Wem das zu dubios ist, der sollte es nicht nutzen.

2. **MQTT-Verbindung herstellen**
   “**Menu**”-Symbol oben rechts; App-Settings
   Server: tcp://eu.thethings.network ; Port: 1883 ; Username: AppID ; User password: App-Access Key (den zeigt die TTN-Konsole an); Client ID: generiert die App selbst
   (Tipp: App-Access-Key abtippen geht immer schief! Browser oder TTN Mobile können copy und paste)

3. **Graph-Widget basteln**
   „**Menu**“-Symbol oben rechts; Tabs ; „+“ ; Tabnamen vergeben; zum neuen Tab wechseln; „+“-Symbol legt ein Widget an; Widget-Type: „Graph“; „Live“; „Name“: beliebig, wird später so angezeigt; Sub.Topic: : AppID/devices/deviceID/up/EUERDATENFELD ; SAVE
   EUERDATENFELD heißt Temperatur, wenn ihr unser Beispiel nicht geändert habt

   Dieses Widget zeigt Eure Temperatur live als Graph an.

Daten nutzen: blinkendes Widget mit MQTTDash (Android)

1. **Sicherheit und Privacy**
   Ihr installiert eine App und gewährt ihr Rechte. Der Hersteller „routix.net“ hat kein Impressum, auch keinen adminC im whois. Die Domain ist in Russland registriert, dort steht auch der Server (tracert).
   Wem das zu dubios ist, der sollte es nicht nutzen.
   Entzug aller Berechtigungen, die man mit Android6 entziehen kann: funktioniert trotzdem.

2. **MQTT-Abonnement anlegen**
   „**+**“-Symbol oben rechts; „Name“: beliebig, wird später so angezeigt; Address: eu.thethings.network ; Port: 1883 ; Username: AppID ; User password: App-Access Key (den zeigt die TTN-Konsole an); Client ID: generiert die App selbst
   (Tipp: App-Access-Key abtippen geht immer schief! Browser oder TTN Mobile können copy und paste)

3. **Widget basteln**
   Name: ihr wählt einen namen für euer Widget, Topic: AppID/devices/deviceID/up/EUERDATENFELD
   EUERDATENFELD heißt Temperatur, wenn ihr unser Beispiel nicht geändert habt.
   Folgefelder: Enable, Update, Large, QoS(0),(Blink: val > 29)

Daten ansehen mit MQTT-Lens (Chrome-Plugin)

1. **Sicherheit und Privacy**
   Ihr installiert eine Chrome-Erweiterung App und gewährt ihr Rechte! Die privacy von Chrome ist umstritten. **Wem das zu dubios ist, der sollte es nicht nutzen.**

2. **Leistung:** MQTT-Lens zeigt eure Datenfelder vollständig an (nicht sehr schön aber lesbar), schicke Visualisierungen kann es nicht, außerdem startet es auf älterer Hardware sehr langsam und stürzt gerne mal ab. Dafür kann man damit über MQTT auch Messages zum device senden.
   Achtung: geht nur, wenn der Port 1883 nicht gesperrt ist!

3. **MQTT-Verbindung anlegen**
   „Connections | „+“-Symbol; „Name“: beliebig, wird später so angezeigt; Hostname: tcp, eu.thethings.network ; Port: 1883 ; Client ID: generiert die App selbst; Username: AppID ; User password: App-Access Key (den zeigt die TTN-Konsole an);
   (Tipp: App-Access-Key abtippen geht immer schief! Copy + paste über den Browser klappt!); Last Will: leer lassen; „create“

4. **MQTT-Abo anlegen**
   Die eben angelegte Verbindung anklicken, „Topic“ (= Feld mit euren Daten) abonnieren (subscribe).
   Topic: AppID/devices/deviceID/up zeigt alle Daten eures devices an.
   Topic: AppID/devices/deviceID/up/EUERDATENFELD zeigt nur das gewünschte Feld.
   EUERDATENFELD heißt Temperatur, wenn ihr unser Beispiel nicht geändert habt.
   Mit “publish” könnte man auch selbst MQTT senden.

Coole Anwendungen? Tolle Bildungsmaterialien?

Beispiel für eine schicke LoRaWAN-Anwendung von Bürgern:

- Umweltsensoren: https://skopjepulse.mk/

Wer mit uns Bildungsmaterial und Unterlagen für die Hackingbox IoT entwickeln mag, ist herzlich willkommen.

Carolin Clausnitzer
clausnitzer@technologiestiftung-berlin.de

oder
Christian Hammel
hammel@technologiestiftung-berlin.de

TECHNOLOGIE STIFTUNG BERLIN

Anhang 1: Linux-User

1. **Arduino-IDE installieren:**
   Die Paketquellen von ubuntu und Derivaten enthalten nur eine veraltete Version. Eine aktuelle Version findet Ihr bei Arduino selbst:
   https://www.arduino.cc/en/main/software

2. **Arduino-IDE unter Linux: Zugriff auf serielle Schnittstelle herstellen**
   Herausfinden, zu welcher group die zugelassenen Schnittstellenuser gehören:
   `ls -l /dev/ttyUSB*` oder `ls -l /dev/ttyACM*` liefert etwas in der Art:
   `crw-rw---- 1 root tty 188, 0  5 apr 23.01 ttyUSB0`
   Die Angabe in der vierten Spalte (hier „tty“) zeigt, welche Nutzergruppe Zugriff hat.
   Dieser Gruppe müssen wir noch beitreten (<username> ist unser Linux-Benutzername):
   `sudo usermod -a -G tty <username>`
   Jetzt noch mal neu starten, dann sollte der Zugriff klappen.

Anhang 2: optionale Kontroll-LED

Wenn plausible Temperaturdaten im TTN ankommen funktioniert alles und niemand braucht eine Kontroll-LED.

Wenn das nicht der Fall ist, ist die erste Wahl als Prüfwerkzeug der serielle Monitor der IDE, mit dem man prüfen kann, was der Sensor eigentlich macht und misst.

Wenn der Sensor mit einem Laptop nicht zugänglich ist, kann die beschriebene Kontroll-LED dabei helfen, herauszufinden, ob der Sensor überhaupt plausible Daten liefert. Im Code sind 30° C eingestellt, ab denen sie anfängt zu blinken, da man 30° noch mit dem Finger erreichen kann.

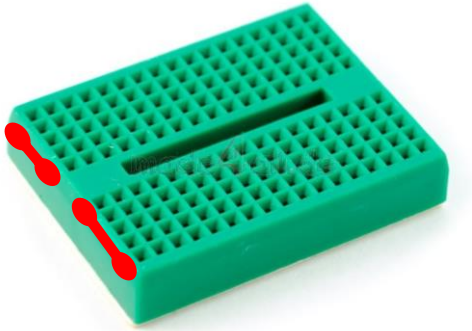
Anhang 2: optionale Kontroll-LED

Kontroll-LED: Arduino und Breadboard:

- Pin GND (auf der „Digitalseite“ vom Arduino –> Vorwiderstand –> kurzes LED-Bein (sw. Kabel, wenn da)
- Pin 5 (Arduino, Digitalseite) -> langes LED-Beinchen (rot für „+“, wenn verfügbar)

Mit der LED kann man auch ohne Computer testen, ob der Sensor richtig misst.

Beim Steckbrett sind einzelne Anschlüsse miteinander verbunden



Diese 5 Anschlüsse sind verbunden.

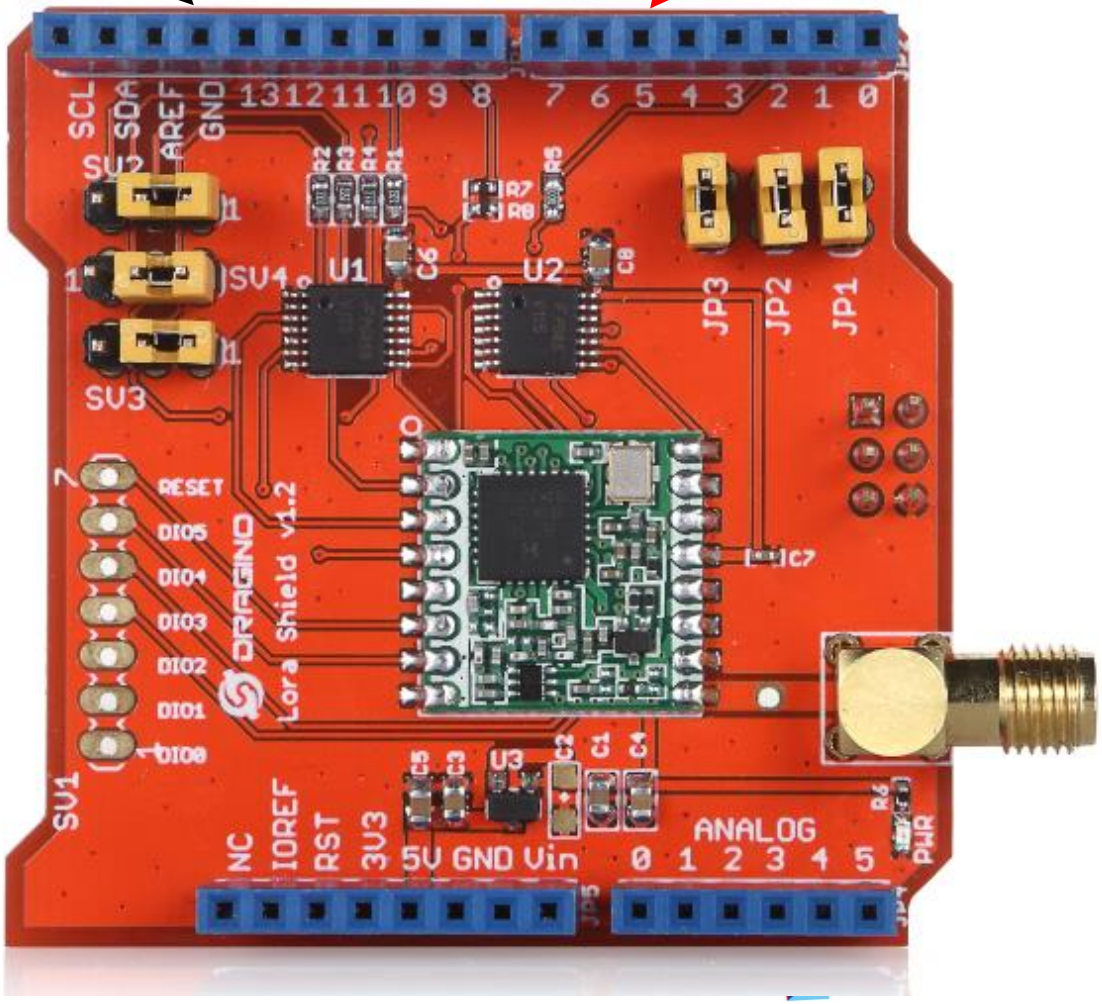


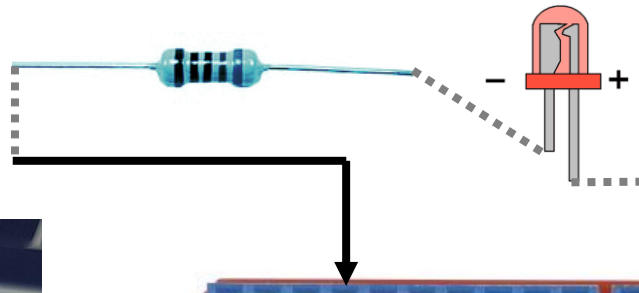
Bild: Technologiestiftung, Dragino

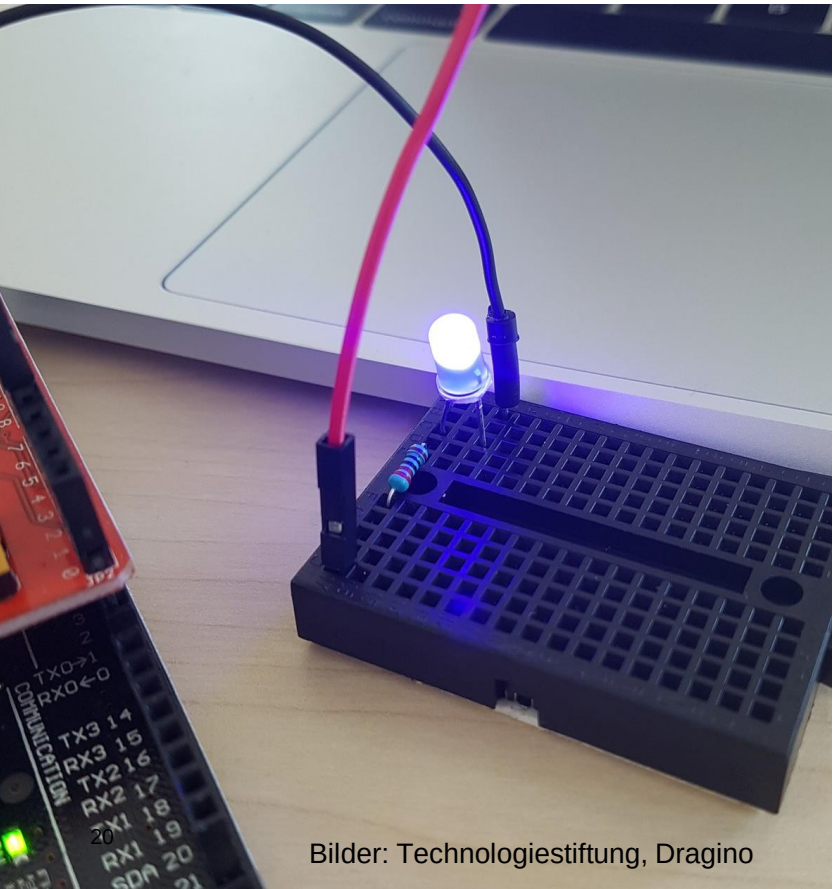
Anhang 2: optionale LED mit Breadboard verkabeln

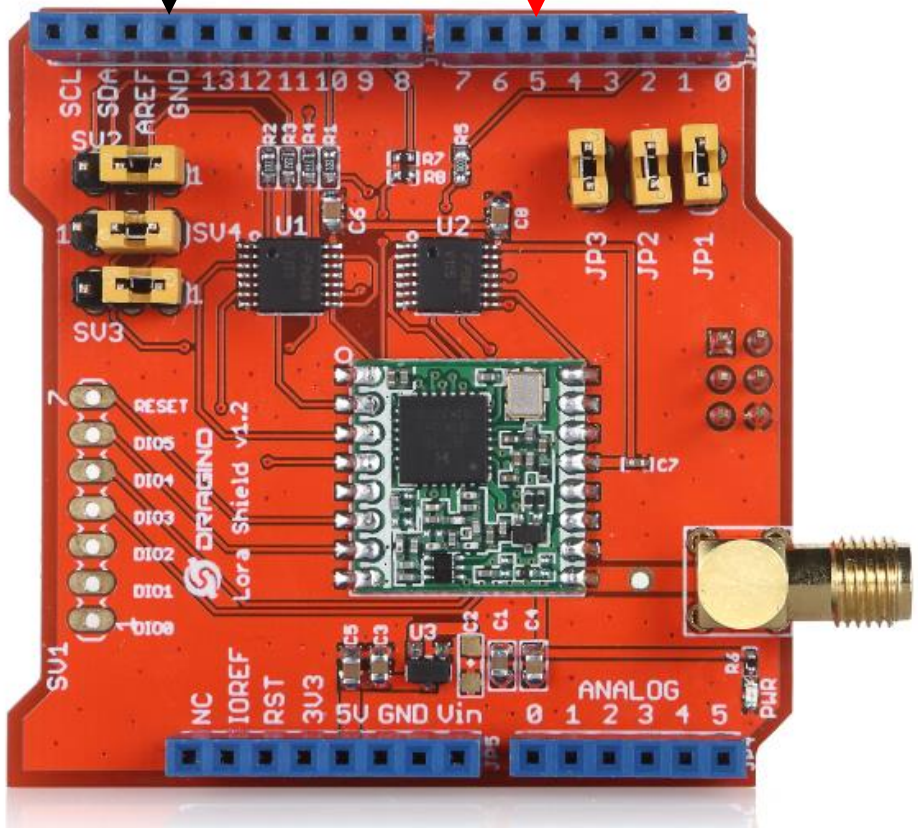
Kontroll-LED: Arduino und Breadboard:

- Pin GND (auf der „Digitalseite“ vom Arduino –> Vorwiderstand –> kurzes LED-Bein (sw. Kabel, wenn da)
- Pin 5 (Arduino, Digitalseite) -> langes LED-Beinchen (rot für „+“)

..... im Breadboard







Bilder: Technologiestiftung, Dragino

Workshop 2: GPS-Tracker

Wir lesen GPS-Daten aus einem Shield mit GPS-Empfänger aus
Wir lernen, wie GPS und wie serielle Kommunikation funktionieren.

Wir übertragen GPS-Sätze als Bytes und lernen, dass man im IoT so wenig Daten überträgt wie möglich.

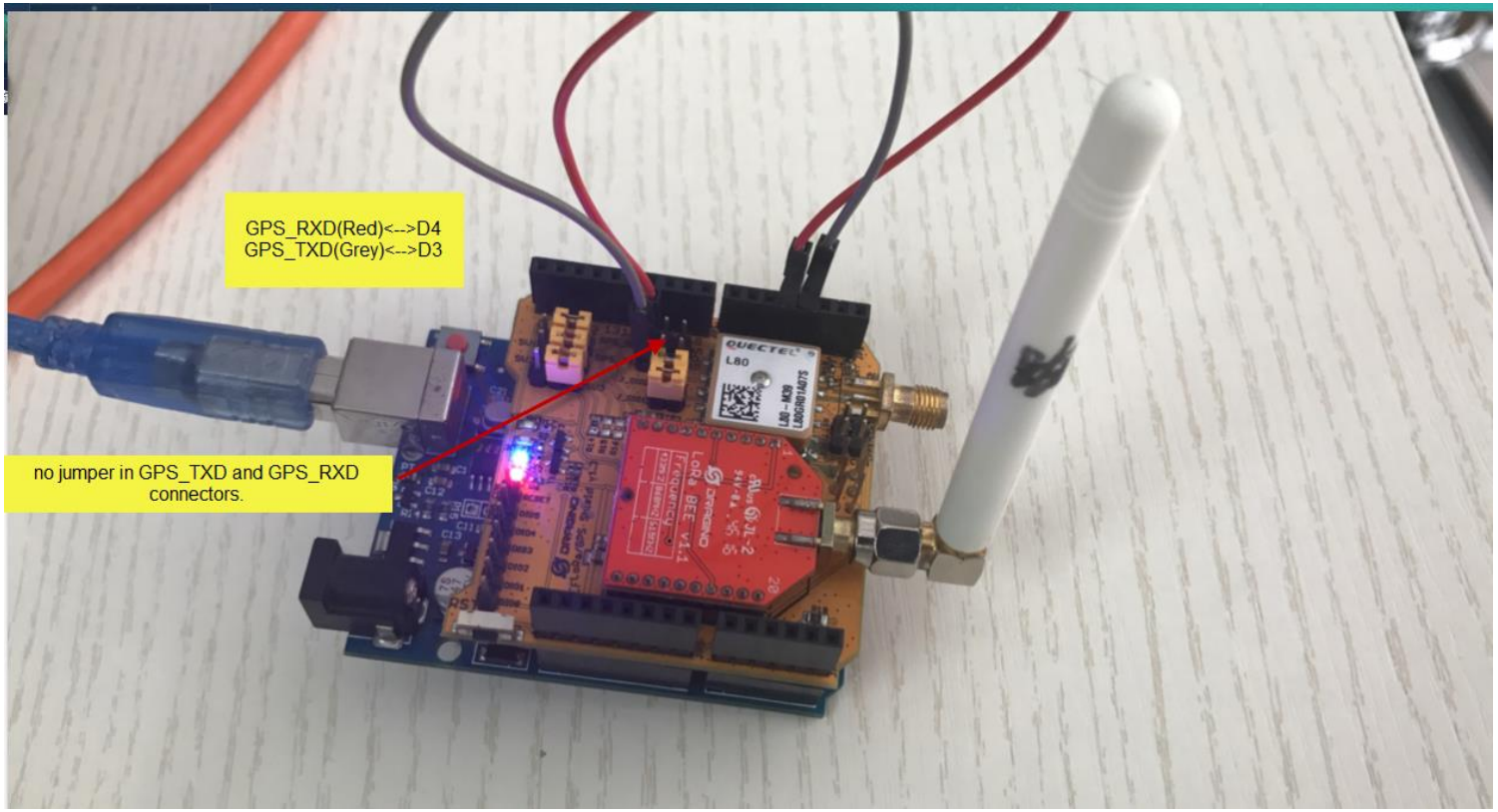
Wir prüfen Daten auf dem Arduino und senden nur, wenn Daten gültig und sind. Wir senden daurch seltener und sparen Strom. Außerdem lernen wir die LoRaWAN-Library näher kennen und bauen uns Timer.

Hinweis: Als batteriebetriebener Tracker zum Dauereinsatz sind die Arduinos nur bedingt geeignet. Sie sind zu groß, um sie z.B. in einem Fahrradrahmen zu verstecken, außerdem sind die Stromsparfähigkeiten des Arduinos sehr begrenzt. GPS lernen und ausprobieren kann man damit aber gut.

TECHNOLOGIE STIFTUNG BERLIN

Bedienung Arduino, Arduino-IDE und Aufbau des Boards

Modifizieren und Hochladen von Code, Montage Shield, Anschluss Laptop: Wie in Workshop 1



Arduino Uno: Zwei Jumper entfernen (nicht wegwerfen!), GPS-Rx mit Pin3 und GPS-Tx mit Pin 4 verbinden (darüber lesen wir seriell aus).

Arduino Mega: Rx an D18 und Tx an D19 auf dem Mega-Board

GPS-Fix vorhanden: LED auf dem Shield blinkt

Bild: Edwin Chen, Dragino

TECHNOLOGIE STIFTUNG BERLIN

Funktionsweise GPS-Empfänger

Positionsbestimmung mit GPS (hier als Überbegriff für GPS, GLONASS, Baidou und Galileo) beruht auf Signalen von Satelliten, die ständig ihre Position und eine extrem genaue Uhrzeit ausstrahlen.

GPS-Empfänger

- berechnen aus den Signallaufzeiten ihre Position, wenn sie Signale von > 3 Satelliten empfangen.
- GPS sendet bei 1.5 GHz mit wenig Leistung. Deshalb Empfang outdoor und mit Sichtverbindung.
- Genauigkeit (ohne zusätzliche terrestrische Referenzsignale (DGPS) oder zusätzliche Signale geostationärer Satelliten) um 10m (Länge, Breite). Errechnete Höhe ist unzuverlässig.
- haben keinen Rückkanal. Standortmeldungen von Trackern müssen deshalb über LoRaWAN (oder Mobilfunk o.ä.) an die Instanz gemeldet werden, die sie auswerten soll.
- geben sog. NMEA-Datensätze seriell aus. Wir nutzen Länge, Breite, Höhe und empfangene Satellitenzahl. Mehr Möglichkeiten in der Dokumentation der Libraray.
- benötigen beim Start einen sog. „Almanach“ mit den Bahndaten sämtlicher Satelliten im System
- müssen den Almanach bei jedem Start (wenn sie stromlos waren) neu vom ersten Satelliten herunterladen, den sie empfangen. Dies dauert eine Viertelstunde oder länger, also trennt eure Shields nicht vom Strom, sonst haben wir einen Warte-Workshop.
  Mobilfunkgeräte können das schneller, weil sie den Almanach aus dem Internet holen (A-GPS).

GPS seriell auslesen

Wir nutzen zwei serielle Schnittstellen. Serial ist der USB-Anschluss zum Laptop. Softwareserial (ss) auf dem Arduino Uno ist mit dem GPS verbunden. Hinweise zum Mega im Code.

Serielle Kommunikation schreibt Daten in einen Sendepuffer, aus dem sie dann ausgegeben werden, z.B. mit der Serial.print – Funktion, die Zeichen auf den seriellen Monitor schreibt.

Wenn viele Zeichen im Puffer stehen oder mehr Zeichen kommen als nur eines, das sofort weiterverarbeitet werden kann, dann benötigt das Zeit und Schleifen.

Seht euch die GPS-Auslese-Funktion smartdelay näher an:

- Die while (millis() ….) – Konstruktion sorgt dafür, dass wir eine ganze Sekunde lang auslesen (Die NMEA-Sätze dauern ungefähr eine Sekunde), damit uns keine Daten verloren gehen.

- ss.available() verrät uns, ob im Puffer überhaupt Zeichen stehen. Die while-Schleife liest so lange das zutrifft. ss.read liest immer ein Zeichen und löscht dieses dann aus dem seriellen Puffer.

- gps.encode(ss.read()) füllt ein Objekt der Library TinyGPS mit Daten und enthält (wenn alles funktioniert) anschließend valide NMEA-Sätze.

- Wer sich rohe NMEA-Sätze auf dem seriellen Monitor sehen will, z.B. um herauszufinden, ob der Empfänger glaubhafte Daten liefert, findet im code der smartdelay-Funktion eine Anleitung.

GPS-Daten auswerten / prüfen

Sehr euch das Hauptprogramm loop() näher an:

- Die GPS-Daten sind aus einem Objekt abrufbar, das TinyGPS angelegt hat. Das wird mit den gps.XXX - Befehlen erledigt (zweite if-Abfrage im Hauptprogramm loop()).

- Weitere Abfragemöglichkeiten in einigen auskommentierten Zeilen der smartdelay-Funktion, wo sie für Prüfzwecke hilfreich sein können.

- Wir wollen nur Daten senden, die valide sind. Dies bewirken die beiden if-Abfragen, ob die Satellitenzahl > 3 (Empfang aber zu wenig Sat-Signale) oder = 255 (ungültige Signale/kein Fix) ist. Mehr LoRaWAN-Sendungen würden unnütz Strom verbrauchen und unnütze Daten liefern, die wir auf Anwendungsebene wieder ausfiltern müssten.
  In den Abfragen setzen wir auch die Position zurück, damit auch dann keine gültige Position mehr in lati / previouslati steht, wenn wir den letzten GPS-Fix verloren haben.

- Wir wollen nur Daten senden, wenn der Tracker sich bewegt hat, dazu speichern wir die letzte Position in previouslati und previaouslongi am Ende des loops und gleichen in der 5. (innersten) if-Abfrage damit ab.

Daten aufbereiten und senden

Sehr euch das Hauptprogramm loop() näher an:

- Daten sparen: Bezeichner wie longitude müssen wir nicht mitsenden. Außerdem soll alles, was mitzuteilen ist, in einer einzigen Sendung senden (overhead, einheitliche Sendungen). Deshalb senden wir nur Zahlenwerte als Bytes in einem 14-byte-Feld message[14].
- Lästige Kommastellen entfernt Multiplikation mit 1 Mio.
- Sendefunktion `do_send()`:
  If-Abfrage: Wenn der LoRaWAN-Sendepuffer voll ist, …. (können wir nichts hineinschreiben)
  `LMIC_setTxData2` schreibt die Daten unseres Feldes in den LoRaWAN-Sendepuffer der LMIC, von wo aus er zu nächsten möglichen Gelegenheit gesendet wird.
- `os_runloop_once()` im Haupt-Loop ruft den job-scheduler auf, der die Sendung veranlasst.
- Wenn der Inhalt des Sendepuffers erfolgreich gesendet ist, erfahren wir das über die Funktion `void onEvent (ev_t ev)`, die uns mitteilt, wenn das Event ev == EV_TXCOMPLETE eingetreten ist. Das findet im Beispiel etwa 3,5 Sekunden nach Schreiben des Sendepuffers statt.
- Wir fragen das in der zweiten if-Abfrage von void loop() ab (voller Puffer – keine Sendung). Achtung: Sendeintervall muss groß genug sein, dass der Puffer sicher leer wird. Sonst verliert unser Tracker Pakete oder sendet gar nicht. Die nötige Dauer hängt vom eingestellten Sendeintervall und der Durchlaufzeit des Rest-Programmes ab. Bei größeren Veränderungen des Beispiels muss geprüft werden, ob das noch der Fall ist.

Timer

Das Beispiel enthält mehrere Operationen, bei denen wir auf die Zeit achten müssen, damit die Operationen funktionieren und sich nicht gegenseitig ausbremsen.

1. Das serielle Auslesen braucht Zeit, damit vollständige Datensätze gelesen werden.
2. Zwischen Schreiben des LoRaWAN-Sendepuffers und dem Abschluss der LoRaWAN-Sendung vergeht Zeit, in der keine neuen Daten geschrieben werden können.
3. Sendeintervalle müssen einen Mindestabstand haben, damit wir die Funkregulierung und die fair use policy von TTN einhalten

delay() würde den gesamten Arduino lahmlegen einschließlich der eigentlich event- und interruptgesteuerten LMIC.

Die sonst übliche Steuerung der LMIC über feste Timerintervalle wollen wir nicht, da wir dann auch Daten senden, wenn sie ungültig (kein Fix) oder uninteressant (Tracker nicht bewegt) sind.

Wir müssen die Timer also mit der Funktion millis() selbst bauen. In loop() in der ersten if…else – Konstruktion, in der Funktion smartdelay() als do…while Anweisung.

Ist die GPS-Auslesezeit zu klein, lesen wir unvollständige Datensätze. Ist das Sendeintervall zu klein gewählt, wird der Sendepuffer nicht leer und wir verlieren Daten, außerdem verstoßen wir gegen die Funkbedingungen. Verwenden wir zu oft delay(), hindern wir LMIC am Senden und der Sendepuffer wird nicht leer.

Stromsparen

Wir sparen Strom durch folgende (weiter oben erklärte) Maßnahmen

- Senden nur bei Bedarf
- Senden aller Daten in einem LoRaWAN-Paket
- Senden roher Daten als Bytes

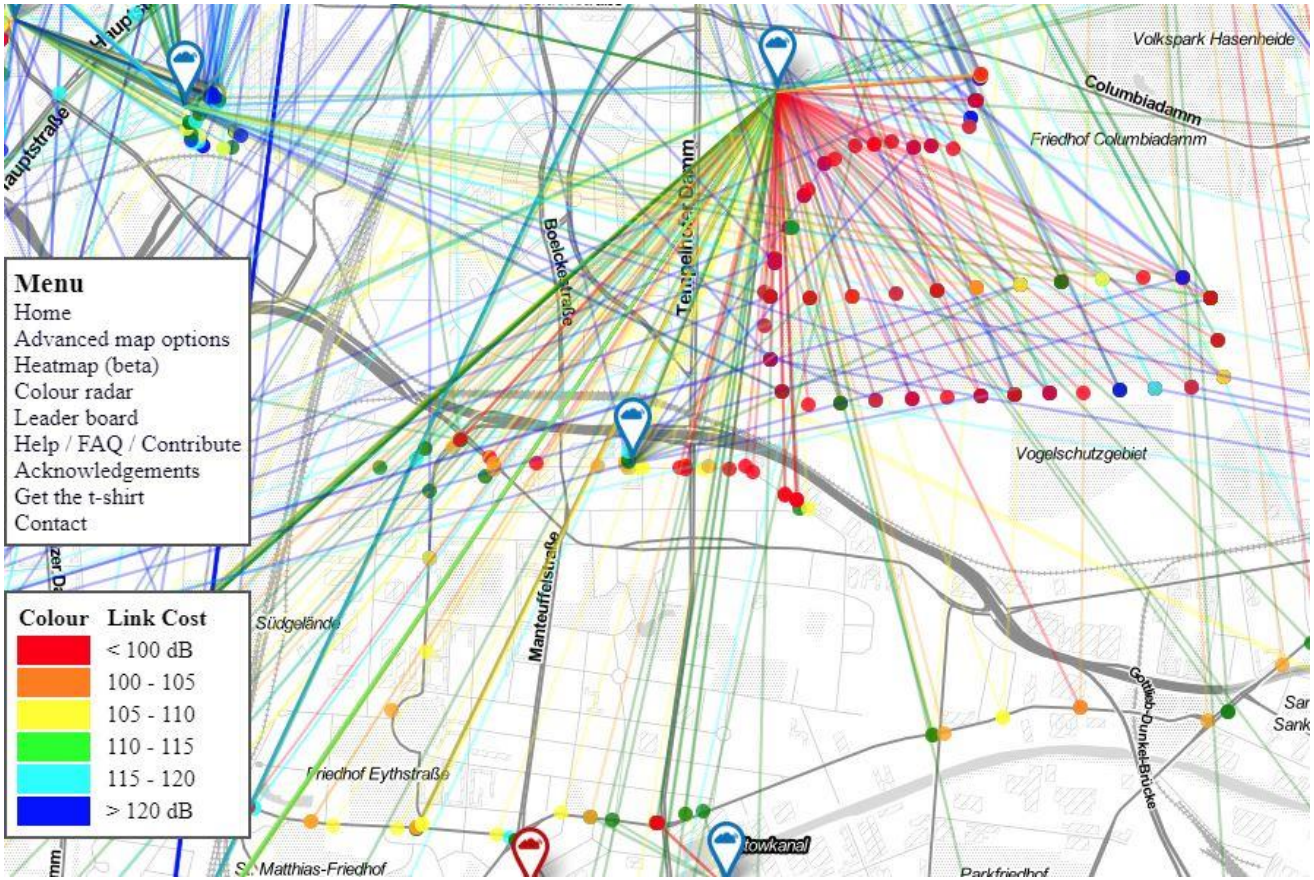
Es gibt auf dem Arduino weitere Möglichkeiten:

- Der Arduino kennt einige Schlaf-Modi und einige hardwarenähere Timer
- In einigen der Schlafmodi laufen die Timer weiter, in anderen die Möglichkeit, den Arduino interruptgesteuert wieder zu wecken.
  Ob und wie die Nutzung von Schlaf-Modi und Interrupts mit dem Auslesen des GPS und der Funktion der LMIC konfligiert, haben wir nicht getestet. Wenn Nutzer Erfahrungen damit haben, freuen wir uns über Rückmeldungen

Insgesamt ist der Arduino nicht auf Stromsparen optimiert. Es gibt Arduino-Modelle (Nano, Ideetron,…), die von Hause aus weniger Strom benötigen. Auch die Portierung auf gänzlich andere Hardware ist naheliegend.

Wir nutzen Arduino, um die Vielfalt unseres Leihgeräte-Zoos nicht noch mehr zu vergrößern.

TTN-Mapper



Wir können unsere Daten über die APIs von TTN an jedes beliebige Kartographie-Tool übergeben.
Mit TTN-Mapper sehen wir zusätzlich die LoRaWAN-Netzqualität.
Hinweis: Wenn nicht anders eingestellt, sind die Daten öffentlich!

TECHNOLOGIE STIFTUNG BERLIN

TTN-Mapper, #2

In Verbindung mit der Smartphone-App TTNmapper (Android und IOS) könnt Ihr jedes beliebige device zur Emittlung der Netzquzalität von LoRAWAN benutzen. Die Geodaten kommen dann vom Smartphone.

Den Tracker aus Workshop 2 könnt ihr aus der TTN-Konsole heraus als selbständigen Tracker mit dem TTN-Mapper verbinden: https://ttnmapper.org/faq.php

Dringende Empfehlung:
Wenn ihr nicht bewusst etwas anderes wollt, markiert eure Daten als experimentell!
Die Daten sind dadurch nicht sofort öffentlich in der Karte. Gut, wenn ihr euren Tracker nur testet und noch besser, wenn Ihr nicht wollt, dass jeder öffentlich im Internet eure Bewegungsdaten verfolgen kann.

Spielregeln

Der Bau eines Trackers ist nicht verboten. Die Nutzung kann jedoch problematisch sein:

Wenn der Tracker mit einem Gegenstand verbunden ist, der einer Person zuordenbar ist oder gar eine Person trackt, werden Trackingdaten zu personenbeziehbaren Daten und unterliegen dem Datenschutz.

Das Tracken eines Wildtieres wird also eher unproblematisch sein.

Für das Tracken von Personen oder von Gegenständen, die diesen zugeordnet werden können, benötigt Ihr die Erlaubnis dieser Personen, ebenso für die Verwendung oder Weitergabe der Daten.

Mit Trackingdaten, mit denen Ihr Euch selbst getrackt habt, dürft ihr natürlich machen, was ihr wollt.

TECHNOLOGIE STIFTUNG BERLIN