Pervasive Computing

LAB:

LoRa, LoRaWan, TheThingsNetwork,

Sensors & LoPy

Sebastian Büttrich
March 2107

Agenda

- Understand LoRa, LoRaWan, TheThingsNetwork

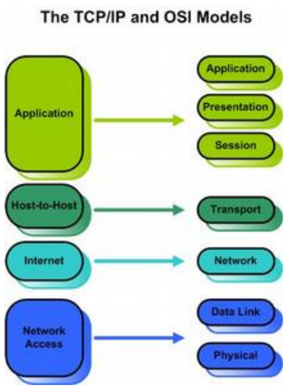
  (this presentation)

  (demo, groupwork)
- Get started with the LoPy and pymakr
- Bring the LoPy onto the TTN network
- Connect sensors to the LoPy
- Send sensor data to TTN
- [TTN Data integration]

LoRa / 1

LoRa is a **proprietary Layer 1 standard** owned by Semtech

Chirp Spread Spectrum (CSS) with forward error coding (in combination with whitening and interleaving).



Bandwidth 125/250/500 kHz

Frequency in Europe: **ISM 868 MHz**

Data rate 11 kbps

Focus is on **long range, power efficiency, robustness.**

Modulation details:

http://www.semtech.com/images/datasheet/an1200.22.pdf

https://www.lora-alliance.org/portals/0/documents/whitepapers/LoRaWAN101.pdf

https://revspace.nl/DecodingLora#Modulation_basics

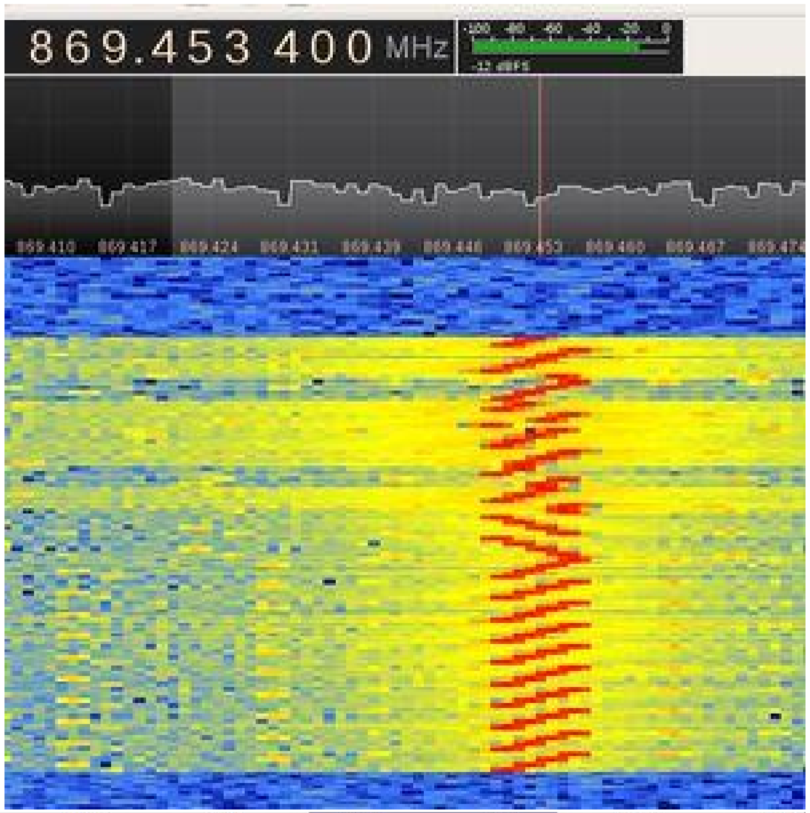
https://myriadrf.org/blog/lora-modem-limesdr/

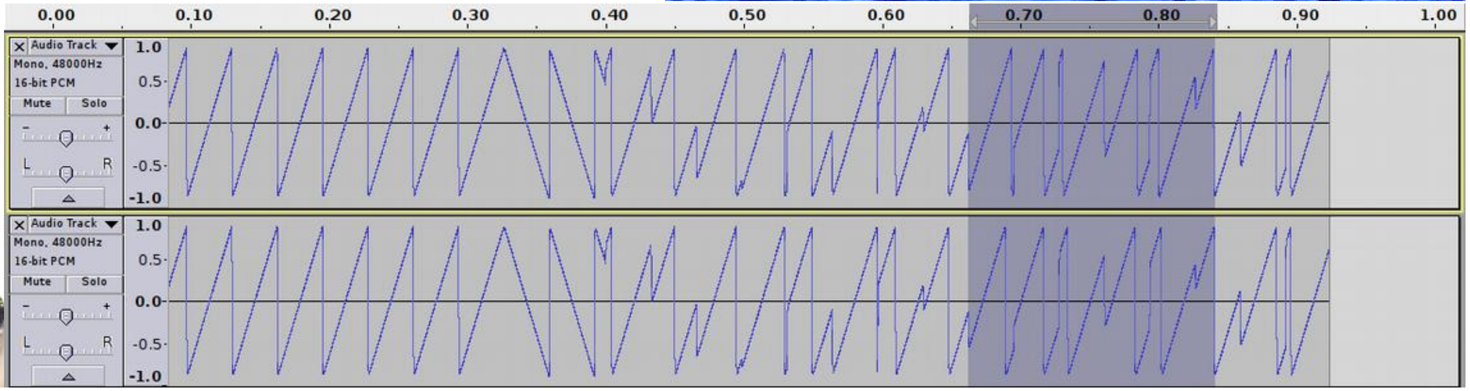
https://static1.squarespace.com/static/54cecce7e4b054df1848b5f9/t/57489e6e07eaa0105215dc6c/1464376943218/Reversing-Lora-Knight.pdf

LoRa / 2

What is a chirp?

Source: https://revspace.nl/DecodingLora





LoRa / 3

Most important choices:
Bandwidth,

Spread Factor SF, determines how wide we spread out the actual bits to be carried

Control rate CR, determines depth of forward error coding

"The higher SF and CR, the further you can go, at the expense of speed/data rate"

LoRa / 4

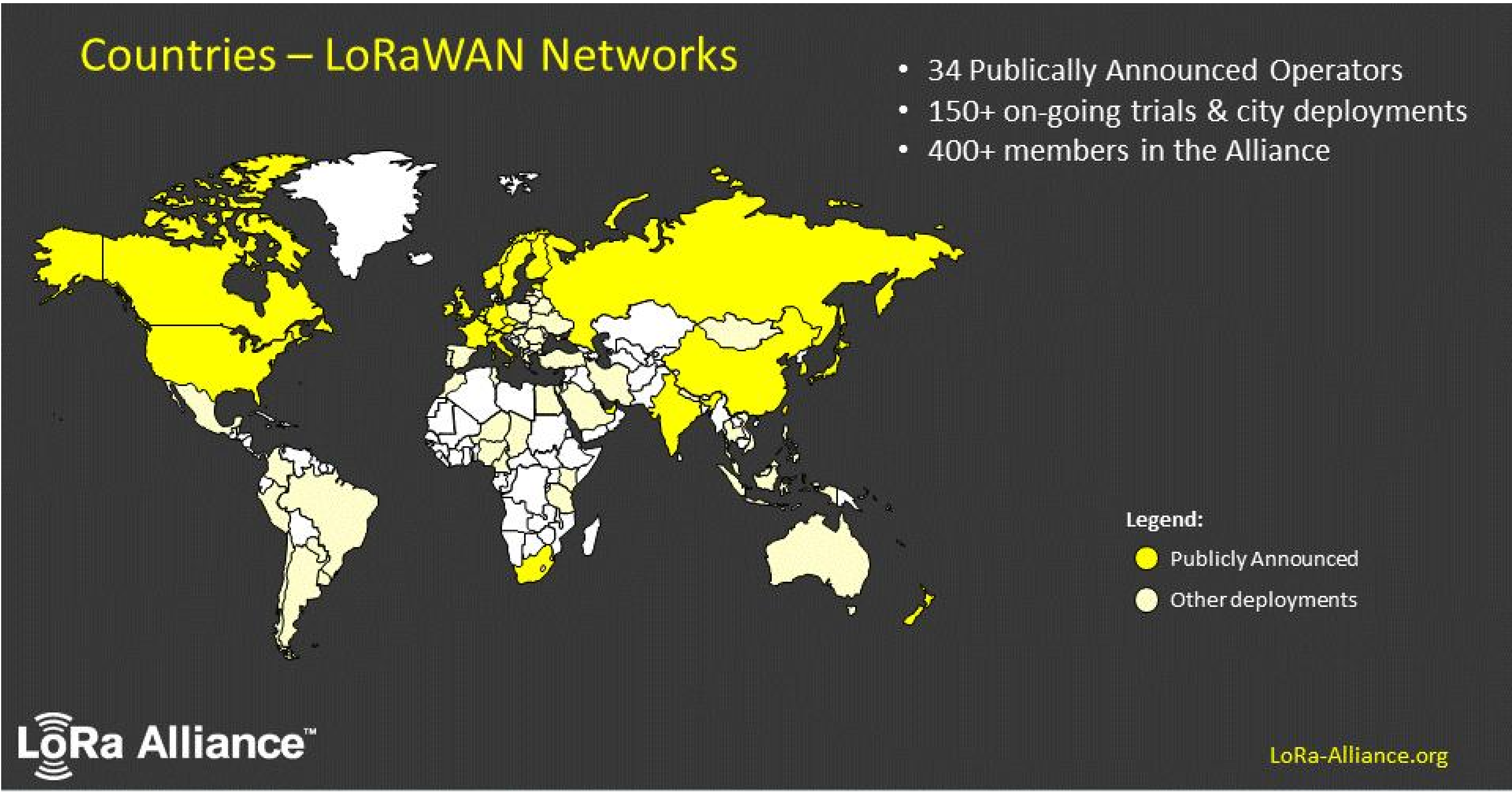
Business aspects:
LoRa / LoRaWan - one of many contenders for the IOT throne.
Others include NB-IoT, Sigfox, ...

SIGFOX
One network A billion dreams
nwave
LoRa™
LTE-M
NB-LTE
INGENU
WEIGHTLESS
IEEE 802.11ah
DASH7™ ALLIANCE
EC-GSM
ZigBee 3.0
Bluetooth® 4.0

LoRaWan / 1

- LoRaWan is an open standard building on top of LoRa
- https://www.lora-alliance.org/

LoRaWan / 2


Countries – LoRaWAN Networks
• 34 Publically Announced Operators
• 150+ on-going trials & city deployments
• 400+ members in the Alliance
Legend:
● Publicly Announced
● Other deployments
LoRa Alliance™
LoRa-Alliance.org

LoRaWan / 3

LoRaWAN™ is a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated Things in a regional, national or global network. LoRaWAN targets key requirements of Internet of Things such as secure bi-directional communication, mobility and localization services. The LoRaWAN specification provides seamless interoperability among smart Things without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the roll out of Internet of Things.

Details:

https://www.lora-alliance.org/What-Is-LoRa/Technology

LoRaWan / 4

- **Star-of-stars topology**
- **Gateways** are transparent bridges relaying messages between **end-devices** and a central **network server** in the backend.
- **Gateways are connected** to the network server via **standard IP connections** while end-devices use single-hop wireless communication to one or many gateways.
- All end-point communication generally **bi-directional**, supports **multicast** enabling **software upgrade over the air** or other mass distribution messages

Details:

https://www.lora-alliance.org/What-Is-LoRa/Technology

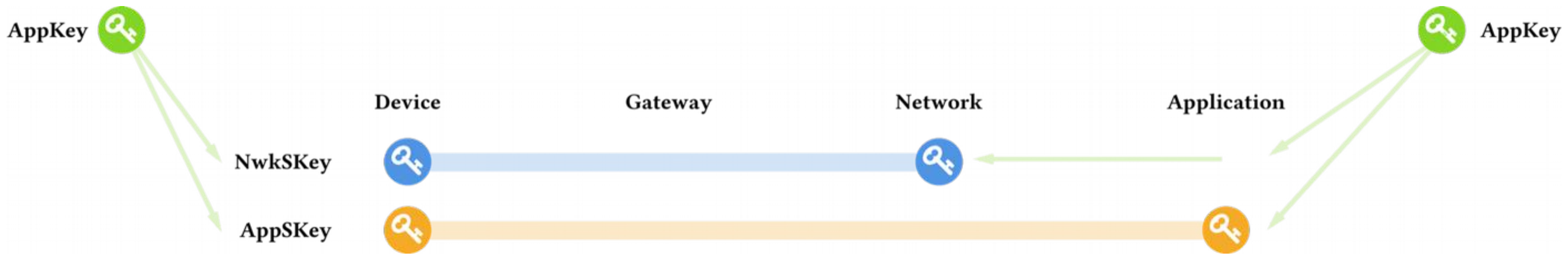
LoRaWan / 5

Security measures:

Unique Network key (EUI64), network level

Unique Application key (EUI64), end to end security on application level

Device specific key (EUI128),
Device 32 bit id, unique in network, ideally globally unique



Source, Details:

https://www.lora-alliance.org/What-Is-LoRa/Technology

LoRaWan / 6

Device classes

A	Battery powered, small loads, long breaks, long latency, unicast
B	low latency, scheduled receive slots, periodic beacon from gateway, uni/multicast, higher power, 14-30 mA
C	no latency, uni/multi, constantly receiving, power hungry

Classes can be dynamically assigned / changed

Source, Details:

https://www.lora-alliance.org/What-Is-LoRa/Technology

The Things Network / 1

THE THINGS
NETWORK

The Things Network / 2 / Manifesto

Everything that carries power will be connected to Internet eventually.

Controlling the network that makes this possible means controlling the world. We believe that this power should not be restricted to a few people, companies or nations. Instead this should be distributed over as many people as possible without the possibility to be taken away by anyone. We therefore founded "The Things Network".

The Things Network is an open source, free initiative with the following properties:

It connects sensors and actuators, called "Things", with transceivers called "Things Gateways" to servers called "Things Access".
The first connection is "Over The Air", the second is "Over The Net". The distributed implementation of these concepts is called "The Things Network".
Anyone shall be free to set up "Things" and connect to "Things Gateways" that may or may not be their own.
Anyone shall be free to set up "Things Gateways" and connect to "Things Access" that may or may not be their own. Their "Things Gateways" will give access to all "Things" in a net neutral manner, limited by the maximum available capacity alone.
Anyone shall be free to set up "Things Access" and allow anonymous connections from the Internet. Their "Things Access" will give access to all "Things Gateways" in a net neutral manner, limited by the maximum available capacity alone. Furthermore their "Things Access" will allow connection of other "Things Access" servers for the distribution of data.
The "Over The Air" and "Over The Net" networks shall be protocol agnostic, as long as these protocols are not proprietary, open source and free of rights.
Anyone who perpetrates a "Things Access" or a "Things Gateway" will do so free of charge for all connecting devices and servers.
Anyone making use of the network is allowed to do so for any reason or cause, possibly limited by local law, fully at own risk and realizing that services are provided "as is" and may be terminated for any reason at any moment. The use may be open for anybody, limited to customers, commercial, not-for-profit, or in any other fashion. "The Things Network" providers will not pose restrictions upon its users.

We invite you to sign this Manifesto, and uphold its principles to the best of your abilities.

Source, Details:

https://github.com/TheThingsNetwork/Manifest

The Things Network / 3 / Essentials

- Open source
- Free ... to set up and run their own, in particular:

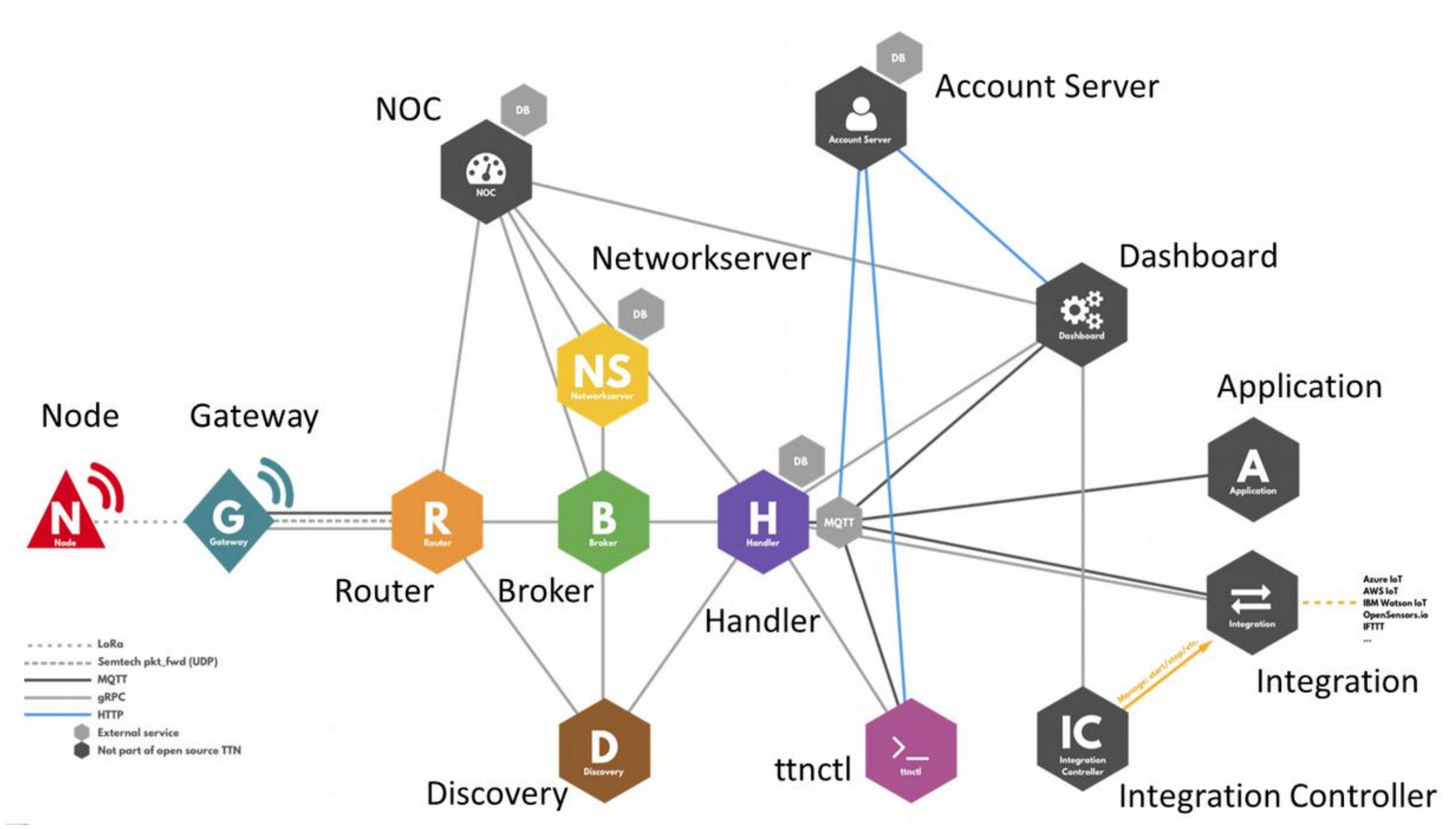
  Anyone who perpetrates a "Things Access" or a "Things Gateway" will do so **free of charge for all connecting devices and servers.**

  This to some degree explains our current interest in TTN, in an educational context.

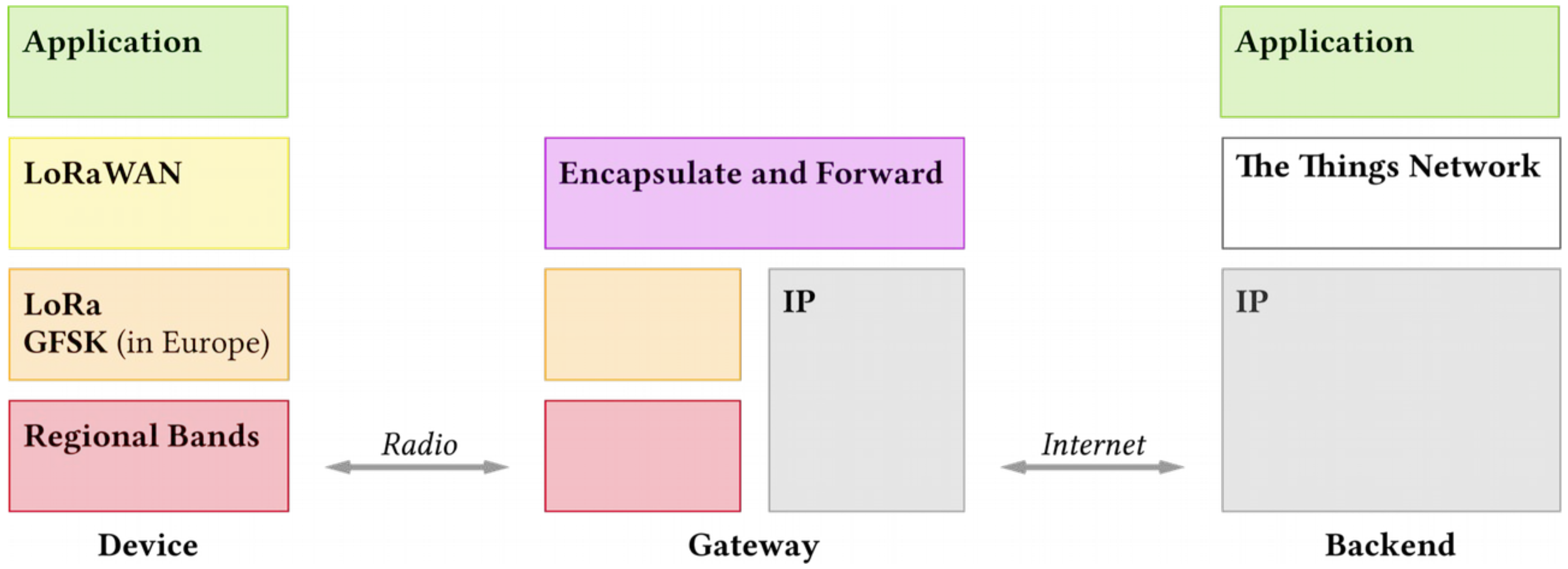
Source, Details:

https://github.com/TheThingsNetwork/Manifest

The Things Network / 4 / Architecture



The Things Network / 5 / Architecture



The Things Network / 6 / Security

= LoRaWan defined

NwkSKey, AppSKey and AppKey. All keys have a length of 128 bits.

NwkSKey for interaction between Node and the Network. Also used to map a non-unique device address (DevAddr) to a unique DevEUI and AppEUI.

AppSKey is used for encryption and decryption of the payload.

NwkSKey and AppSKey are unique per device, per session. If you dynamically activate your device (OTAA), these keys are re-generated on every activation.

Source, Details:
https://github.com/TheThingsNetwork/Manifest

The Things Network / 7 / Security, cntd

Dynamically activated devices (OTAA) use the application key (AppKey) to derive the two session keys during the activation procedure. In The Things Network you can have a default AppKey which will be used to activate all devices, or customize the AppKey per device.

What you will use, in your code:

DevEUI, AppEUI, AppKey

Keys will be generated on TTN server, on registration.

Source, Details:

https://www.thethingsnetwork.org/wiki/LoRaWAN/Security

HANDS-ON Lab / 1

From here on, we go to hands-on / demo mode.

You might prefer to use the plain text version,

As you will copypaste some code.

HANDS-ON Lab / 2 / pymakr

```
======================================================

TEXT GUIDE for LoRa, LoRaWan, LoPy, sensors LAB

======================================================


Work with your LoPy / pymakr.

============================================


We demo this.


Understand:


IDE

REPL

CTRL-A,B,C,D

uploading

syncing
```

HANDS-ON Lab / 3 / REPL

What is the REPL?

MicroPython Interactive Interpreter Mode. A commonly used term for this is REPL (read-eval-print-loop) which will be used to refer to this interactive prompt.

It looks like this:

```
>>>
```

Learn how it works: https://docs.pycom.io/pycom_esp32/reference/repl.html

HANDS-ON Lab / 4 / CTRL-x

CTRL-A

raw REPL - we wont need this.

CTRL-B:

brings you back to REPL.

CTRL-C:

You can interrupt a running program by pressing Ctrl-C. This will raise a KeyboardInterrupt which will bring you back to the REPL, providing your program doesn’t intercept the KeyboardInterrupt exceptio

HANDS-ON Lab / 5 / CTRL-D

CTRL-D:

performs a soft reset. You will see a message something like:

```
>>>
PYB: soft reboot
MicroPython v1.4.6-146-g1d8b5e5 on 2016-10-21; LoPy with ESP32
Type "help()" for more information.
>>>
```

Another way to do a soft reset:

```
>>> import sys
>>> sys.exit()
```

or, as python code:

```
raise SystemExit
```

HANDS-ON Lab / 6 / Things go wrong

When things go wrong - **and they do!** - ... you can:

===

- Reset.
- Restart pymakr.
- Reconnect device.
- Do a hard reset:

```
>>> import machine
>>> machine.reset()
```

- Factory reset the file system:

```
>>> import os
>>> os.mkfs('/flash')
```

- Connect via WiFi and ftp – try to avoid it, though!
- Safe Boot.
- Update Firmware.

  Learn more:

  https://docs.pycom.io/pycom_esp32/pycom_esp32/toolsandfeatures.html#safeboot

HANDS-ON Lab / 7 / traffic lights

Now that we control the LoPy (somewhat),

let s try a project.

Make a new project,

make a new empty folder for it.

take some sample code, e.g. the traffic light,

and paste it into a new window:

```
import pycom
import time
pycom.heartbeat(False)
for cycles in range(10): # stop after 10 cycles
    pycom.rgbled(0x007f00) # green
    time.sleep(5)
    pycom.rgbled(0x7f7f00) # yellow
    time.sleep(1.5)
    pycom.rgbled(0x7f0000) # red
    time.sleep(4)
```

HANDS-ON Lab / 8 / sync

Save as main.py.

To be safe, also make a boot.py:

```
-----------------------------------------------------
from machine import UART
import os
uart = UART(0, 115200)
os.dupterm(uart)
-----------------------------------------------------
```

Now sync that project into the device:

Now, here i often hang ... i do

```
>>> import os
>>> os.mkfs('/flash')
>>> os.listdir()
[]
>>> os.getcwd()
'/flash'
```

More on this:

https://micropython.org/resources/docs/en/latest/wipy/library/os.html

HANDS-ON Lab / 9 / sync success

The sync:

```
Syncing the project with the Pycom device. Please wait...
Successfully synced!
```

Note!!! You cannot sync if your i2c bus is busy, e.g. busy with a sensor.

Soft reboot, and your code should run -

e.g. the traffic light.

HANDS-ON Lab / 10 / LoRaWAN

Now let s talk LoRa.

First, let s get our DevEUI:

```
-----------------------------------------------------------------
#get your device's EUI
from network import LoRa
import binascii
lora = LoRa(mode=LoRa.LORAWAN)
print(binascii.hexlify(lora.mac()).upper().decode('utf-8'))
-----------------------------------------------------------------
```

HANDS-ON Lab / 11 / TTN

Now that we know our DevEUI,

we need to go to

TTN and register our Devices under a certain App that we agreed on.

Se the Forum thread -
groups fill in their Device ID with group number.

Everybody is welcome to register on TTN themselves, if they like to.

When this is done, we know our AppEUI and AppKey,
which we share on the forum again..

HANDS-ON Lab / 12 / Connect device

We are ready to run this code, to register our device OTAA:

```
import binascii
import pycom
import socket
import time
from network import LoRa

# Colors
off = 0x000000
red = 0xff0000
green = 0x00ff00
blue = 0x0000ff

# Turn off heartbeat LED
pycom.heartbeat(False)

# Initialize LoRaWAN radio
lora = LoRa(mode=LoRa.LORAWAN)

# Set network keys
app_eui = binascii.unhexlify('YOUR_APP_EUI')
app_key = binascii.unhexlify('YOUR_APP_KEY')

# Join the network
lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key), timeout=0)
pycom.rgbled(red)

# Loop until joined
while not lora.has_joined():
    print('Not joined yet...')
    pycom.rgbled(off)
    time.sleep(0.1)
    pycom.rgbled(red)
    time.sleep(2)

print('Joined')
pycom.rgbled(blue)

s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
s.setblocking(True)

i = 0
while True:
    count = s.send(bytes([i % 256]))
    print('Sent %s bytes' % count)
    pycom.rgbled(green)
    time.sleep(0.1)
    pycom.rgbled(blue)
    time.sleep(9.9)
    i += 1
```

Check if all devices show up ok, and are sending data.

HANDS-ON Lab / 13 / sensors

A general sensor can be read like this:

```
----------------------------------------------------------------------
#reading an analog sensor with lopy
#https://docs.pycom.io/pycom_esp32/library/machine.ADC.html#class-
adcchannel-read-analog-values-from-internal-or-external-sources
import machine
adc = machine.ADC()              # create an ADC object
apin = adc.channel(pin='P17')    # create an analog pin on P17 - look out
for pin confusion
for cycles in range(1000): # stop after 1000 cycles
    val = apin()                    # read an analog value
    apin.value()          # same
    print(val)
    time.sleep(1)
----------------------------------------------------------------------
```

HANDS-ON Lab / 14 / sensor data > TTN

Now combine the two elements -

1/ LoRaWan network

&

2/ Sensors

to send sensor data to TTN.

HANDS-ON Lab / 15 / humidity sensor

```
----------------------------------------------------------------------
import time
from machine import I2C
from struct import unpack
class Chirp:
        def __init__(self, address):
                self.i2c = I2C(0, I2C.MASTER, baudrate=10000)
                self.address = address

        def get_reg(self, reg):
                val = unpack('<H', (self.i2c.readfrom_mem(self.address, reg, 2)))[0]
                return (val >> 8) + ((val & 0xFF) << 8)

        def moist(self):
                return self.get_reg(0)

        def temp(self):
                return self.get_reg(5)

        def light(self):
                self.i2c.writeto(self.address, '\x03')
                time.sleep(1.5)
                return self.get_reg(4)

if __name__ == '__main__':
        addr = 0x20 #or 32
        chirp = Chirp(addr)
        for cycles in range(10):
                print('moist: ' +str(chirp.moist()))
                print('temp: ' +str(chirp.temp()))
                print('light: ' +str(chirp.light()))
                time.sleep(1)
----------------------------------------------------------------------
```