Popular Electronics

PRESENTS

Create>

Deep Dive Applications Series

Arduino for the Cloud

Dr. Claus Kühnel

Arduino for the Cloud

Arduino Yùn and Arduino Yùn Shield

By Dr. Claus Kühnel

TABLE OF CONTENTS

1 ○ Preface

In any survey of the relevant forums and portals addressing microcontroller technology, you cannot miss the topic of Arduino—the open-source prototyping platform based on flexible and easy-to-use hardware and software.

The primary motivation for adopting Arduino is direct interaction with an operating environment, which it can control via sensors and various actors. The field of "physical computing" addresses the myriad applications for which access to, and control of, the environment is important.

Thanks to its simple and accessible user experience, Arduino has been deployed in countless and varied projects and applications. The Arduino software is easy to use for beginners yet flexible enough for advanced users. Consequently, makers and professionals alike use Arduino for purposes spanning the evaluation of technology to prototyping to commercial and industrial products.

The increasing complexity of today's requirements for electronic systems have long exceeded the capabilities of classic microcontrollers alone. In most cases now the networking of components is imperative, the absence of which would render the Internet of Things (IoT) wishful thinking.

This eBook will demonstrate how the Arduino Yùn can be used to close the gap between controlling the environment and the networking of information.

2 ○ Arduino Yùn Hardware

The Arduino Yùn is not the only device of the Arduino family that is suitable for networking. Let us have a look into the Arduino Products page (https://www.arduino.cc/en/Main/Products) for IoT devices. Table 1 on the next page shows the most important Arduinos from the IoT area of the above-mentioned website.

Arduino Yùn, Arduino Industrial 101, and Arduino Yùn Mini use an Atmel ATmega32u4 as microcontroller, as does the Arduino Leonardo. On the network side, we find a Linux device based on Qualcomm's Atheros AR9331 running Linino, an OpenWRT derivative. Figure 1 shows an Arduino Yùn. You can see clearly the building block around the Atheros 9331 and the USB and network connectors.

The Arduino Tian offers more performance in both areas. The microcontroller in this case is the Atmel SAMD21, a 32-Bit Cortex-M0+, and the Atheros AR9331 is replaced by an AR9432. All these Arduinos offer Ethernet TCP/IP and Wi-Fi communication. A USB Host interface allows expansions via USB for the Linux device, enabling, for example, memory enhancement with USB stick or USB hard disk, the connection of a USB webcam, or other USB devices.

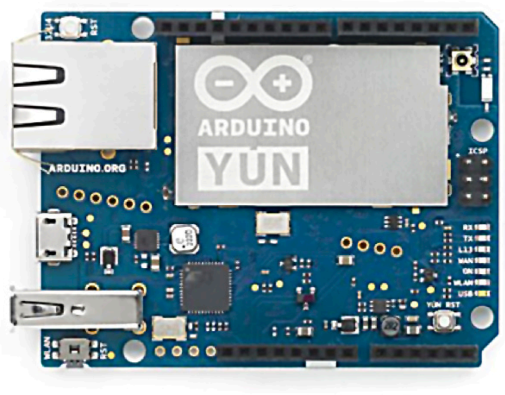


FIGURE 1 | ARDUINO YÙN

The next entries in Table 1 show Arduino Ethernet and Arduino Leonardo ETH. Both boards get their network capability by a Wiznet W5x00 Ethernet Controller connected via serial interface to the microcontroller. Both boards offer Ethernet TCP/IP only. The Arduino MKR1000 uses an Atmel SAMD21 as microcontroller and an Atmel ATWINC1500 IoT Module offering Wi-Fi Direct

Table 1 | Arduinos for Networking

Arduino Type	Microcontroller	Network	ETH	WiFi	USB	Price
Arduino Yùn	ATmega32u4	Atheros AR9331	802.3 10/100 Mbit/s	802.11b/g/n 2.4 GHz	Type-A 2.0 Host	$ 71.99
Arduino Industrial 101	ATmega32u4	Atheros AR9331i	802.3 10/100 Mbit/s (on headers)	802.11 b/g/n 2.4 GHz	2.0 Host (on headers)	$ 38.83
Arduino Yùn Mini	ATmega32u4	Atheros AR9331	802.3 10/100 Mbit/s	802.11 b/g/n 2.4 GHz	2.0 Host	$ 68.33
Arduino Tian	Atmel SAMD21	Atheros AR9342	802.3 10/100/1000 Mbit/s	802.11 b/g/n 2.4 GHz du-al-band	Type-A 2.0 Host	$ 96.51
Arduino Ethernet	ATmega328	W5100 TCP/IP Embedded Ethernet Controller	10/100Mbit/s	n.a.	n.a.	$ 44.26
Arduino Leonardo ETH	ATmega32u4	W5500 TCP/IP Embedded Ethernet Controller	10/100Mbit/s	n.a.	n.a.	$ 48.69
Arduino MKR1000	Atmel SAMD21	ATWINC1500	n.a.	802.11 b/g/n 2.4 GHz	embedded host and device	$ 35.49
Arduino Yùn Shield	n.a.	Atheros AR9331	802.3 10/100Mbit/s	802.11b/g/n	Type-A 2.0 Host	$ 48.70
Arduino Ethernet Shield V2	n.a.	W5500 TCP/IP Embedded Ethernet Controller	10/100Mbit/s	n.a.	n.a.	$ 24.40
Arduino Uno	ATmega328	n.a.	n.a.	n.a.	Virtual COM port	$ 22.17
Arduino Leonardo	ATmega32u4	n.a.	n.a.	n.a.	Virtual COM port	$ 19.97

and Soft-AP support. There is no wired Ethernet TCP/IP.

With the Arduino Yùn shield (Figure 2) or the Arduino Ethernet shield you can build your own device based on either the Arduino Uno or Arduino Leonardo.

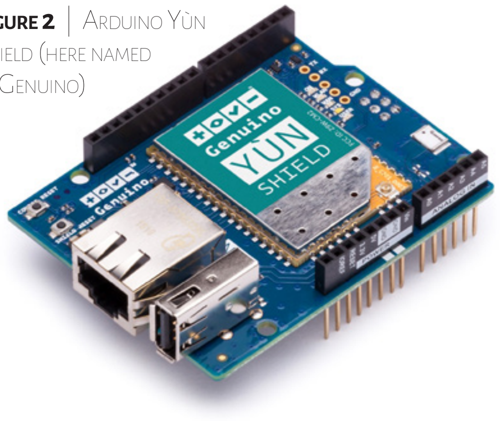
Now we come to the decision of which Arduino device to use for our project.

Arduino Yùn is an expensive part, but it offers everything we need for physical computing and networking. If you want to go cheaper, then you have to reduce the functional and/or performance requirements.

If you happen to have an Arduino Uno or Leonardo available, then you can reduce cost by using an Arduino Yùn shield, yielding equivalent functionality.

Furthermore, Arduino is open-source hardware, and therefore all of the original design files (Eagle CAD) for the Arduino hardware are available under a Creative Commons Attribution Share-Alike license (CC BY-SA) [1].

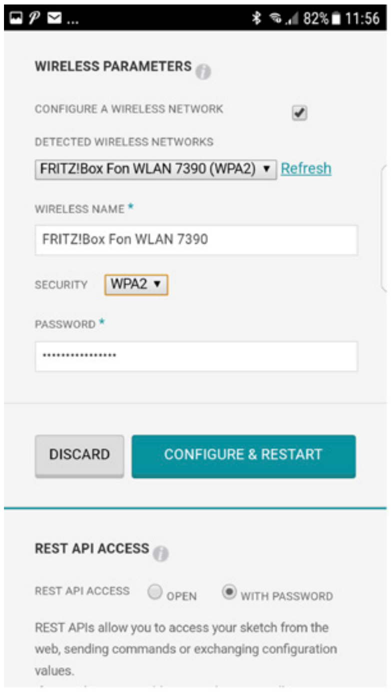
Figure 2 | Arduino Yùn Shield (here named as Genuino)



This license allows for both personal and commercial derivative works, as long as they credit Arduino and release their designs under the same license.

In my book covering Arduino Yùn [2] I used a Dragino Yùn shield [3] along with the Arduino Yùn. This shield is compatible with the Arduino Yùn shield and was earlier available. Dragino fulfills the license conditions, with design files published on Github.

Beware, though, that counterfeits abound. While such noncompliant clones cost less than authentic Arduino devices, honoring the license terms helps the Arduino Foundation to maintain an excellent community, which also publishes up-to-date tutorials and documentation, as well as managing the required certifications (e.g., FCC).


82% 11:56

WIRELESS PARAMETERS

CONFIGURE A WIRELESS NETWORK ✓

DETECTED WIRELESS NETWORKS: FRITZ!Box Fon WLAN 7390 (WPA2) ▾ Refresh

WIRELESS NAME *: FRITZ!Box Fon WLAN 7390

SECURITY: WPA2 ▾

PASSWORD *: ••••••••••••••••

DISCARD | CONFIGURE & RESTART

REST API ACCESS

REST API ACCESS: OPEN [✓ WITH PASSWORD]

REST APIs allow you to access your sketch from the web, sending commands or exchanging configuration values.

Figure 3 | Network Configuration

The Arduino software is also open source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL [4][5].

Finally, a word about the names Arduino and Genuino. This is an artifact of branding issues that were resolved with organizational and regional management changes at the Foundation. Today there is a single Arduino organization [6].

3 ○ Update the Firmware

Software is always "under construction." This means you should always use the latest revision of software when starting a new project.

The actual Arduino Yún Linux OS is OpenWRT-Yún 1.5.3. In most cases the installed OS version may differ. Always download the latest stable Linux OS to make your Arduino Yún more stable and feature rich.

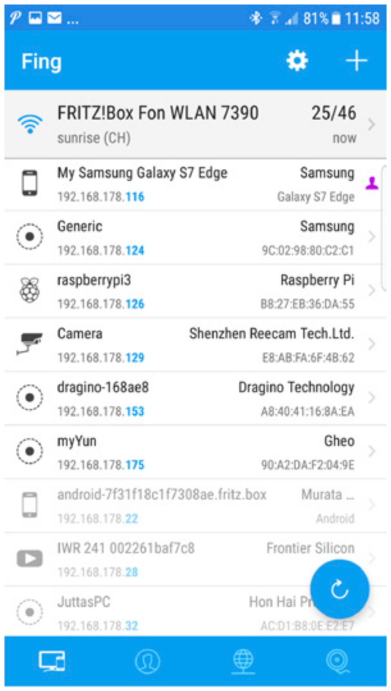
For upgrading the Linux OS there is a step-by-step instruction at the Arduino website https://www.arduino.cc/en/Tutorial/YunSysupgrade. I used the easy way via the Web Panel to get the latest stable Linux OS on my Arduino Yún.

4 ○ Network Configuration

Thanks to the network interface, you can integrate the Arduino Yún into your home network as well. I consider here primarily the wireless interface, because the Arduino Yún can be deployed without long cable connections near sensors, for example.

For the configuration of the network I used the Web Panel again and connected the Arduino Yún with my router as shown in Figure 3. After saving the configuration I found a new member in the network. Here I use the Android app Fing (Figure 4) to detect devices in my network and find *myYun* with the IP address 192.168.178.175.

Now I can access the Arduino Yún via SSH as shown in Figure 5.

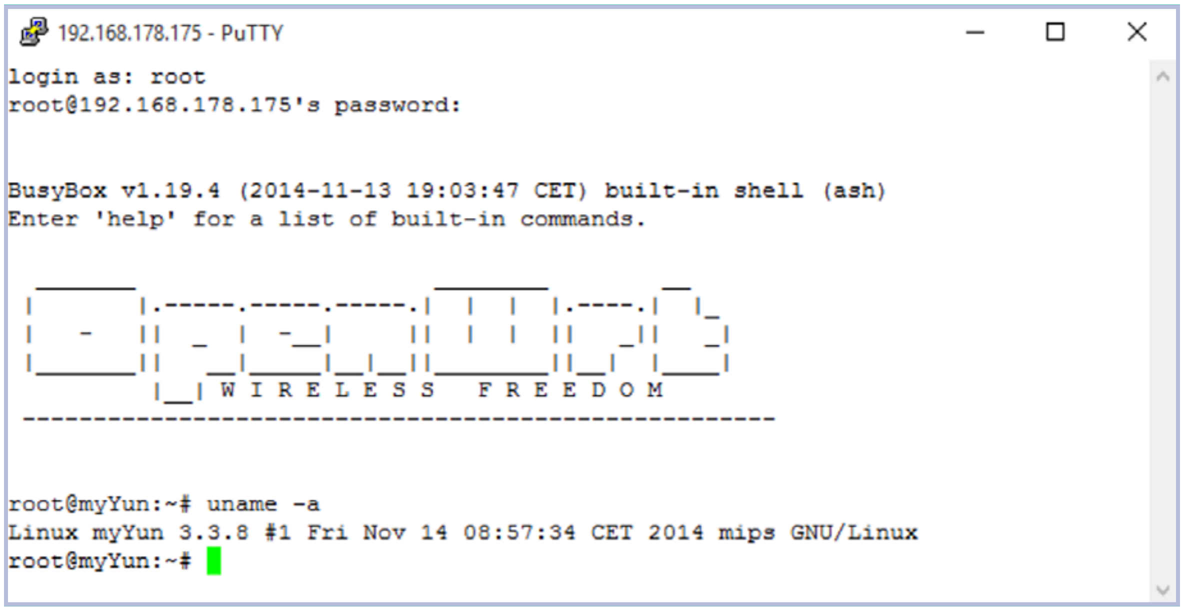

81% 11:58

Fing

FRITZ!Box Fon WLAN 7390 — 25/46
sunrise (CH) — now

My Samsung Galaxy S7 Edge 192.168.178.116	Samsung Galaxy S7 Edge
Generic 192.168.178.124	Samsung 9C:02:98:80:C2:C1
raspberrypi3 192.168.178.126	Raspberry Pi B8:27:EB:36:DA:55
Camera 192.168.178.129	Shenzhen Reecam Tech.Ltd. E8:AB:FA:6F:4B:62
dragino-168ae8 192.168.178.153	Dragino Technology A8:40:41:16:8A:EA
myYun 192.168.178.175	Gheo 90:A2:DA:F2:04:9E
android-7f31f18c1f7308ae.fritz.box 192.168.178.22	Murata ... Android
IWR 241 002261baf7c8 192.168.178.28	Frontier Silicon
JuttasPC 192.168.178.32	Hon Hai Pr AC:D1:B8:0E:E2:E7

Figure 4 | myYun in the network



```
login as: root
root@192.168.178.175's password:


BusyBox v1.19.4 (2014-11-13 19:03:47 CET) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------


root@myYun:~# uname -a
Linux myYun 3.3.8 #1 Fri Nov 14 08:57:34 CET 2014 mips GNU/Linux
root@myYun:~#
```

Figure 5 | SSH access to Arduino Yún

We can see that BusyBox v1.19.4 and a Linux kernel 3.3.8 are installed.

Figure 6 shows the integration of the Arduino Yún into my home network. The Arduino Yún is connected with the development PC via the on-board micro-USB connector. Via this connector, the Arduino obtains its power and the software upload can occur. As you will see later I use the WLAN connection for software upload. Therefore, for powering the Arduino Yún, a mobile phone charger equipped with a micro-USB connector is sufficient. In Figure 6, this part is not shown.

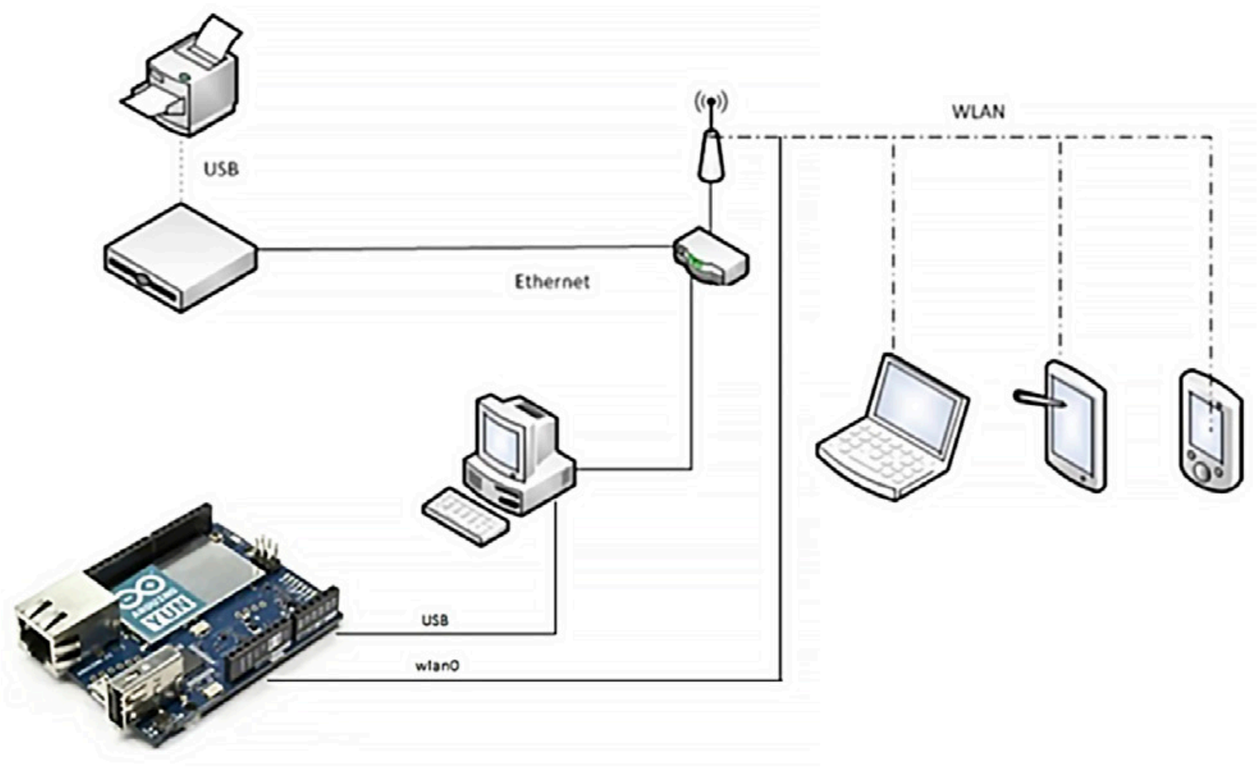


Figure 6 | Arduino Yún in my network

5 ○ Network Performance

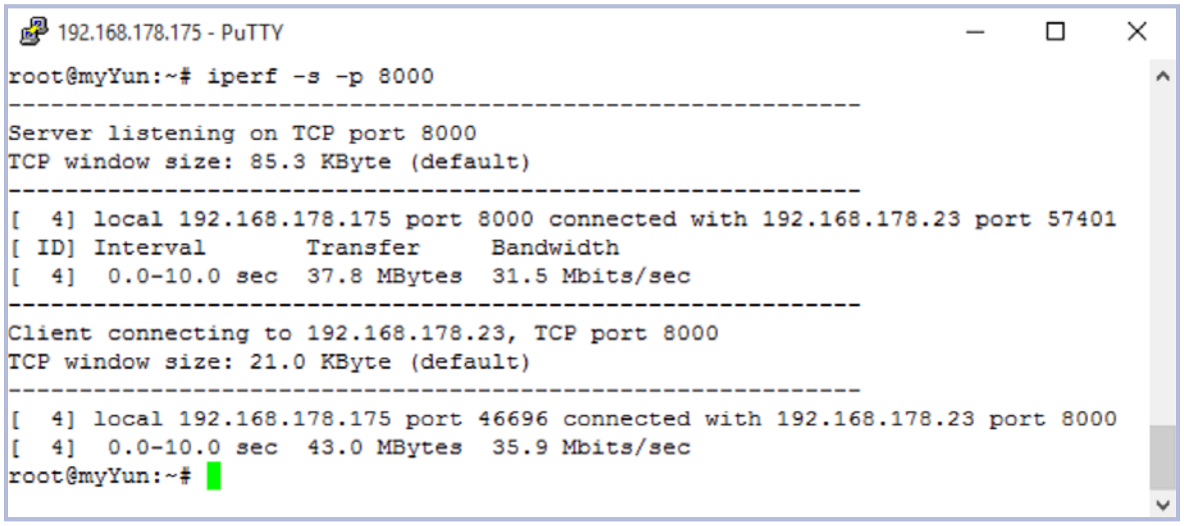
When we prepare an IoT project with an Arduino Yún it is important to know the network performance. The networking performance was measured for an Arduino with Ethernet shield [7].

Using the speed test tool *iperf* we can do the same. The Arduino Yún serves as *iperf* server and a Windows PC as *iperf* client.

First we have to install *iperf* on our Arduino Yún by the command:

```
$ opkg install iperf
```

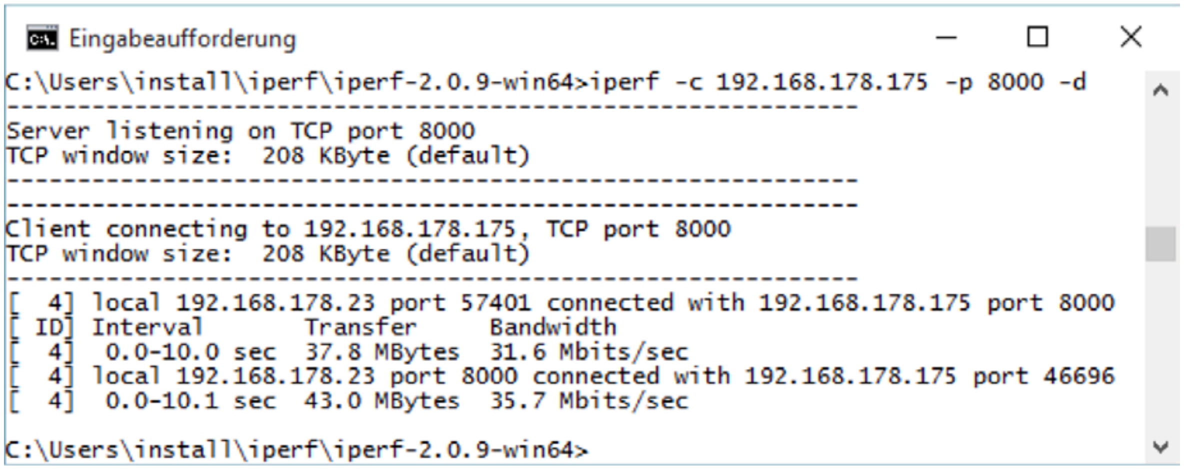
and use the Arduino Yún as the server for this test. Figure 7 shows how to start the server side for the bandwidth test.



```
root@myYun:~# iperf -s -p 8000
------------------------------------------------------------
Server listening on TCP port 8000
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.178.175 port 8000 connected with 192.168.178.23 port 57401
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  37.8 MBytes  31.5 Mbits/sec
------------------------------------------------------------
Client connecting to 192.168.178.23, TCP port 8000
TCP window size: 21.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.178.175 port 46696 connected with 192.168.178.23 port 8000
[  4]  0.0-10.0 sec  43.0 MBytes  35.9 Mbits/sec
root@myYun:~#
```

Figure 7 | *iperf* Server on Arduino Yún

My Windows development PC serves as *iperf* client. It calls the IP address of the Arduino Yún and the same port. The option *–d* makes the test in both directions. Figure 8 shows the command to start the *iperf* client from Windows command line.



```
C:\Users\install\iperf\iperf-2.0.9-win64>iperf -c 192.168.178.175 -p 8000 -d
------------------------------------------------------------
Server listening on TCP port 8000
TCP window size:  208 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 192.168.178.175, TCP port 8000
TCP window size:  208 KByte (default)
------------------------------------------------------------
[  4] local 192.168.178.23 port 57401 connected with 192.168.178.175 port 8000
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  37.8 MBytes  31.6 Mbits/sec
[  4] local 192.168.178.23 port 8000 connected with 192.168.178.175 port 46696
[  4]  0.0-10.1 sec  43.0 MBytes  35.7 Mbits/sec

C:\Users\install\iperf\iperf-2.0.9-win64>
```

Figure 8 | *iperf* Client on Windows PC

If your PC has no *iperf* installed you can download it from https://iperf.fr/iperf-download.php. Please pay attention to the version of *iperf*. If on Arduino Yún iperf-2.0 is installed, then you have to install the same version on the client side.

From Figure 7 and/or Figure 8 we get a bandwidth of over 30 MBits/sec for this configuration. For Arduino with Ethernet shield this result was about 3 MBits/sec only in [7]. This is a very important point for the cost-performance discussion in Chapter 2.

6 ○ Connecting Two Worlds

As described in Chapter 2, the Arduino Yún includes two processors. The *Bridge* library ensures that both processors can communicate and hides the implementation details from the user. Figure 9 shows how the *Bridge* connects the microcontroller side to the Linux side, and vice versa.

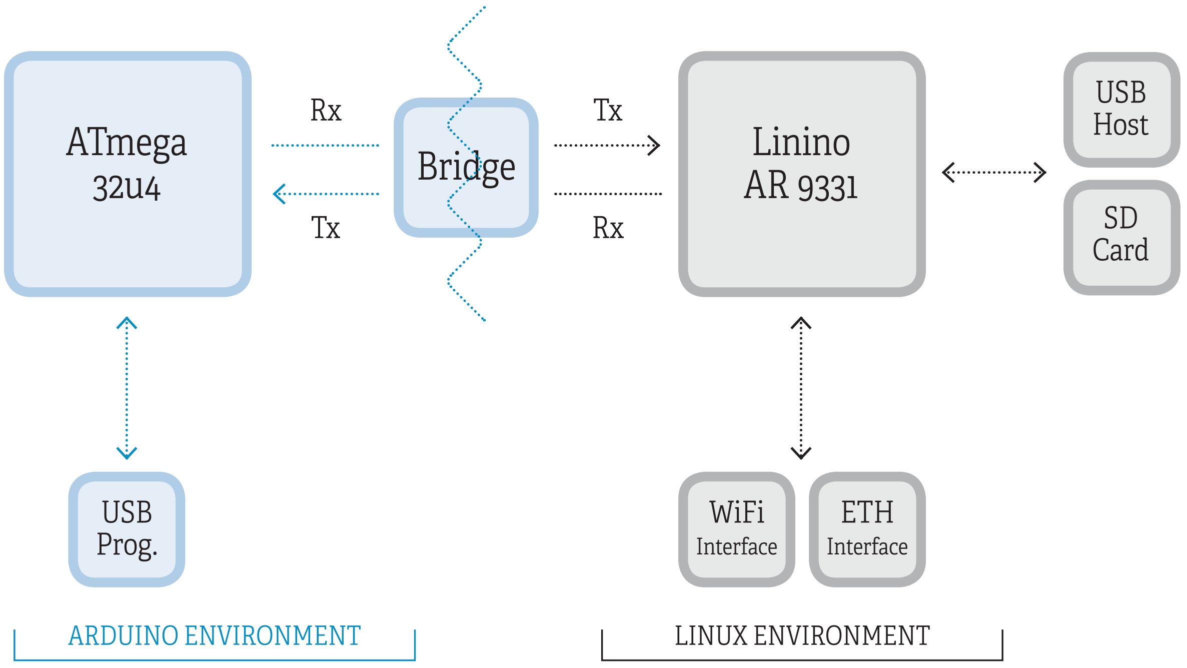


Figure 9 | Bridge

The two processors allow the separation of real-time tasks from tasks without real-time requirements effectively. All tasks that must fulfill real-time requirements should be placed on the microcontroller. Here short routines and/or the interrupt-system can react as expected.

A very good example is the access to 1-Wire devices like the well-known and often-used DS1820 temperature sensor with its hard timing conditions. The access to a DS1820 sensor from a Linux device is not very reliable and it absolutely requires error handling.

Interrupts can help in time-critical tasks. In my Arduino book [8] I include a chapter covering the ATmega interrupts.

The *Bridge* library simplifies communication between the two processors (http://arduino.cc/en/Reference/YunBridgeLibrary). Bridge commands from ATmega32u4 will be interpreted by Python on AtherosAR9331. Initiated by ATmega32u4, commands on the Linux side can be executed. Data exchange

TABLE 2 | Components of the Bridge library

CLASS	DESCRIPTION
Process	Process is used to launch processes on the Linux processor, and other things like shell scripts.
Console	Console can be used to communicate with the network monitor in the Arduino IDE, through a shell. Functionally, it is very similar to Serial.
FileIO	An interface to the Linux file system. Can be used to read/write files on the SD card.
HttpClient	Creates an HTTP client on Linux. Acts as a wrapper for common CURL commands by extending Process.
Mailbox	An asynchronous, session-less interface for communicating between Linux and Arduino.
Bridge Client	An Arduino-based HTTP client, modeled after the EthernetClient class.
Bridge Server	An Arduino based HTTP server, modeled after the EthernetServer class.
Temboo	An interface to Temboo.com making it easy to connect to a large variety of online tools.

between both CPUs is organized by shared memory.

Via the *Bridge* library, the ATmega32u4 is extended with functions that provide network connection and other functions on a Linux device. Conversely, the Linux device gains simpler and faster access to sensors and actuators through this microcontroller supplement.

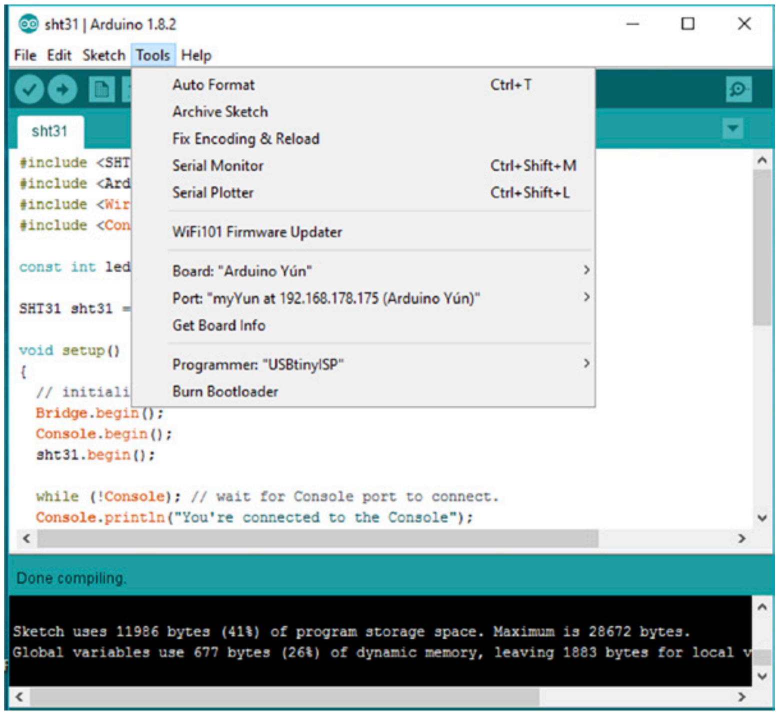
The *Bridge* library contains several classes described in Table 2 (http://arduino.cc/en/Reference/YunBridgeLibrary).

In the next chapter we will have a look into program examples that use the *Bridge* library in the described way.

7 ○ Collecting Weather Data

To collect weather data we have different possibilities. We can use sensors to measure local temperature, humidity, barometric pressure, etc., or we can ask a weather portal on the internet for weather data of a local measuring station.

To visualize these measured



```
#include <SHT
#include <Ard
#include <Wir
#include <Con

const int led

SHT31 sht31 =

void setup()
{
  // initiali
  Bridge.begin();
  Console.begin();
  sht31.begin();

  while (!Console); // wait for Console port to connect.
  Console.println("You're connected to the Console");
```

```
Sketch uses 11986 bytes (41%) of program storage space. Maximum is 28672 bytes.
Global variables use 677 bytes (26%) of dynamic memory, leaving 1883 bytes for local v
```

FIGURE 10 | Arduino IDE

data and to embed them into a website we can use ThingSpeak, for example.

Before we get into the program examples, a few preliminary remarks are necessary. In Chapter 3 I described the firmware update for the Arduino Yún. It is the same for the Arduino IDE. The actual version of Arduino IDE is 1.8.2. You will get it from the Arduino website https://www.arduino.cc/en/Main/Software for Windows, MacOS, and Linux.

I use the WLAN connection for software download and console IO. The micro-USB of the board is used for powering only.

Figure 10 shows the Arduino IDE v1.8.2 with selected 'Arduino Yún' (Board) and 'myYUN at 192.168.178.175' (Port). These are the preconditions for all further steps described here.

Telnet can be used for the communication with the console. If you redirect port 8888 of the Arduino Yún IP address to localhost:6571 then you can access the console from PC by telnet 192.168.178.175 8888 in the configuration here [9].

You have to install socat for this redirection following these steps:

```
$ opkg update
$ opkg install socat
$ (socat TCP-LISTEN:8888,fork TCP:127.0.0.1:6571) &
```

7.1 ○ Temperature and Humidity by SHT31

There are various sensors to measure temperature and humidity. You will find a lot of applications using the DHT11/DHT22. To get better accuracy I use here the Grove SHT31 sensor based on Sensirion's SHT31 sensor [10]. The SHT31 library can be downloaded from Grove's Github https://github.com/Seeed-Studio/Grove_SHT31_Temp_Humi_Sensor.

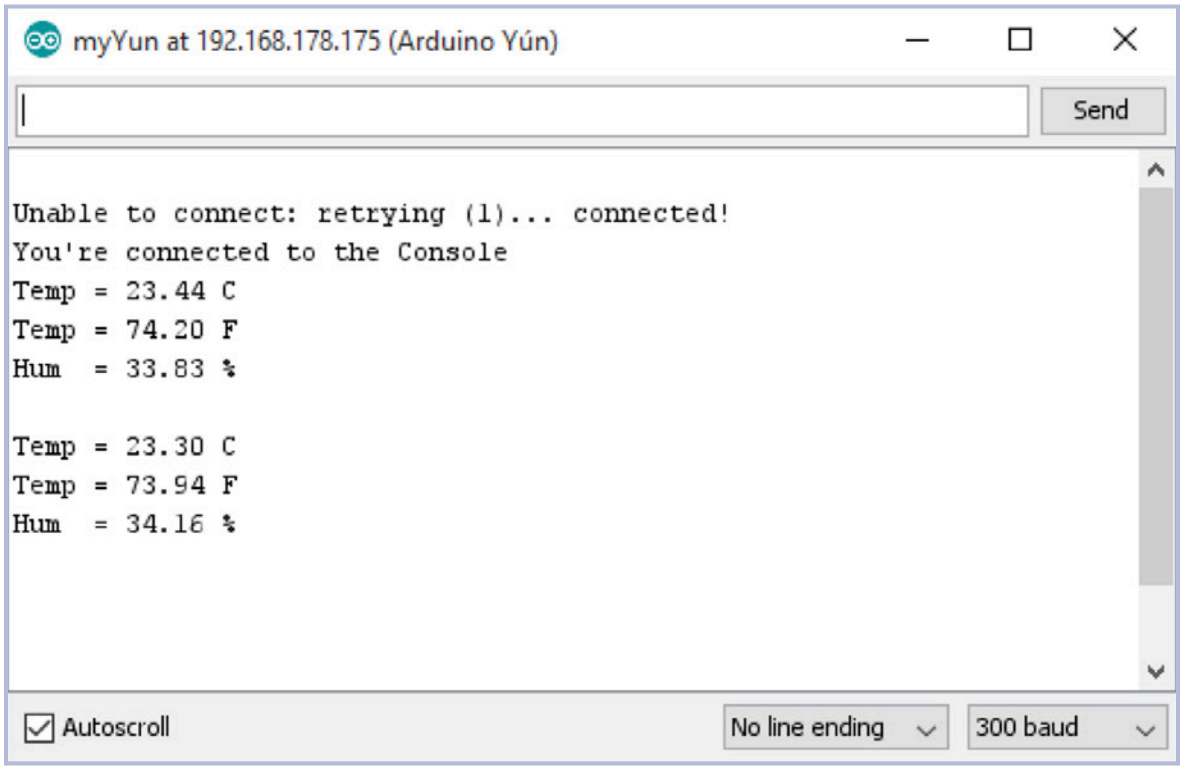
Only a few adaptions to the Grove source code are needed and the sensor values will be sent to the console. The complete access to the sensor via I²C is encapsulated in the SHT31 library. Listing 1 shows the source code of the program *sht31.ino*.

Listing 1 | Source code sht31.ino

```
#include <SHT31.h>
#include <Arduino.h>
#include <Wire.h>
#include <Console.h>

const int ledPin = 13;          // the pin that the LED is attached to
const unsigned int cycle = 60000; // measuring cycle 60 sec

SHT31 sht31 = SHT31();
```

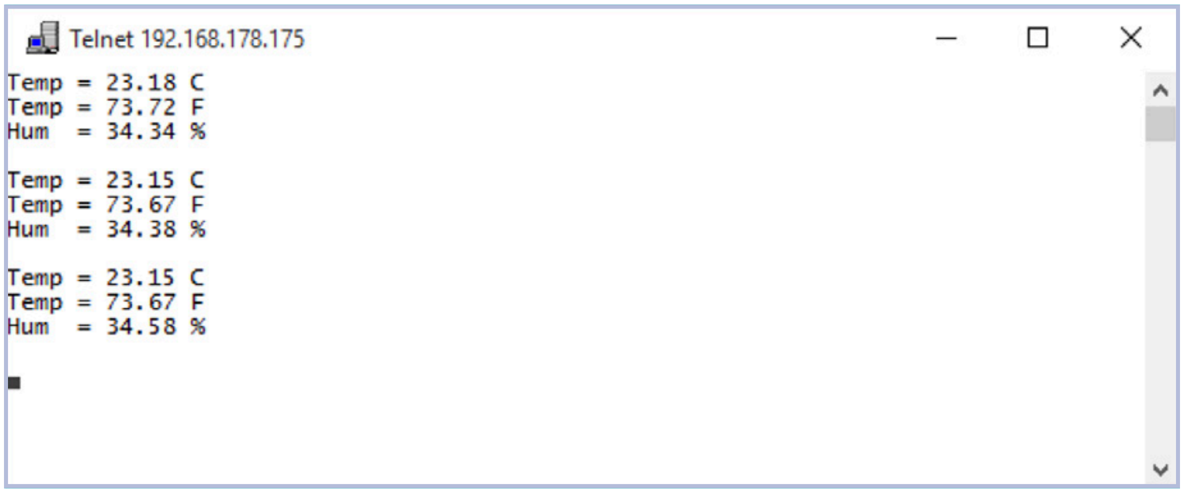
```
void setup()
{
 // initialize serial communication:
 Bridge.begin();
 Console.begin();
 sht31.begin();

 while (!Console); // wait for Console port to connect.
 Console.println("You're connected to the Console");
 // initialize the LED pin as an output:
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 digitalWrite(ledPin, HIGH);   // sets the LED on
 float temp = sht31.getTemperature();
 float hum = sht31.getHumidity();
 Console.print("Temp = ");
 Console.print(temp);
 Console.println(" C"); //The unit for Celsius because original
              //arduino don't support special symbols
 float fTemp = temp * 1.8 + 32;
 Console.print("Temp = ");
 Console.print(fTemp);
 Console.println(" F"); //The unit for Fahrenheit because original
              //arduino don't support special symbols
 Console.print("Hum  = ");
 Console.print(hum);
 Console.println(" %");
 Console.println();
 digitalWrite(ledPin, LOW);   // sets the LED off
 delay(cycle);
}
```

We have two ways to see the console output. The easiest way is to use the serial monitor from Arduino IDE. Figure 11 shows connecting and monitoring the output. The test here was done inside at room conditions. The second way is to call the Telnet client installed on PC and connect to the Arduino Yún by the command o(pen) 192.168.178.175 8888. You will get the same output but independent from the Arduino IDE (Figure 12).



```
Unable to connect: retrying (1)... connected!
You're connected to the Console
Temp = 23.44 C
Temp = 74.20 F
Hum  = 33.83 %

Temp = 23.30 C
Temp = 73.94 F
Hum  = 34.16 %
```

Figure 11 | Serial Monitor of Arduino IDE



```
Temp = 23.18 C
Temp = 73.72 F
Hum  = 34.34 %

Temp = 23.15 C
Temp = 73.67 F
Hum  = 34.38 %

Temp = 23.15 C
Temp = 73.67 F
Hum  = 34.58 %
```

Figure 12 | Telnet client on PC

To have the measured values on the Linux side as well we can use the FileIO class to save these values in RAM. Listing 2 shows the enhancements (marked bold) in the source code.

Listing 2 | Source code sht31fio.ino

```
#include <SHT31.h>
#include <Arduino.h>
#include <Wire.h>
#include <Console.h>
#include <FileIO.h>
```

```
const int ledPin = 13;          // the pin that the LED is attached to
const unsigned int cycle = 60000; // measuring cycle 60 sec

SHT31 sht31 = SHT31();

void setup()
{
 // initialize serial communication:
 Bridge.begin();
 Console.begin();
 sht31.begin();
 FileSystem.begin();

 while (!Console); // wait for Console port to connect.
 Console.println("You're connected to the Console");
 // initialize the LED pin as an output:
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 digitalWrite(ledPin, HIGH);   // sets the LED on
 float temp = sht31.getTemperature();
 float hum = sht31.getHumidity();
 Console.print("Temp = ");
 Console.print(temp);
 Console.println(" C"); //The unit for Celsius because original
                 //arduino don't support special symbols
 float fTemp = temp * 1.8 + 32;
 Console.print("Temp = ");
 Console.print(fTemp);
 Console.println(" F"); //The unit for Fahrenheid because original
                 //arduino don't support special symbols
 Console.print("Hum  = ");
 Console.print(hum);
 Console.println(" %");
 Console.println();
 File f = FileSystem.open("/tmp/TEMP", FILE_WRITE);
 f.print(temp);
```

```
  f.close();
  File ff = FileSystem.open("/tmp/TEMPF", FILE_WRITE);
  ff.print(fTemp);
  ff.close();
  File fff = FileSystem.open("/tmp/HUMI", FILE_WRITE);
  fff.print(hum);
  fff.close();
  digitalWrite(ledPin, LOW);   // sets the LED off
  delay(cycle);
}
```

7.2 ○ Query Weather Underground

If you lack access to a local weather station but are interested in local weather data, then weather data from Weather Underground (http://www.wunderground.com) can help to close this gap.

The last program examples were compiled for the ATmega32u4 in the Arduino IDE. For the task here we need the Linux device only. On most Linux devices you also have the GNU C compiler *gcc* available to compile application programs directly on the target. Here, under OpenWRT, this is not the case. Hence, it takes a suitably equipped developer's PC to compile a C/C++ source code for our target. But this is no problem. We have Python available on Arduino Yún, and I built *wunderweather.py* asking Weather Underground for local weather data. Local means here my location, Altendorf in Switzerland.

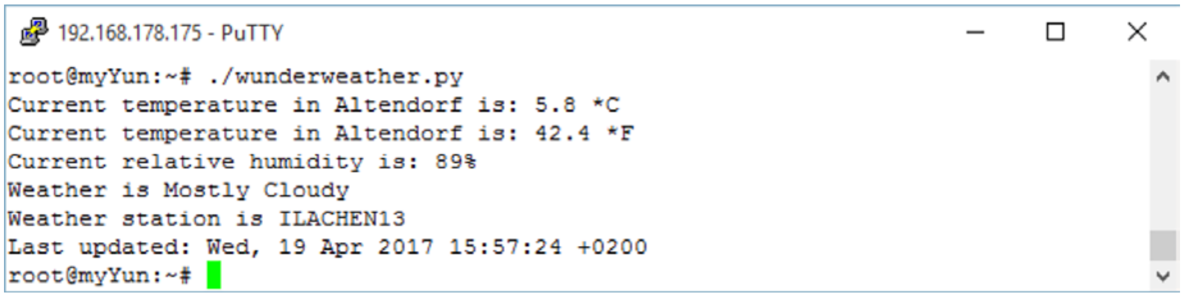
Listing 3 shows the Python source code of *wunderweather.py*. You have to create an account at Weather Underground to get an API key and should adapt your location. After output the weather data temperature and humidity will be saved as variables in RAM.

LISTING 3 | Source code wunderweather.py

```
#!/usr/bin/python

# Reading weather data from wunderground network

import urllib2
import json

f = urllib2.urlopen('http://api.wunderground.com/api/<API_KEY>/
                      geolookup/conditions/q/Switzerland/Altendorf.json')
json_string = f.read()
parsed_json = json.loads(json_string)

location = parsed_json['location']['city']
temp_c  = parsed_json['current_observation']['temp_c']
```

```
temp_f = parsed_json['current_observation']['temp_f']
rel_hum = parsed_json['current_observation']['relative_humidity']
weather = parsed_json['current_observation']['weather']
station = parsed_json['current_observation']['station_id']
updated = parsed_json['current_observation']['observation_time_rfc822']
print("Current temperature in %s is: %s *C" % (location, temp_c))
print("Current temperature in %s is: %s *F" % (location, temp_f))
print("Current relative humidity is: %s " % (rel_hum))
print("Weather is %s " % (weather))
print("Weather station is %s" % (station))
print("Last updated: %s" % (updated))

f.close()

f = open("/tmp/TEMP","w")
f.write(str(temp_c))
f.close()

f = open("/tmp/TEMPF","w")
f.write(str(temp_f))
f.close()

f = open("/tmp/HUMI","w")
f.write(str(rel_hum)[:-1])
f.close()
```

Figure 13 shows the call and output of the program *wunderweather.py*. After a wonderful springtime we got a winter break again.



192.168.178.175 - PuTTY

```
root@myYun:~# ./wunderweather.py
Current temperature in Altendorf is: 5.8 *C
Current temperature in Altendorf is: 42.4 *F
Current relative humidity is: 89%
Weather is Mostly Cloudy
Weather station is ILACHEN13
Last updated: Wed, 19 Apr 2017 15:57:24 +0200
root@myYun:~#
```

Figure 13 | Call and output of *wunderweather.py*

7.3 ○ Visualize Weather Data by ThingSpeak

To visualize the requested weather data we will upload it to the ThingSpeak, an open IoT platform in the cloud. As usual, you have to create an account. After that you can collect data in a so-called channel. This channel has an API key required by the program sending data to this channel.

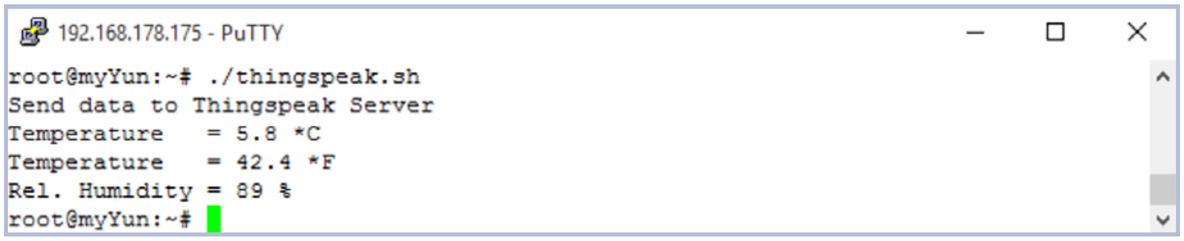
Listing 4 shows the source of the Shell script *thingspeak.sh*. This script reads temperature and humidity saved as variables in RAM and sends it by *cURL* to the ThingSpeak server.

Listing 4 | Source code thingspeak.sh

```
#!/bin/sh

echo "Send data to Thingspeak Server"

#Thingspeak
api_key='<API_Key>'

DATE="$(date +"%d-%m-%Y")"
read TEMP < /tmp/TEMP
echo "Temperature  = $TEMP *C"
read TEMPF < /tmp/TEMPF
echo "Temperature  = $TEMPF *F"
read HUMI < /tmp/HUMI
echo "Rel. Humidity = $HUMI %"

curl --insecure --data \
  "api_key=$api_key&field1=$TEMP&field2=$TEMPF&field3=$HUMI&field4=$DATE" \
  https://api.thingspeak.com/update  > log 2>&1
```

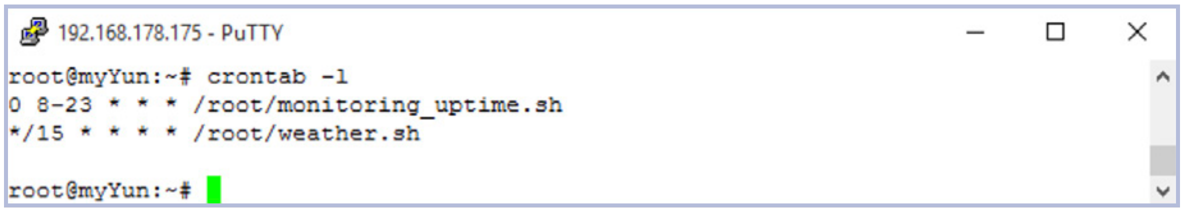
Figure 14 shows the call and output of the program *thingspeak.sh*.

To automate the process of requesting weather data I placed the Shell script *weather.sh* into the Crontab (Figure 15).



```
root@myYun:~# ./thingspeak.sh
Send data to Thingspeak Server
Temperature    = 5.8 *C
Temperature    = 42.4 *F
Rel. Humidity = 89 %
root@myYun:~#
```

Figure 14 | Call and output *thingspeak.sh*



```
root@myYun:~# crontab -l
0 8-23 * * * /root/monitoring_uptime.sh
*/15 * * * * /root/weather.sh

root@myYun:~#
```

Figure 15 | List of Crontab entries

The script itself is very simple (Listing 5). You can use *sht31.ino* or *wunderweather.py* to generate the weather data, and *thingspeak.sh* will upload the data afterwards. If you use *sht31.ino* then comment the call of *wunderweather.py*. Otherwise, replace *sht31.ino* by *blink.ino* as heartbeat. You will find it on my Github, too.

Listing 5 | Source code weather.sh

```
#!/bin/sh

/root/wunderweather.py
/root/thingspeak.sh

echo “Done.”
```

Finally, you can see a day profile of temperature and humidity from the 19th to 20th of April 2017 for my location here in Switzerland (Figure 16). You can find actual profiles at https://thingspeak.com/channels/255753.

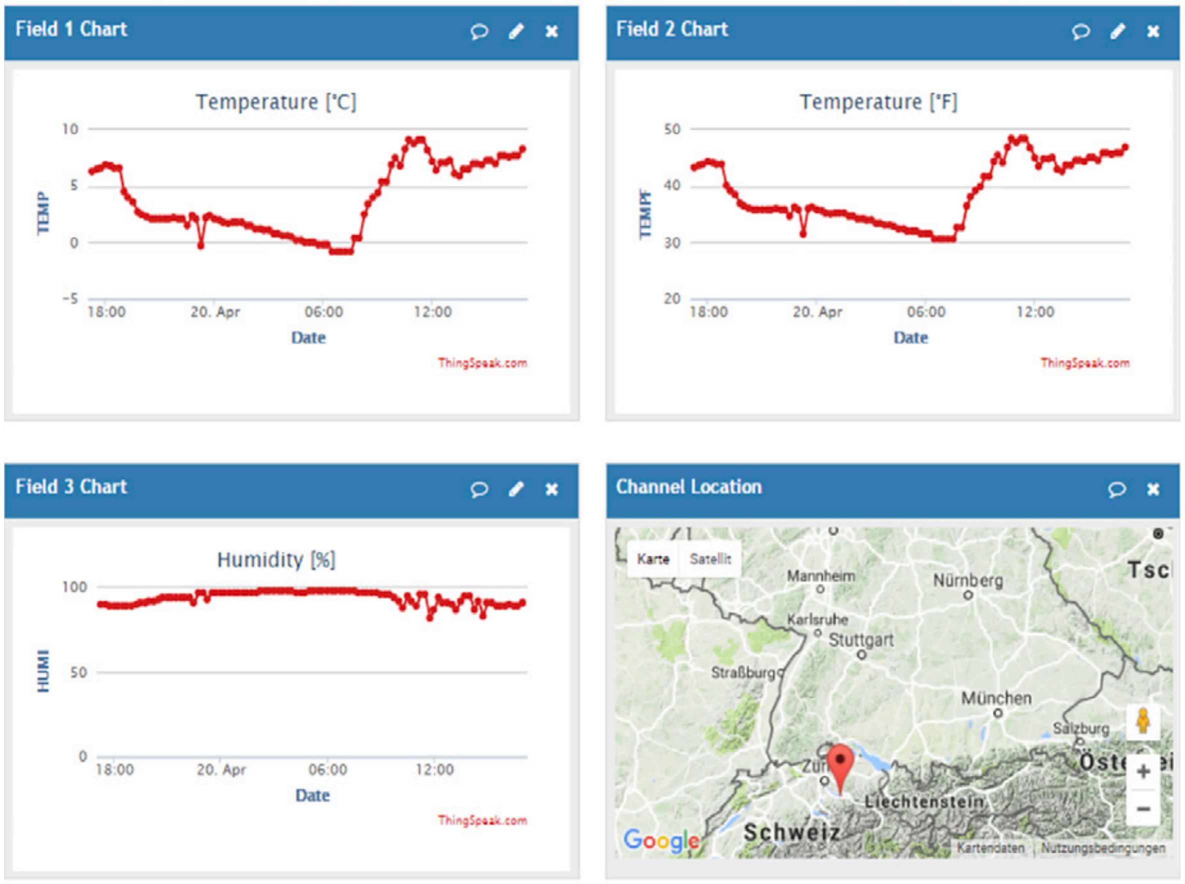


Figure 16 | Data visualization by ThingSpeak

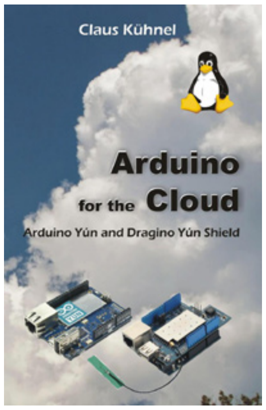
8 ○ Conclusion

This eBook describes the Arduino Yún configured for cloud applications. Due to the combination of microcontroller and Linux device you can separate effectively real-time tasks from tasks that need network access.

All programs introduced here are saved on my Github (https://github.com/ckuehnel/arduino) for download.

Further explanations to Arduino Yún and the Arduino Yún shield as an add-on for a conventional Arduino can be found in my book, *Arduino for the Cloud*.

Have fun with Arduino Yún!



About the Author

Dr. Claus Kuhnel studied at and graduated from the Technical University of Dresden (D) in the field of information electronics. This was followed by an education in biomedical engineering. Professionally, he is responsible for development of embedded systems for lab devices. In addition to his professional tasks, he has published numerous articles and books on microcontroller-related issues.

9 ○ References

[1] Creative Commons Attribution Share-Alike license
https://creativecommons.org/licenses/by-sa/1.0/

[2] Kuhnel, C.:
Arduino for the Cloud - Arduino Yun and Dragino Yun Shield
ISBN 978-1-62734-035-9
http://www.universal-publishers.com/book.php?method=ISBN&book=1627340351

[3] Dragino Yùn shield
http://www.dragino.com/products/yunshield/item/86-yun-shield.html

[4] GNU GENERAL PUBLIC LICENSE
https://www.gnu.org/licenses/gpl-3.0.txt

[5] GNU LESSER GENERAL PUBLIC LICENSE
https://www.gnu.org/licenses/lgpl-3.0.txt

[6] TWO ARDUINOS BECOME ONE
https://blog.arduino.cc/2016/10/01/two-arduinos-become-one-2/

[7] How to Measure Arduino Network Performance
http://www.instructables.com/id/How-to-measure-Arduino-network-performance/

[8] Kühnel, C.:
Arduino - Hard- und Software Open Source Plattform (German language)
ISBN 978-3-907857-16-8
https://www.amazon.de/Arduino-Hard-Software-Source-Plattform/dp/390785716X

[9] Console Class
http://www.ibuyopenwrt.com/index.php/8-yun-compatible/158-console-class

[10] Datasheet SHT3x-DIS - Humidity and Temperature Sensor
https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/2_Humidity_Sensors/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital.pdf

The future ain't what it used to be...

at Technica Curiosa, it's so much more.

TechnicaCuriosa is the new and exciting hub of a highly connected family of iconic media brands—brands endowed with rich legacies of world-changing, decades-spanning influence.

Because new discoveries and innovation come from connecting non-obvious and disparate dots, we've architected our titles to cross-pollinate big ideas in fascinating ways. In doing so, we expand the field of view—and include you in it.

SUBSCRIBE NOW It's FREE

@TechnicaCuriosa

Turbine-powered Firebirds | Artificial Photosynthesis | Mining Helium-3 on the Moon
Mechanix Illustrated
August 2017 / Vol.1 • No.1 / www.TechnicaCuriosa.com
Architecture for the 22nd Century
Asteroid-suspended Skyscrapers
Goodbye Knobs Ready for Mid-air Haptics?
Special Focus Self-driving Cars
Shuttle Astronaut Mike Massimino
Engineers His Way to Space

Popular Electronics
August 2017
www.TechnicaCuriosa.com $5.95
ALL SIGNAL, NO NOISE
SPECIAL FOCUS: The Brain-computer Interface
Build it Yourself
Harvest Free Energy
Open-sourcing human-machine interaction
Getting Started with Machine Learning
Bonus Feature! FREE eBook Arduino for the Cloud
FPGA LEARNING LAB | HARNESS VISIBLE LIGHT FOR THE INTERNET OF THINGS | EXPLOIT WIFI BACKSCATTER AND DITCH THE BATTERIES