Sistem za merjenje in pregled temperature

Seminarska naloga pri predmetu Brezžična senzorska omrežja

Jure Jesenšek, 63120159, jj4001@student.uni-lj.si
Marko Lavrinec, 63130134, ml9987@student.uni-lj.si

Junij 2018

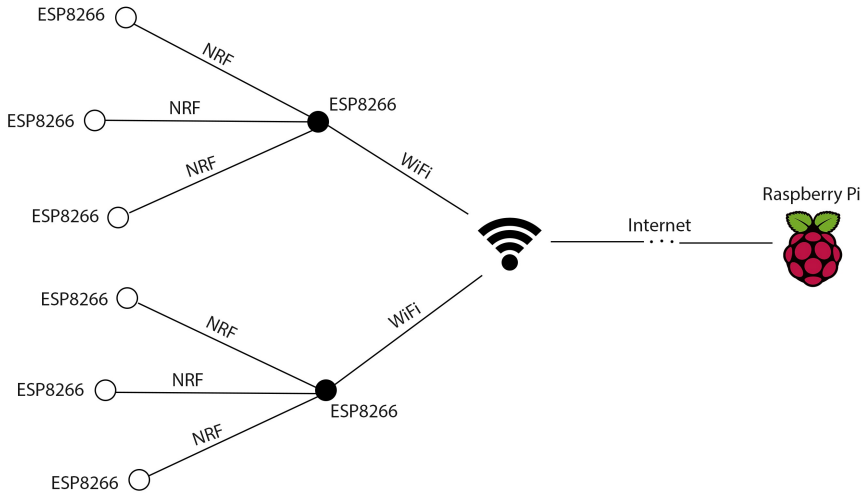
1 Uvod

V tem delu bo predstavljena realizacija rešitve, kjer več naprav spleta stvari (angl. `Internet of Things - IoT`) med seboj komunicira in si pošilja podatke. Glavna vozlišča podatke sprejemajo, nato pa zbrane podatke preko brezžične povezave WiFi pošljejo v obdelavo na strežnik. Rezultate meritev si lahko nato uporabnik ogleda v poljubnem spletnem brskalniku tako tekstovno kot tudi grafično na grafih. V našem primeru bomo obdelovali temperaturne podatke, ki jih bodo naše naprave izmerile in posredovale.

2 Opis problema

Pri predmetu Brezžična senzorska omrežja smo si za seminarsko nalogo izbrali problem merjenja, zbiranja, posredovanja in prikazovanja temperature. V tem delu bo opisan postopek vzpostavitve povezav med več IoT napravami, ki bodo svoje podatke (temperaturo) pošiljale med vozlišči, glavna vozlišča pa jih bodo nato poslala na strežnik v nadaljnjo obravnavo.

Temperaturo bomo merili s senzorjem BMP280 (del modula GY-91), ki je preko vodila I2C povezan s ploščico ESP8266. Le-ta vsebuje WiFi modul za komunikacijo s strežnikom. Poleg komunikacije preko WiFi-ja pa bodo ploščice med seboj komunicirale (posredovale temperaturo) preko čipov NRF, ki so prav tako s procesno enoto priključene preko vmesnika I2C. Za uporabo vmesnika NRF smo se odločili zaradi manjše porabe energije kot pri WiFi-ju, poleg tega pa lahko z uporabo tega vmesnika pokritost prostora s temperaturnimi senzorji razširimo tudi izven dosega WiFi dostopne točke. Glavna vozlišča, ki so povezana preko povezave WiFi, zbranim meritvam temperature dodajo še svoje meritve, nato pa preko brezžičnega povezave meritve pošljejo na MQTT odjemalca. Raspberry Pi vrši funkcijo MQTT posrednika (angl. *broker*) in odjemalca ter HTTP strežnika, preko katerega bo lahko uporabnik sledil zgodovini meritev.



Slika 1: Skica omrežja.

Na Sliki 1 je prikazana zgoraj opisana skica omrežja. Tu imamo torej končne naprave ESP8266 (bele pike), ki merijo temperaturo in jo prek NRF-ja pošiljajo na glavna vozlišča (črne pike). Končne naprave zaradi manjše porabe energije, ter zaradi neodvisnosti od omrežja, ne uporabljajo povezave WiFi. Glavna vozlišča so prav tako sestavljena iz ploščic ESP8266. Te so povezane na brezžično omrežje WiFi in preko protokola MQTT pošiljajo podatke na ploščico Raspberry Pi. Poleg MQTT strežnika na njem teče tudi HTTP strežnik namenjen demonstraciji zajetih podatkov.

3 Postavitev okolja

Za namen te seminarske naloge smo si pripravili javno dostopen GitHub repozitorij [5], za skupno pripravo in obdelavo izvorne kode, ki jo bodo poganjale naše naprave.

Na ploščici ESP8266 teče FreeRTOS [2], ki olajša programiranje večopravilnih vgrajenih sistemov. Knjižnica FreeRTOS je napisana v jeziku C/C++, ki smo ga uporabili tudi za pisanje naše naloge. Poleg jedra FreeRTOS smo prenesli in namestili razvojno okolje `esp-open-sdk` [1]. Za pisanje in urejanje kode smo uporabili program Clion podjetja JetBrains.

Projekt smo zaradi boljše preglednosti in večje modularnosti strukturirali in razdelili v več C++ razredov, te pa v več datotek (`.cpp` in `.h`) (opisani v poglavju 5). Funkcionalnosti, kot so branje temperaturnega senzorja, branje stanja gumbov, pošiljanje MQTT sporočil in komunikacija preko NRF, smo

ločili od glavnega dela programa ter jih od samega začetka razvijali v ločenih razredih.

Na Raspberry Pi smo si namestili posrednik Eclipse Mosquitto Broker in odjemalec Eclipse Mosquitto Client, ki sta namenjena delu s protokolom MQTT. Na napravo smo še namestili strežnik Apache2, bazo SQLite in PHP, ki jih bomo uporabljali za obdelavo in prikaz pridobljenih podatkov.

Na našem omrežnem usmerjevalniku smo na strani svetovnega spleta odprli vrata 80 za spletni strežnik in vrata 21883, ki se v lokalnem omrežju preslikajo v vrata s številko 1883, preko katerih privzeto komunicira MQTT. Promet, ki prihaja na ta vrata, smo preusmerili na Raspberry Pi.

3.1 Eclipse Mosquitto

Eclipse Mosquitto [4] je brezplačna odprtokodna implementacija MQTT strežnika. Protokol MQTT kot tudi sama implementacija naj bi bili varčni pri porabi z energijo, zaradi česar sta še posebej primerna za uporabo pri spletu stvari.

Na Raspberry Pi, na katerem teče Debian distribucija OS Linux, smo z ukazom:

```
sudo apt-get install mosquitto mosquitto-clients
```

namestili Eclipse Mosquitto. Po namestitvi smo v nastavitvah programa na lokaciji /etc/mosquitto/mosquitto.conf nastavili, da program deluje na vratih 1883, ter da smejo sporočila pošiljati in prejemati samo uporabniki, ki poznajo uporabniško ime in geslo. Uporabnike in njihova gesla smo shranili v /etc/-mosquitto/pwfile [3].

Z ukazom:

```
sudo systemctl enable mosquitto.service
```

smo nato nastavili, da se program zažene skupaj z operacijskim sistemom, s čimer smo zagotovili da bo sporočilna vrsta delovala tudi v primeru ponovnega zagona naše naprave.

Na Android mobilno napravo smo namestili brezplačen program MQTT Dashboard, s katerim smo preverili delovanje naših nastavitev strežnika MQTT. V nadaljevanju bi lahko ta program uporabljali tudi v realnih primerih za sprotno preverjanje temperatur in morebitnih drugih podatkov, ki bi jih pošiljali v MQTT vrsto [6].

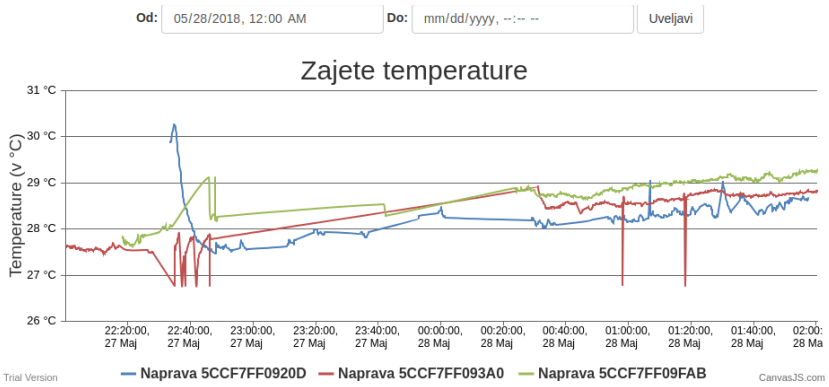
3.2 Spletni strežnik

Na Raspberry Pi smo namestili spletni strežnik Apache2, podporo za programski jezik PHP in bazo SQLite za hranjenje podatkov.

Napisali smo skripto za ukazno lupino bash, ki se poveže na MQTT temo (angl. *topic*) in ob vsaki pojavitvi sporočila pokliče našo PHP skripto, ki meritev shrani v bazo. Za ta namen smo naredili 2 tabeli v bazi. Prva tabela (*messages*) hrani vsa sporočila ne glede na to ali so pravilno formatirana in vsebujejo podatke o temperaturi. Dodali pa smo še tabelo *results*, ki hrani samo rezultate

meritev, napravo ki jih je poslala, čas meritve in poljuben parameter senzor s katerim je bilo merjeno. Parameter senzor bi lahko uporabili v primeru, ko bi želeli imeti zanesljivejše merjenje temperature in bi v ta namen poleg čipa BMP280 uporabili tudi čip MPU-9250.

Na spletni strežnik smo dodali tudi skripte za prikaz sporočil in izris grafa temperatur. Primer izrisanega grafa meritve prikazuje slika 2. Zelena naprava na sliki je rdeči napravi preko vmesnika NRF pošiljala svoje podatke. Rdeča in modra naprava pa sta bili neposredno povezani na brezžično omrežje Wifi in sporočilno vrsto MQTT. Izrisovanje grafa je mogoče poljubno filtrirati po datumu meritve.



Slika 2: Graf meritev.

Uporabniku je na voljo tudi stran za ogled vseh sporočil, ki so bila posredovana na naš MQTT strežnik, vendar pa je stran z grafom veliko preglednejša in lažje razumljiva.

Graf privzeto izrisuje zadnjih deset tisoč meritev vseh naprav, uporabnik pa si lahko nato sam izbere poljubni časovni interval, v katerem želi videti graf temperatur.

3.3 Zanesljivost postavitve

Pri zanesljivosti storitev je potrebno imeti v mislih dejstvo, da lahko pride pri delovanju tudi do izpada električne energije ali pa spletne povezave. V primeru kakršnega koli izpada se naprave ob ponovnem zagonu vrnejo na začetno stanje brez posredovanja sistemskega administratorja. Med drugim se tudi naše naprave ponovno vključijo ob ponovnem prejemu električne energije brez posredovanja. Tako izgubimo samo podatke in zahtevke, do katerih pride v času nedelovanja storitve. Ob ponovnem zagonu pa se na novo samodejno vključita tudi MQTT in spletni strežnik. Torej lahko rečemo, da naš izdelek deluje po principu *plug and play*.

4 Delovanje naprave

Naprava ESP8266 se pri zagonu postavi v stanje izvajanja načina `NRF_CLIENT`. Naprava v tem stanju meri temperaturo in jo preko vmesnika NRF pošilja drugi napravi. Komunikacija preko WiFi in pošiljanje sporočil MQTT je onemogočeno.

Uporabnik pa lahko s pritiskom na gumb 4 (gumb najbližje modulu GY-91 - skrajno desni gumb na sliki 3) delovanje prestavi v t.i. `NRF_SERVER_AND_MQTT_CLIENT` način delovanja. V tem načinu se komunikacija preko NRF prestavi iz oddajanja v sprejemanje, vklopi pa se tudi povezava preko WiFi in pošiljanje sporočil MQTT posredniku.

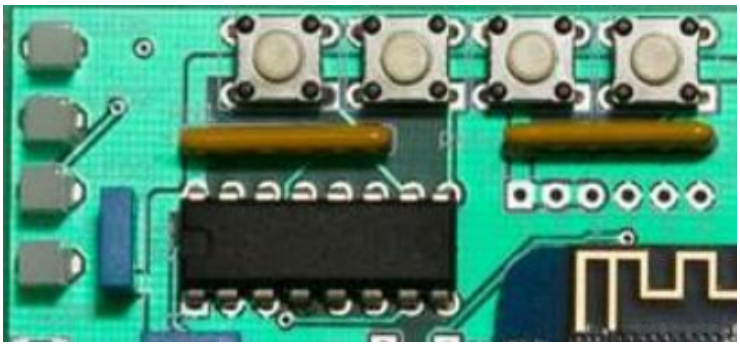
Preklop v načinu izvajanja se izvede s pomočjo ustavitve oziroma zagona primernih opravil (angl. *task*). Uporabljeni sta funkciji `vTaskSuspend()` in `vTaskResume()`, ki preko oprimka na opravilo (angl. *task handle*) le-tega spravita v stanje `Suspended` oziroma `Running` ali `Ready`.

Sporočila, ki vsebujejo temperaturo so sledeče oblike:

NASLOV_MAC:TEMPERATURA (npr.: 5CCF7FF093A0:28.890)

Glavne naprave pa tudi same merijo temperaturo, ki jo poleg prejetih sporočil posredujejo MQTT posredniku.

S pritiskom na gumb 1 (najbolj oddaljen od modula GY-91 - skrajno levi gumb na sliki 3) pa se izvajanje vrne nazaj v prvoten način (`NRF_CLIENT`).



Slika 3: Gumbi. Gumb 1 je skrajno levi gumb, gumb 4 pa skrajno desni.

5 Razčlenitev kode

Glavni del kode je napisan v datoteki `main.cpp`. V njej se pripravi vse potrebno za delovanje sistema. Nastavi se FreeRTOS, inicializirajo potrebni razredi, spremenljivke, semaforji in vrste.

Ker projektni Makefile podpira uporabo jezika C++ (poleg C) smo pri pisanju kode uporabili tudi konstrukte tega jezika. Zavoljo lepše uporabe končne kode, smo vso napisano kodo (razen `main.cpp`) postavili v imenski prostor `namespace TempMonitor`. Prav tako smo se izogibali definicije konstant preko

ukazov za predprocesor (`#define`), temveč smo raje uporabili modifikatorja tipa `constexpr`, oziroma `const`. Funkcije za krmiljenje modulov smo logično razporedili v razrede, ki so sestavljeni iz `.cpp` in `.h` datotek, o katerih pa več piše v naslednjih podpoglavjih.

5.1 Temperaturni senzor BMP280

Vmesnik razreda `Bmp280_temp_sensor`, ki skrbi za temperaturni modul BMP280 je sestavljen iz naslednjih metod:

- `Bmp280_temp_sensor() = default` - trivialen konstruktor.
- `bool init()` - metoda za začetno inicializacijo potrebnih spremenljivk, struktur, parametrov in ostalih nastavitev. Vrne `true`, če je bila inicializacija uspešna.
- `bool initI2C()` - metoda za (ponovno) nastavitev I2C vodila. Vrne `true`, če je bila inicializacija vodila uspešna. Metoda je predvsem uporabljena zaradi uporabe večih modulov, ki so priklopljeni na vodilo I2C. S to metodo se vodilo ponovno nastavi na način, kot ga potrebuje modul BMP280.
- `float read()` - prebere (prenese) izmerjeno temperaturo iz modula. V kolikor je prišlo do napake pri branju, ali v primeru neprimerno inicializirane naprave ali vodila, metoda vrne konstanto `NaN`.

5.2 Gumbi

Za preklapljanje med različnimi načini izvajanja smo uporabili 2 izmed 4 gumbov, ki se nahajajo na ploščici. Razred `Button_reader` vsebuje naslednje metode:

- `void init()` - inicializira I2C in GPIO v stanje, primerno za branje statusa gumbov.
- `bool read()` - prebere trenutno stanje gumbov v notranjo spremenljivko. Vrne `true`, če je branje uspelo. Kateri gumb je pritisnjen pa preverimo z naslednjimi metodami.
- `bool button1_pressed() const` - vrne `true`, če je bil ob času branja pritisnjen prvi gumb. Podobne metode obstajajo tudi za 2., 3. in 4. gumb.

Branje in vračanje stanja gumbov smo ločili na dve metodi zaradi želje po čim manjšem številu preklopov nastavitev na I2C vodilu. Prav tako lahko z enim branjem izvemo stanja vseh štirih gumbov.

5.3 Odjemalec MQTT

Za pošiljanje sporočil MQTT (angl. *MQTT publish*) smo napisali vmesnik `Mqtt_client`. Sestavljen je iz naslednjih metod:

- `Mqtt_client()` - preprost konstruktor.
- `bool init(const char *const client_id, uint8_t client_id_len)` - inicializira odjemalca. Argument `msg` določi ime (ID) odjemalca, `client_id_len` pa dolžino niza, ki vsebuje ime. Ime se uporabi za identifikacijo naprave kot del poslanih sporočil s temperaturo. Trenutno se za identifikacijo uporabi naslov MAC WiFi modula. Metoda vrne `true`, če je bila inicializacija uspešna.
- `bool connect()` - vzpostavi povezavo s posrednikom (angl. *MQTT broker*). Vrne `true`, če je bila vzpostavitev uspešna.
- `bool publish(const char *const msg, const uint8_t msg_len)` - posredniku pošlje sporočilo `msg` dolžine `msg_len`.
- `bool yield(const uint32_t duration_ms = MQTT_COMMAND_TIMEOUT_MS)` - obdrži povezavo s posrednikom živo, in blokira izvajanje za določen čas. Uporabljeno tudi kot zamik (angl. *delay*) med pošiljanji. Trenutno se za privzeto vrednost uporabi 5000 ms (5 sekund).
- `void disconnect()` - posredniku pošlje `MQTT disconnect` paket s katerim se prekine povezava.

V ta razred bi spadala tudi metoda za branje iz MQTT strežnika, vendar pa te funkcionalnosti na ploščicah ESP8266 nismo potrebovali. Za branje iz vrste MQTT smo namreč uporabili Raspberry Pi.

5.4 NRF komunikacija

Za komunikacijo preko čipa NRF skrbi razred `Nrf_comm` z metodami:

- `Nrf_comm() = default` - trivialni konstruktor.
- `void init()` - pripravi modul za komunikacijo.
- `bool send(char * data, uint8_t len)` - prejemniku (poslušalcu) pošlje sporočilo `data`, dolžine `len`. Vrne `true`, če je bilo pošiljanje uspešno.
- `bool receive(char *buffer, uint8_t len)` - sprejme poslano sporočilo in ga zapiše v medpomnilnik (angl. *buffer*) `buffer`. Zapiše največ `len` znakov. Vrne `true`, če je bilo prejemanje uspešno.

5.5 Pomožni razredi

Poleg do sedaj naštetih razredov smo napisali tudi razne razrede, ki so služili začasnemu hranjenju podatkov, razrede, ki so vsebovali pogosto uporabljene konstante, ipd. Napisali smo naslednje datoteke oziroma razrede:

5.5.1 `constants.h`

Datoteka vsebuje globalne konstante, ki se uporabljajo za:

- nastavljanje I2C vodila: baud rate, številke nožic (angl. *pin number*), naslovi na vodilih, ipd.
- dolžine medpommnilnikov (na primer za pošiljanje MQTT sporočil), dolžine črkovnih nizov (angl. *string*) (na primer za ime MQTT odjemalca ali pa niza, ki vsebuje sporočilo o temperaturi...) in dolžine vrst (za prejemanje in pošiljanje sporočil)

5.5.2 `device_role.h`

Vsebuje `enum class Device_role`, ki je uporabljen za določanje načina izvajanja kode na ploščici. Enum vsebuje vrednosti `NRF_CLIENT` in `NRF_SERVER_AND_MQTT_CLIENT`. Ko je globalna spremenljivka tipa `Device_role` v `main.cpp` nastavljena na `NRF_CLIENT`, je modul v stanju, ko meri temperaturo in meritve posreduje preko NRF. Ko pa je globalna spremenljivka nastavljena na `NRF_SERVER_AND_MQTT_CLIENT`, pa modul sprejema meritve druge ploščice preko NRF, ta sporočila združi s svojimi meritvami, in le-te posreduje preko WiFi in MQTT proti MQTT posredniku. To vrednost spreminjamo s pritiski na gumbe, kot je opisano v poglavju 4.

5.5.3 `secret_constants.h`

Datoteka vsebuje konstante, ki se uporabljajo pri nastavitvi komunikacije preko MQTT. Konstante so:

- `MQTT_HOST`: IP naslov MQTT posrednika,
- `MQTT_TOPIC` in `MQTT_TEST_TOPIC`: tema (angl. *topic*), na katero se pošiljajo MQTT sporočila,
- `MQTT_PORT`: vrata na katerih posluša MQTT posrednik,
- `MQTT_USER` in `MQTT_PASS`: uporabniško ime in geslo.

Datoteka je namenoma izvzeta iz sledenja zgodovine in shrambe git z vnosom v datoteki `.gitignore`, saj smo se želeli izogniti javnemu razkrivanju teh podatkov. Zaradi lažje uporabe uporabnikom, ki so bolj domači z jezikom C, kot pa s C++, smo konstante definirali na "C-jevski način" z ukazi predprocesorju `#define`, in ne z modifikatorji tipa `const` oziroma `constexpr`.

5.5.4 Mapa `wifi_config`

Mapa vsebuje datoteki `ssid_config.h` in `private_ssid_config.h`. Druga datoteka vsebuje konstante za vzpostavitev povezave preko WiFi: `WIFI_SSID` in `WIFI_PASS`, ki sta zaradi lažje uporabe prav tako definirani z ukazom `#define`.

Ker datoteka prav tako vsebuje informacije, ki jih ne želimo deliti, je za datoteka izključena iz sledenja, kot je opisano v poglavju 5.5.3. Uporabnik na vsakem računalniku ustvari svojo datoteko s potrebnimi konstantami. Datoteka `ssid_config.h` pa vsebuje kratko obrazložitev, navodila za uporabo izključene datoteke in vključuje (z ukazom `#include`) datoteko `private_ssid_config.h`. V `main.cpp` vključimo zaglavno datoteko `ssid_config.h`.

5.6 Notranja sinhronizacija in shranjevanje meritev

Ker večje število naprav hkrati komunicira preko vodila I2C, smo z konstrukti `mutex` uvedli kritične sekcije okoli komunikacije z vodilom I2C. Tako sme le ena naprava naenkrat uporabljati vodilo, saj bi lahko v nasprotnem primeru prišlo do mešanja nastavitev med različnimi moduli. Uporabili smo že definiran tip `SemaphoreHandle_t` in funkciji `xSemaphoreTake()` in `xSemaphoreGive()`.

Za komunikacijo med različnimi procesi (zajemanje temperature, pošiljanje in sprejemanje sporočil preko NRF in posredovanje MQTT sporočil preko WiFi) smo uporabili vrste `QueueHandle_t`. Pri zajetju temperature ali po prejetju le-te preko vmesnika NRF, se dobljena vrednost shrani v vrsto. Opravilo, ki temperaturne vrednosti posreduje preko MQTT ali NRF, pa zapise prebere iz vrste in pošlje naprej.

6 Izzivi med delom in ostale ugotovitve

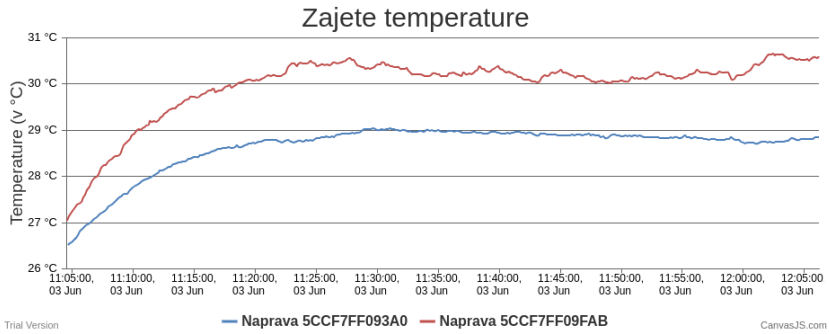
Med programiranjem našega izdelka smo naleteli na kar nekaj izzivov in težav. Prav tako smo prišli do nekaterih ugotovitev glede delovanja naprav in modulov. Nekaj izmed njih je opisanih v nadaljevanju.

6.1 Napačne meritve zaradi segrevanja

Tekom razvijanja programske kode smo opazili opazno razliko med izmerjeno temperaturo in dejansko temperaturo prostora, kjer smo se nahajali. Izmerjena temperatura je bila za nekaj stopinj višja kot bi morala biti. Pri pregledovanju grafov meritev temperature (vidno na sliki 4) smo opazili da se temperatura nesorazmerno dvigne že po nekaj minutah merjenja. Ker je temperatura najbolj narasla po temu, ko smo modul po daljšem mirovanju (nedelovanju) zopet vklopili, menimo, da meritve pokvarita segrevanje prižgane LED diode na modulu GY-91 ter sam modul ESP8266, ki se med delovanjem prav tako opazno segreje. LED diode na modulu GY-91 nam ni uspelo izklopiti, saj ni del programsko krmiljenih senzorjev BMP280 in MPU-9250.

6.2 MQTT yield

V knjižnici `paho_mqtt_c` se nahaja funkcija `mqtt_yield()`, ki obdrži povezavo po protokolu MQTT aktivno za podano trajanje (v milisekundah). Metoda blokira izvajanje za podano število milisekund, po preteku tega časa pa funkcija vrne



Slika 4: Naraščanje izmerjene temperature.

vrednost, ki sporoča, ali je povezava še vedno aktivna. V primeru, da se je v tem času povezava prekinila, je potrebna ponovna povezava na MQTT strežnik, za trenutno povezavo pa je potrebno klicati metodo `mqtt_disconect()`, ki s tem sprosti vtičnico (angl. *socket*), ki pripada tej povezavi.

V prvotni verziji naše kode, smo med klicem metode `Mqtt_client.yield()` in `Mqtt_client.publish()` dodali še dodaten zamik izvajanja s klicem funkcije `vTaskDelay()`. Prav tako smo vmes še izvajali sprejemanje sporočil preko vmesnika NRF. Posledica teh dodatnih zamikov med klicema dveh funkcij pri komunikaciji z MQTT, je bil padec povezave med odjemalcem (ESP8266) in posrednikom (Raspberry Pi). Ker pa padca povezave nismo primerno zaznali in ostanke povezave pravilno počistili (s klicem metode `disconnect()`), smo po vsakem padcu povezave ustvarili novo. To je privedlo do tega, da smo porabili vse proste vtičnice, s čimer po 12 pošiljanjih (in padcih povezave) ni bilo več mogoče uspešno izvršiti metode `mqtt_connect()`.

Težavo smo rešili z razbitjem opravila na dve ločeni opravili. Prvo je skrbelo za prejemanje sporočil preko NRF, drugo pa za pošiljanje sporočil preko MQTT. V slednji se takoj po vrnitvi iz metode `Mqtt_client.yield()` preveri, ali je v tem času povezava padla (ali je `yield()` vrnil negativen odgovor). Če je povezava ostala živa, sledi pošiljanje vseh ne poslanih sporočil v vrsti, čemur zopet sledi klic metode `Mqtt_client.yield()`. V kolikor pa je prišlo do padca povezave, se kliče `Mqtt_client.disconnect()` in nadaljuje s ponovno vzpostavitvijo povezave in nato s prej opisanim pošiljanjem sporočil.

6.3 Prejem NaN vrednosti na MQTT strežnik

V primerih ko branje temperature na senzorju BMP ni bilo uspešno, je to vrnilo vrednost NaN. Ta vrednost se je nato tudi poslala na MQTT strežnik, zato smo na njem implementirali rešitev, ki vsa prejeta sporočila sicer shrani, vendar pa v nadaljnjo obdelavo in prikaz pošlje samo smiselna. Tako imamo narejeno tudi zaščito, ki skrbi da v primeru prejetja nenavadnih sporočil, le-ta ne vplivajo na samo delovanje spletnega strežnika.

7 Nadaljnji razvoj

Obstaja kar nekaj možnosti za nadaljnji razvoj in nadgradnjo naše rešitve. Poleg temperature bi lahko pošiljali tudi ostale podatke, ki lahko zajemajo senzorji na naši ploščici. Te podatke bi lahko hranili in obdelovali na Raspberry Pi.

Prav tako bi lahko razširili mrežo povezanih naprav, ki bi komunicirale preko vmesnika NRF. Kot do sedaj bi bila samo zadnja vrsta naprav priključena na omrežje in pošiljala podatke preko protokola MQTT.

Optimizirati bi se dalo tudi naslove naprav. Trenutno se namreč v bazo shrani MAC naslov naprave, za katerega vemo, da je unikaten, lahko pa bi optimizirali naslavljanje in s tem prihranili pri poslanih in prejetih podatkih.

Prav tako pa se bi se lahko nadgradilo tudi spletni strežnik, ki bi ponujal več podatkov o temperaturi, dodatne grafe in podobno. Aplikacijo bi lahko tako nadgradili, da bi jo povezali na termostat, s katerim bi potem lahko povratno uravnavali temperaturo (in ostale parametre) v prostoru.

8 Zaključek

Uspešno nam je uspelo narediti končni produkt interneta stvari, ki prebira temperaturo in jo nato pošilja na drugo napravo. Pošiljanje lahko izvaja ali preko vmesnika NRF, ali pa neposredno preko povezave WiFi, s čimer se poveča doseg delovanja. Za pošiljanje preko interneta uporabljamo protokol MQTT. Na strežniku, kjer teče MQTT odjemalec, samo postavili tudi strežnik, ki podatke shranjuje v bazo, ter HTTP strežnik, ki omogoča pregled prejetih podatkov.

Uporabnik lahko pregleduje rezultate meritev in si izrisuje graf, s čimer opazuje nihanje temperature na lokacijah, kjer ima postavljene svoje senzorje.

References

[1] esp-open-sdk. https://github.com/pfalcon/esp-open-sdk/, 2018.

[2] The FreeRTOS Kernel - Market Leading, De-facto Standard and Cross Platform RTOS kernel. https://www.freertos.org/, 2018.

[3] BRUH Automation. How-to get started with mosquitto mqtt broker on a raspberry pi. https://www.youtube.com/watch?v=AsDHEDbyLfg, 2016.

[4] Eclipse. Mosquitto. https://mosquitto.org/, 2018.

[5] Jesenšek J. Lavrinec M. ESP-8266 temperature monitor. https://github.com/jurejesensek/ESP-8266-temperature-monitor, 2018.

[6] Nghia TH. IoT MQTT Dashboard. https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard, 2018.