Středoškolská technika 2016

Setkání a prezentace prací středoškolských studentů na ČVUT

Pathfinder

Matouš Hýbl

Klvaňovo gymnázium a Střední zdravotnická škola Kyjov, příspěvková organizace
třída Komenského 549, Kyjov

Kyjov 2016

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval(a) samostatně a použil (a) jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ. Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné. Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Kyjově dne ..... podpis .....

Poděkování

Chtěl bych především poděkovat své rodině a kolegům z Brightify, kteří mi pomáhali řešit všemožné problémy, ať už softwareové nebo hardwareové. Dále bych chtěl poděkovat komunitě, která se vytvořila kolem WiFi modulu ESP8266, protože vytvořila nástroje, bez kterých by tato práce nevznikla.

Anotace

Tato práce popisuje konstrukci a programování jednoduchých IoT zařízení, založených na programovatelném WiFi modulu ESP8266, které spolu komunikují pomocí protokolu MQTT.

Klíčová slova: IoT, WiFi, ESP8266, MQTT

Annotation

This paper describes construction and programming of some simple IoT devices, that are based on a programmable WiFi module ESP8266. These modules communicate using MQTT protocol.

Keywords: IoT, WiFi, ESP8266, MQTT

Obsah

Úvod

Internet věcí (IoT) je jedno z nejdiskutovanějších technologických témat současnosti, většina technologických společností investuje velké množství prostředků na výzkum a vývoj IoT. Internet věcí je založen na jednoduché myšlence, že pokud by bylo možné připojit cokoliv k internetu, bylo by možné, aby spolu tato zařízení komunikovala a společně by přispívala k řešení nějakého problému. Nejznámější případ použití je domácí automatizace - například pokud by v každém pokoji byly senzory teploty připojené k internetu a vytápěcí systém rovněž připojený k internetu, bylo by možné, aby tento vytápěcí systém řídil podle dat ze senzorů vytápění v celém domě. Další použití by bylo například v optimalizaci využití elektrické energie v domácnosti – díky IoT by se například elektromobil mohl nabíjet podle toho, kdy je nejlevnější energie, tedy hlavně v noci. Internet věcí přináší spoustu nových příležitostí jak učinit náš život jednodušší a pohodlnější.

Nápady na takovou domácí automatizaci se objevovaly už s prvními počítači, vývoj takových zařízení ale umožnil až technický pokrok v několika posledních letech, a to hlavně díky pokroku v oblasti bezdrátového přenosu signálu a nízkospotřebových procesorů. Například ARM procesory mají nyní prakticky stejný výkon jako x86 procesory, avšak se zlomkovou spotřebou. Trvalé připojení k internetu také již není problém, většina domácností je připojena přípojkami s vysokou rychlostí, popřípadě je připojena bezdrátově pomocí mobilních sítí, které jsou díky LTE mnohem rychlejší než většina klasických přípojek. V místech, kde dosud žádné připojení nebylo zejména kvůli náročnosti vybudování vysílačů, se začíná budovat připojení k internetu pomocí horkovzdušných balónů – Project Loon od Google, který například pokryje internetem většinu ostrovů Indonésie, které by byly pomocí standardních postupů zcela nepokrytelné.

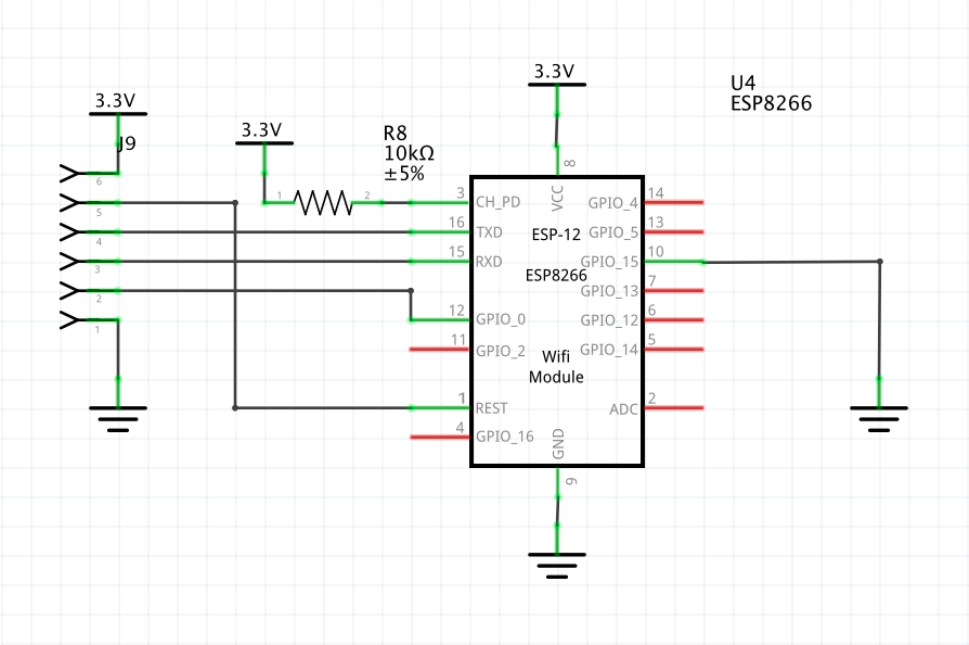
Na druhou stranu přináší Internet věcí mnoho výzev z oblasti bezpečnosti. Zařízení připojená k internetu musí být relativně odolná vůči vnějším útokům, protože sebemenší únik informací by mohl například lupičům prozradit, kdy majitel domu přichází domů, a ti by mohli vyloupení mnohem pečlivěji připravit. IoT by se ale také mohlo stát důležitým prvkem bezpečnosti – takový dům by mohl vyhodnotit, že majitel se obvykle domů v tuto hodinu nevrací a poslat varovnou zprávu, díky které by bylo možné lupiče chytit přímo při činu.

1 Hardware

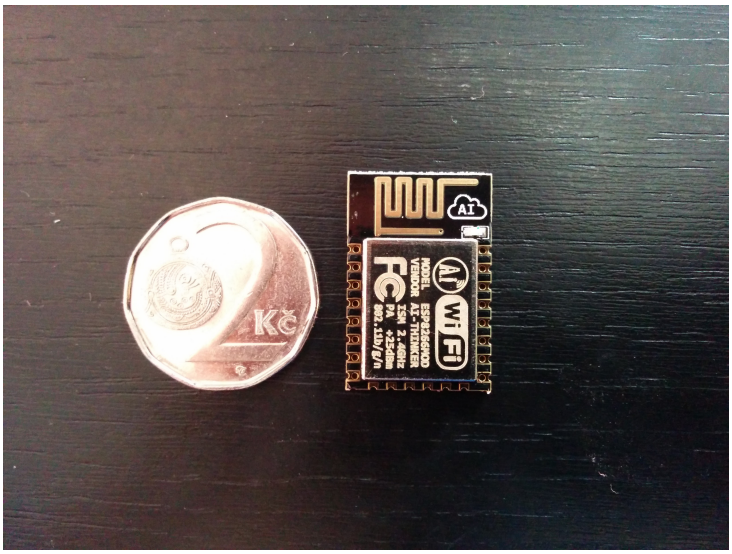
IoT zařízení musí být připojitelné k internetu, čehož je možné dosáhnout pomocí kabelového připojení, pomocí WiFi, nebo pomocí signálu zakódovaného v elektrické síti. V této práci bude rozebrána možnost připojení zařízení pomocí WiFi, protože kabelové řešení není při větším počtu zařízení ideální vzhledem k množství kabeláže, která by musela být rozvedena po celém domě, a protože internet zakódovaný v elektrické síti podléhá rušení a nedovoluje velké datové toky. Všechna tři popisovaná zařízení budou využívat WiFi modulu ESP8266. Tento modul způsobil malou revoluci v hobby internetu věcí, protože je dostatečně levný a malý na to, aby se dal v takových zařízeních použít.

1.1 ESP8266

ESP8266 je WiFi modul vyráběný čínskou firmou Espressif, vyrábí se ve třinácti variantách rozlišených podle toho, jakým způsobem a kolik pinů je vyvedeno z pouzdra ven. Jeho nejvybavenější varianta je ESP-12E, které má přibližně velikost palce. Modul je založen na 32-bitovém procesoru, podporuje WiFi standardy 802.11 b/g/n, lze jej použít jako WiFi AP i jako klient. V základu je vybaven firmwarem ovládaným pomocí AT commands, určeným k připojení k dalším zařízením pomocí sériové linky. ESP8266 má napájecí napětí 3,3 V a jeho logika je rovněž 3,3-voltová. Pro jeho použití je nutné jej připojit k napájení a přivést napětí na pin CH_PD. Modul se programuje pomocí UARTu, k jeho programování tedy stačí jakýkoliv obyčejný USB<->UART převodník, který podporuje 3,3V úrovně, pro programování je potřeba navíc připojit RST pin k DTR výstupu převodníku a uzemnit GPIO0.



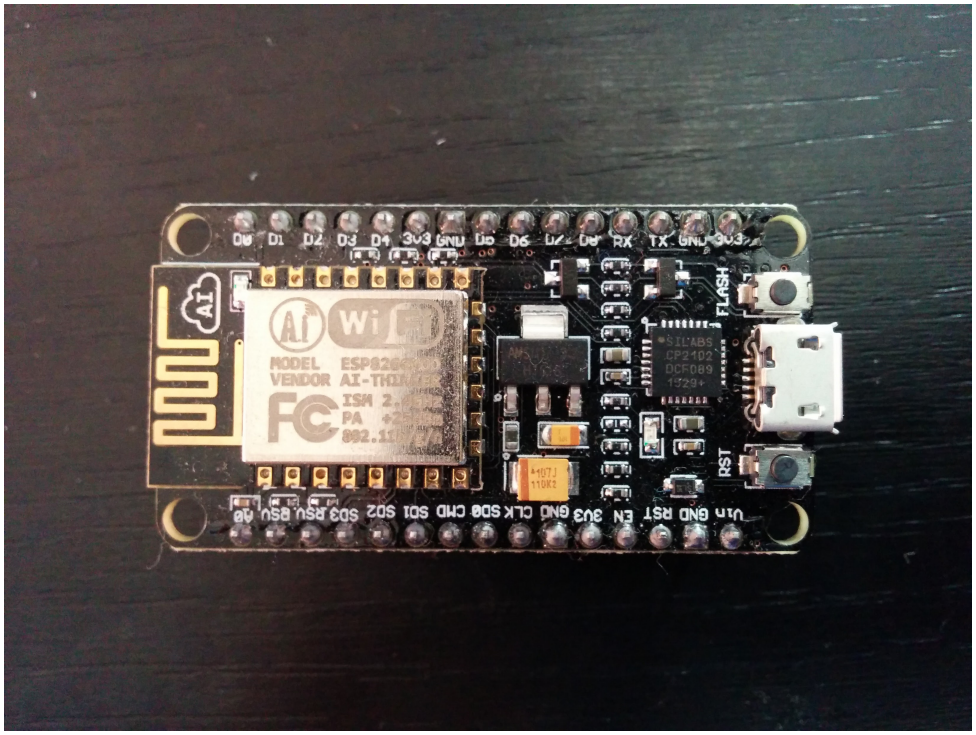
Základní zapojení s programovacím konektorem



Miniaturní modul ESP12E

1.2 NodeMCU Devkit

Vzhledem k tomu, že většina ESP modulů je vyráběna pouze v SMD variantě, která má navíc metrickou rozteč pinů 2 mm místo standardizované imperiální rozteče 2,53 milimetrů, vzniklo několik redukcí a devkitů pro ESP8266. Jedním z nejpokročilejších devkitů je NodeMCU Devkit, který je určený k použití v nepájivých kontaktních polích a má v sobě integrovaný převodník USB <-> UART, stabilizátor napětí, který stabilizuje napětí z 5 V USB na 3.3 V, které vyžaduje ESP. Pro programování ESP tedy stačí připojit Devkit do USB. Při vývoji projektu Pathfinder byl tento modul použit pouze pro prototypování.



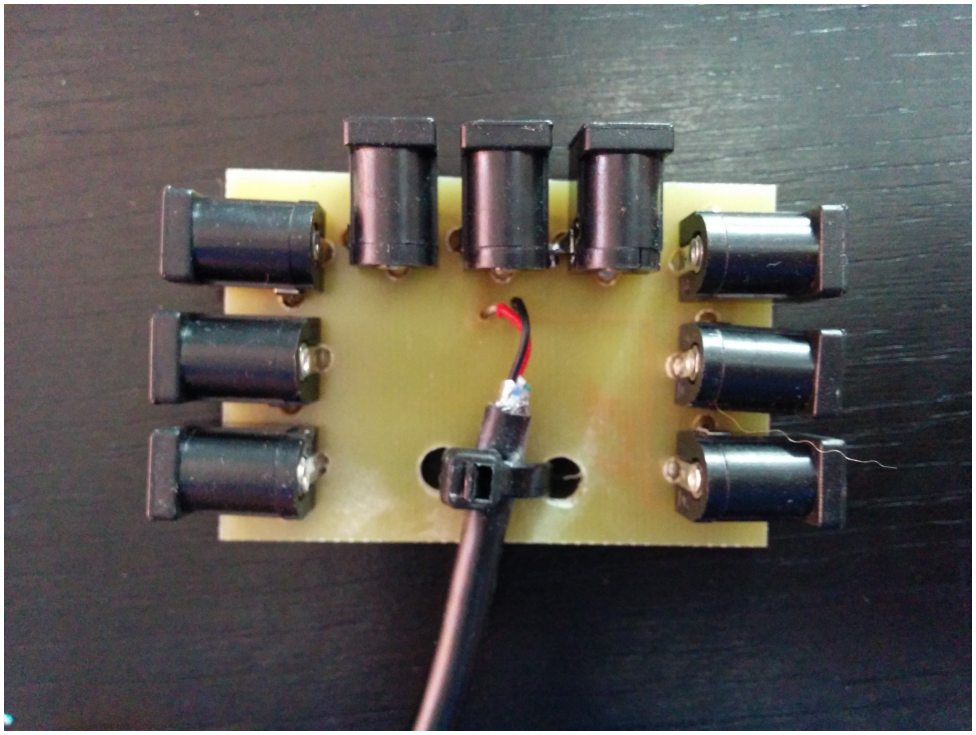
NodeMCU Devkit

1.3 Napájení

Jedním z největších problémů IoT je to, jakým způsobem zajistit co nejefektivnější napájení velkého množství zařízení. Jedním z možných řešení by byla centralizace modulů do jednoho velkého zařízení, což ale není v mnoha případech možné. Nevýhodou tohoto řešení by bylo zejména to, že by poskytovalo data z pouze jednoho místa například části nějaké místnosti. Dalším možným řešením by bylo mít v každém zařízení transformátor, což by bylo neefektivní, drahé a mohlo by to způsobovat rušení. Nehledě na to, že při vyšším množství zařízení by bylo nutné použít mnoha prodlužovaček, což je značně nepraktické. Tento problém byl vyřešen pomocí tzv. Hubu.

1.3.1 Pathfinder Hub

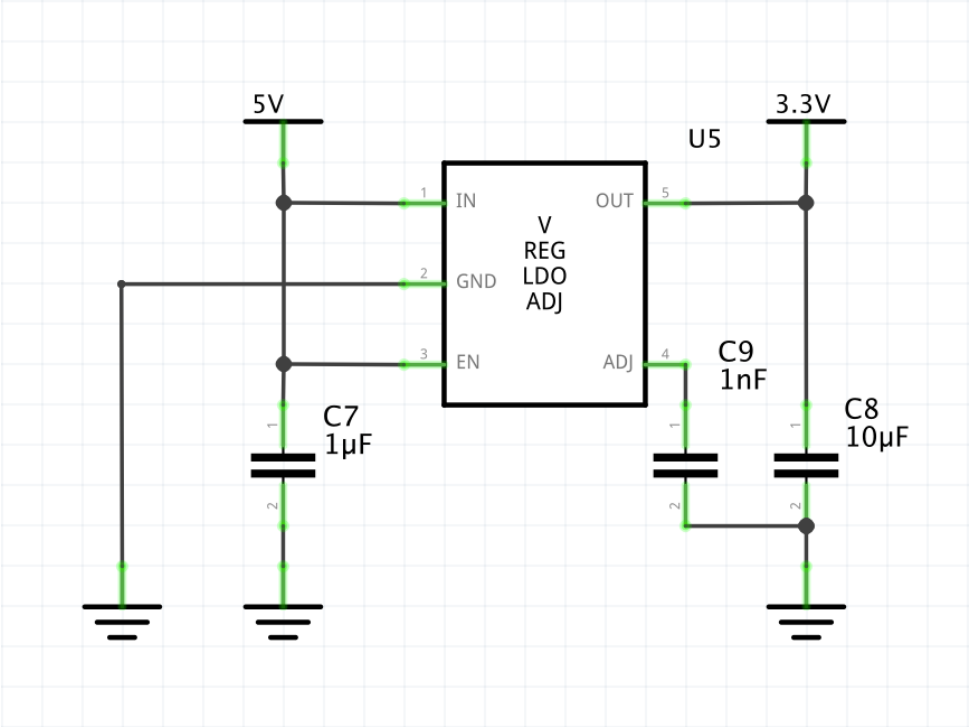
Pathfinder Hub je centralizované napájení pro IoT zařízení, jedná se o rozbočovač, který je pomocí USB připojený k napětí 5 V, díky použití USB je možné napájet Hub z jakékoliv nabíječky na současné mobilní telefony. Zařízení jsou k Hubu připojena pomocí 2,1 mm napájecích konektorů. V budoucnu by měl vzniknout Hub s integrovaným transformátorem a také Hub, který bude integrovaný v zásuvce. Možným vylepšením Hubu by bylo vybavit jej ESP8266 a senzorem magnetického pole, takže by bylo možné, aby Hub monitoroval svou spotřebu.



Pathfinder Hub s USB kabelem

1.3.2 Stabilizace napětí v zařízeních

Pomocí Hubu jsou zařízení napájena pomocí 5 V, ale moduly ESP vyžadují napájecí napětí nižší a to 3,3 V, na každé desce je tedy stabilizátor. Obvod stabilizátoru byl použit stejný, jako se používá ve vývojových deskách NodeMCU, a to obvod založený na integrovaném obvodu SPX3819M5-L-3-3, který je doplněný pouze několika málo kondenzátory.

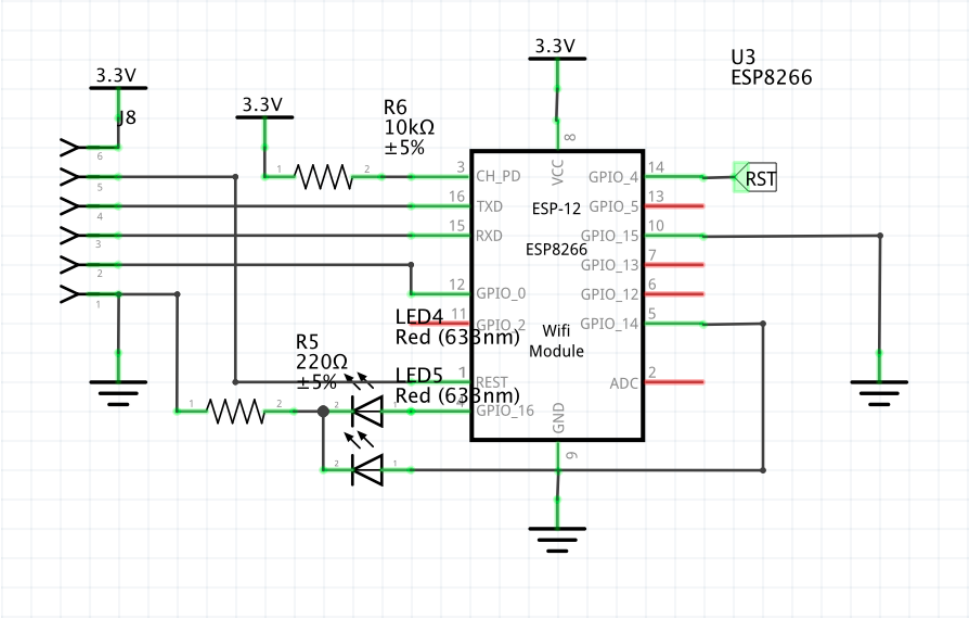


Zapojení stabilizátoru napájení

1.4 Pathfinder zařízení

1.4.1 Minimální zapojení Pathfinder zařízení

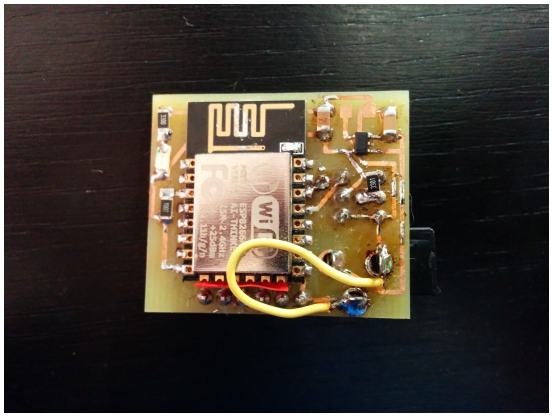
Každé zařízení musí z důvodu alespoň minimální uživatelské přívětivosti obsahovat resetovací tlačítko a dvě LED, určené pro indikování stavu zařízení. Červená LED indikuje stav, kdy je zařízení odpojeno od WiFi, zatímco zelená LED indikuje stav, kdy je zařízení připraveno k použití.



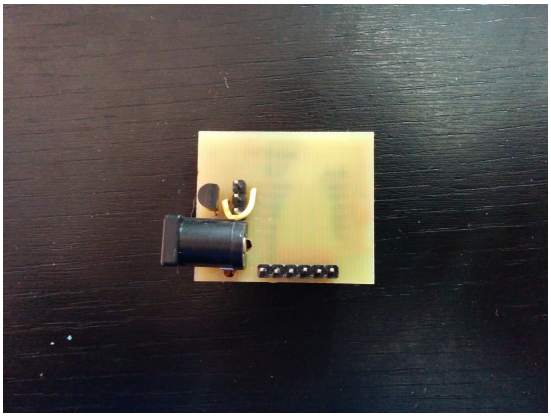
Obrázek 1.1: Minimální zapojení Pathfinder zařízení

1.4.2 Pathfinder Temp

Pathfinder Temp je nejjednodušší IoT zařízení vyvinuté pro projekt Pathfinder, jedná se o destičku plošných spojů obsahující pouze ESP-12E, ke kterému je připojen One-Wire teploměr DS18b20. Deska dále obsahuje stabilizátor napětí a konektor pro připojení k Hubu. DPS má velikost 30 x 35 mm a je umístěna v krabičce vytištěné na 3D tiskárně.



Spodní pohled

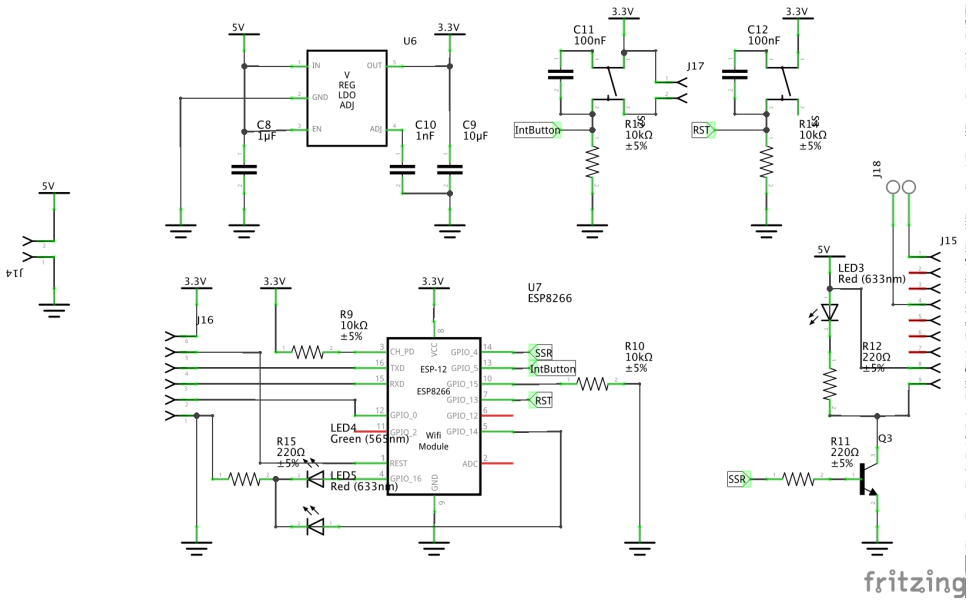


Horní pohled

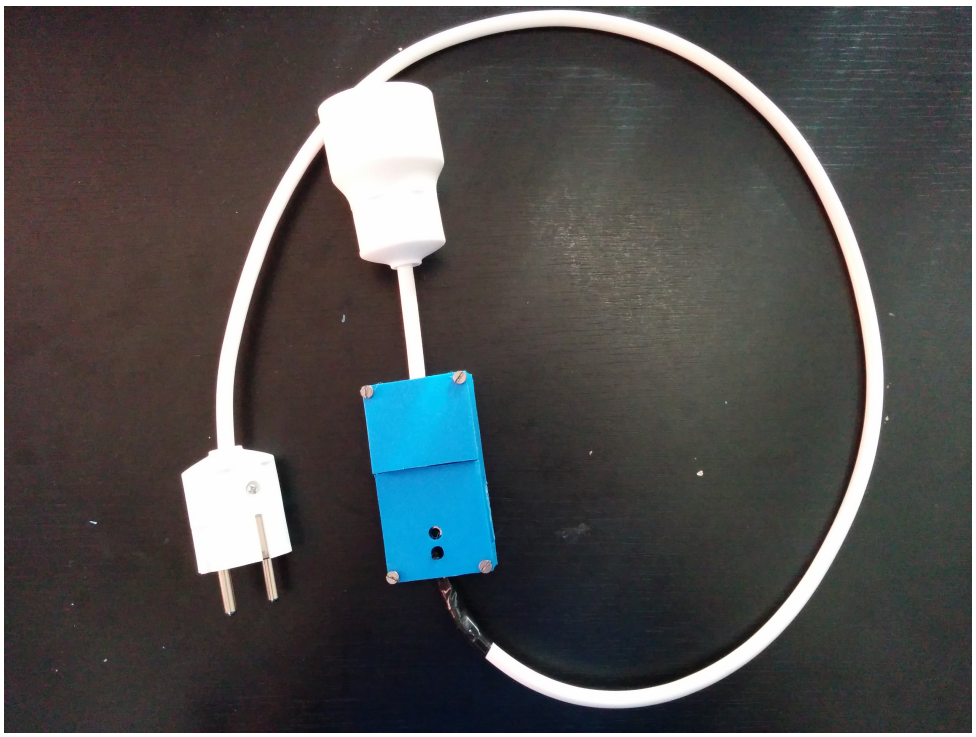
1.4.3 Pathfinder Plug

Pathfinder Plug je IoT zařízení určené ke spínání nízkopříkonových elektrospotřebičů do 460 W. Jedná se o modul ESP12E doplněný o stabilizátor napětí, SSR relé, tlačítko pro manuální ovládání. Vzhledem k praktičnosti není tento modul napájen pomocí Hubu, ale má vlastní zdroj stejnosměrného napětí 5 V.

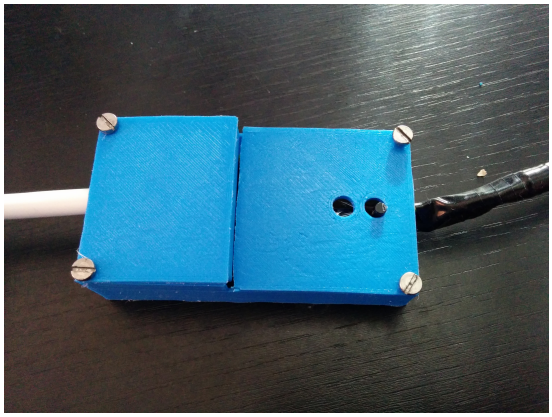
Plug lze rozšířit o externí tlačítko, díky kterému je možné jednoduše manuálně ovládat například stolní lampu, která bude přes plug připojen do elektrické sítě.



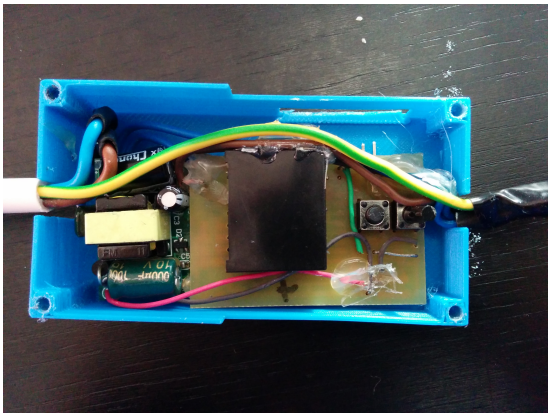
Zapojení Plugu



Plug i se zásuvkami



Detail krabičky Plugu

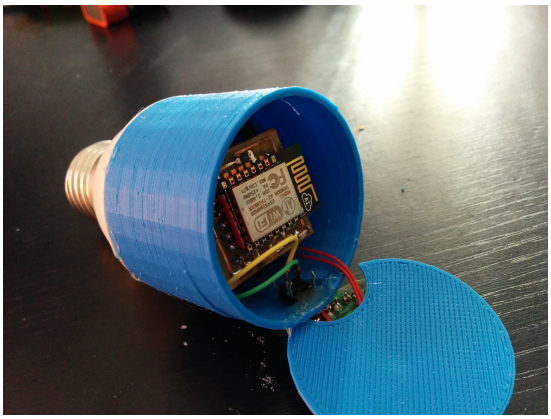


Elektronika Plugu

1.4.4 Pathfinder Bulb

Pathfinder Bulb je dálkově ovládaná LED žárovka, která je založena na levné čínské stavebnici 2,7 W LED žárovky s 38 standardními LED diodami. Struktura Bulb je velmi podobná struktuře Plugu, obsahuje transformátor dodávající stejnosměrné napětí 5 V, tlačítko pro manuální ovládání a SSR relé. Vzhledem k tomu, že řídicí elektronika byla relativně velká a nevlezla by se do původní stavebnice, bylo nutné žárovku rozšířit pomocí prstence vytištěného pomocí 3D tiskárny. Jednotlivé plošné spoje jsou od sebe oddělené pomocí kruhových oddělovačů s otvorem pro kabely rovněž vytištěných na 3D tiskárně. Bohužel před dokončením této práce byl vyroben pouze prototyp založený na starém návrhu plošného spoje, který ale nefungoval tak, aby jej bylo možné používat.

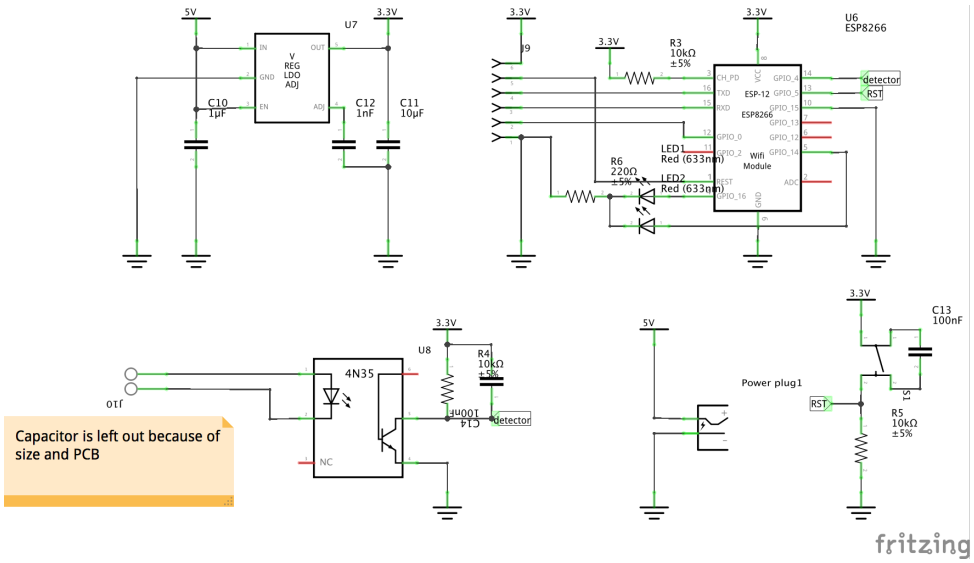
Bulb



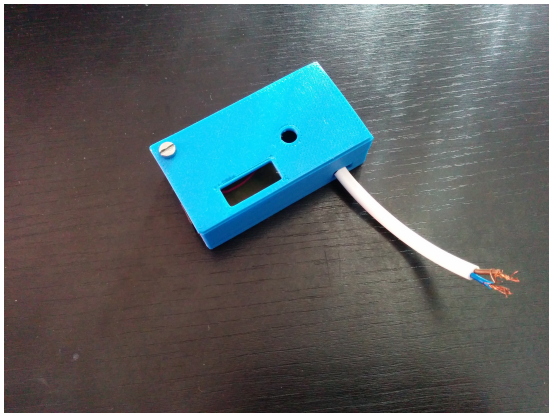
Elektronika Bulbu

1.4.5 Pathfinder Bell

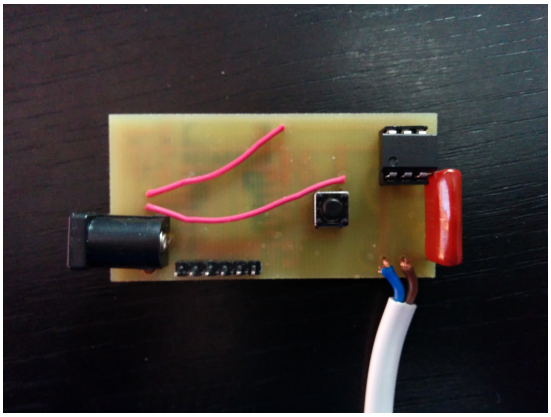
Pathfinder Bell je další senzor založený na modulu ESP8266, byl navržen jako senzor detekce toho, kdy domovním zvonkem prochází elektrický proud, ale lze jej použít univerzálně jako senzor střídavého elektrického napětí. Je založen na optočlenu LDA100, který má integrovány dvě antiparalelní diody, lze jej tedy bez větších problémů použít pro detekci střídavého napětí. Jediným problémem je proud protékající diodami, protože je limitován na 50 mA. Pokud by byl použit rezistor, musel by mít výkon $P = U \times I = 230\ V \times 0,05\ A = 11,5\ W$, což u tak miniaturního zařízení nepřipadá v úvahu, nehledě na vyprodukované teplo. Proto bylo nutné použít kondenzátor, který se v obvodu střídavého proudu chová díky kapacitanci jako rezistor. Pro kapacitu kondenzátoru v takovém obvodu platí vzorec $C = \frac{I}{2\pi f U}$, po dosazení je výsledná kapacita kondenzátoru rovna 553 nF, protože kondenzátory o takové hodnotě se nevyrábějí, byl v zapojení použit kondenzátor o kapacitě 470 nF a maximálním napětí 400 V.



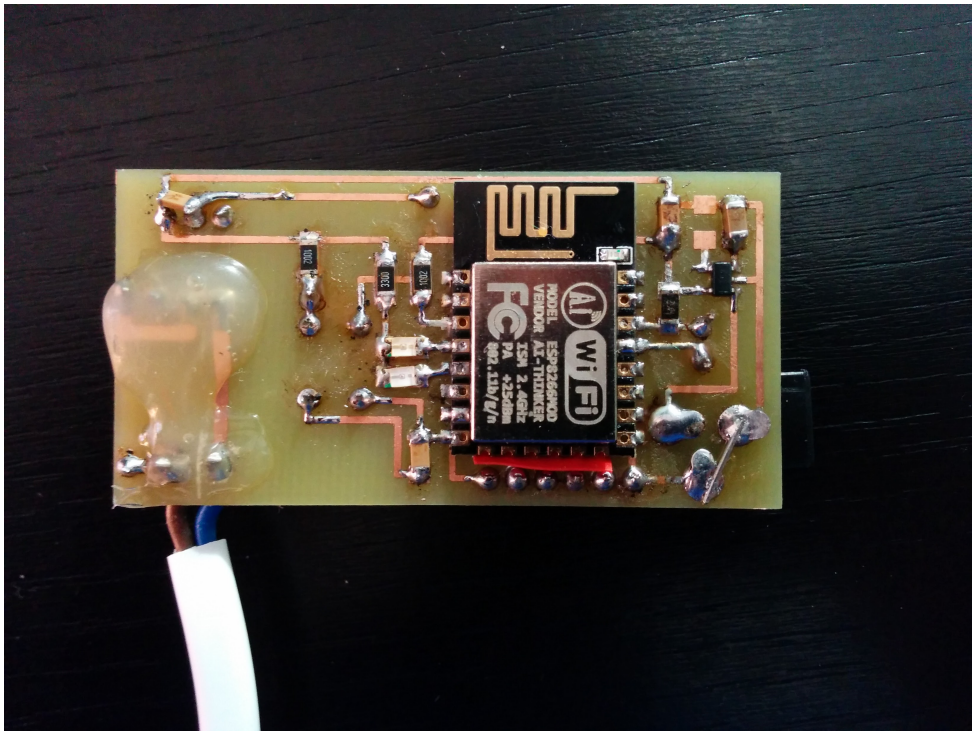
Zapojení Bellu



Krabička Bellu



Horní pohled



Plošný spoj Bellu

2 Software

Platforma Pathfinder je založena na protokolu MQTT, který obstarává komunikaci jednotlivých zařízení mezi sebou i mezi ovládacími aplikacemi. Pro použití tohoto protokolu bylo nutné naprogramovat klienty jak pro koncová zařízení jako je Plug, Bulb a Temp, tak i klient pro platformu Android™ Sojourner, který zahrnuje aplikaci pro mobilní telefon.

2.1 MQTT

MQTT je komunikační protokol vyvinutý v roce 1999 Andym Stanford-Clarkem (IBM) a Arlenem Nipperem (Cirrus Link Solutions). Tento protokol byl navržen k monitorování tlaku ropovodu. V té době bylo nutné optimalizovat množství přenášených dat, proto byl protokol navržen tak aby množství přenášených dat bylo co nejmenší. Protokol byl také navržen tak, aby byl nenáročný na systémové prostředky. Protokol je založen na návrhovém vzoru publisher/subscriber, což znamená, že publisher odesílá data s určitým tématem (topic). Jakémukoliv subscriberovi, který přijímá data s tímto tématem, budou data doručena. Velkou výhodou MQTT je to, že publisher a subscriber jsou na sobě nezávislé. Mezi jakýmkoliv publisherem a subscriberem je MQTT Broker, což je software, který má na starost distribuci dat.

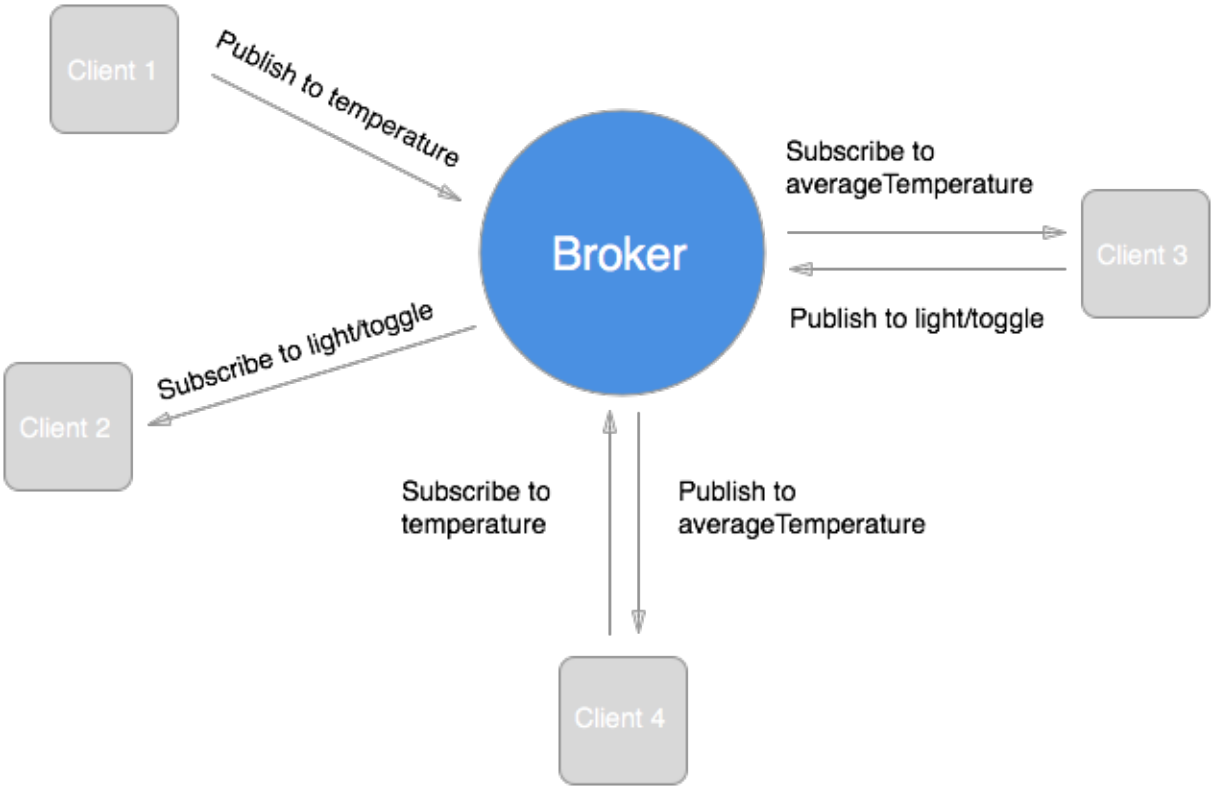


Schéma fungování MQTT

2.1.1 MQTT Broker

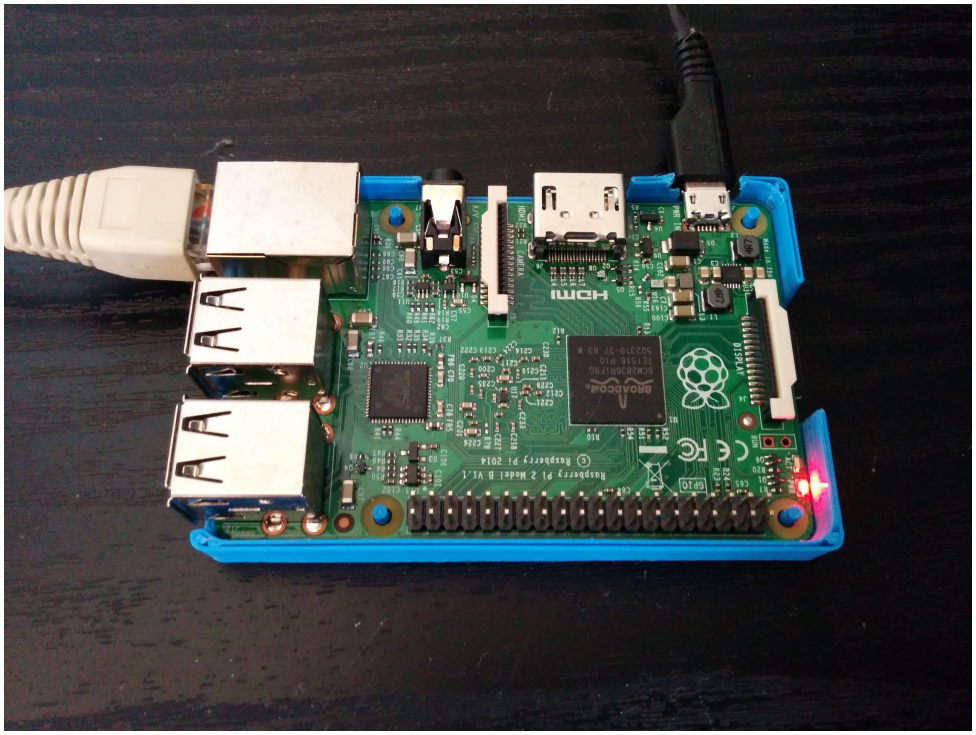
Platforma Pathfinder používá jako MQTT Broker Mosquitto, které je nainstalované na RaspberryPi 2, je ovšem možné použít i jiné brokery, například Mosca – broker pro platformu Node.js nebo HiveMQ.

Instalace

Repozitáře Raspbianu neobsahují nejaktuálnější verzi Mosquitto, která je potřebná pro komunikaci s Pathfinder zařízeními, proto je nutné instalovat Mosquitto z oficiálního Mosquitto repozitáře.

Konfigurace

Konfigurace Mosquitto probíhá úpravou souboru **/etc/mosquitto/mosquitto.conf**. Pokud je MQTT port otevřený na routeru, je nutné MQTT zabezpečit a to minimálně pomocí uživatelského jména a hesla, aby případný útočník neměl jednoduchý přístup k brokeru a nemohl pomocí něj ovládat zařízení.



Obrázek 2.1: Raspberry Pi 2 s běžícím brokerem

2.1.2 Preprocesory

MQTT klienty nemusí být pouze reálná zařízení, ale mohou to být také obyčejné programy, které nějak zpracovávají data, která broker přijme. Jednoduchým příkladem je klient, který bude přijímat zprávy na téma (topic) **home/temperature**, přijatá data bude průměrovat a průměrnou teplotu bude publikovat na téma **home/averageTemperature**. Při použití preprocesorů je ale nutné, aby se předešlo problémům, používat unikátní identifikátor klienta – u IoT zařízení stačí MAC adresa, zatímco na preprocesoru ne, protože jich může na jednom zařízení být spuštěných několik a MAC adresa už by nebyla unikátní.

2.1.3 MQTT Wildcards

Client nemusí subscribovat na jediné téma, ale může jednoduše subscribovat na celý strom témat, k čemuž slouží MQTT Wildcards. Existuje single level wildcard “+”, díky kterému, pokud klient subscribne například **home/+/temperature**, bude dostávat zprávy

na témata například **home/bedroom/temperature**, **home/kitchen/temperature**. Další wildcard je “#”, což je multi level wildcard, což znamená, že pokud klient subscribne **home/#**, bude dostávat zprávy z celého stromu témat spadajích pod téma home. Doporučuje se nikdy neprovádět subscribe na celý strom témat typu ”#”, protože by mohlo dojít k zahlcení zařízení.

2.1.4 Retained Messages

Další důležitou funkcionalitou MQTT jsou retained messages. Jedná se o zprávy, publikované na určitý topic, které jsou doručeny všem klientům okamžitě, kdy na tento topic subscribnou. Retained zpráva je zrušena v momentě, kdy je na daný topic publikována jakákoliv jiná zpráva. Retained messages jsou vhodné zejména pro publikování stavových zpráv, například při startu zařízení lze odeslat retained message s informacemi o zařízení, třeba IP adresou a podobně, a poté nastavit retained message do Last will and Testament, popsaném dále, která obsahuje informaci o odpojení zařízení.

2.1.5 Last will and Testament

Pro případy neočekávaného odpojení zařízení od brokeru je možné nastavit Last will and Testament (poslední vůli a závěť). Tato zpráva je odeslána na specifikované téma vždy, když dojde k neočekávanému odpojení od brokeru, nebo když zařízení neodpovídá.

2.2 Firmware

Firmware pro IoT zařízení byl naprogramován v programovacím jazyce C++ za použití frameworku Sming, který používá Arduino API pro přístup hardwarovým periferiím, díky čemuž lze používat většinu Arduino knihoven. Sming obsahuje mnoho knihoven – pro různé komunikační protokoly MQTT, HTTP, UDP, senzory (DHT11) atd. Jednotlivé firmwary pro různá Pathfinder zařízení se liší jen v tom, které zprávy jim posílá broker v závislosti na subscribnutých tématech. Firmware pro Pathfinder zařízení musí být implementovány tak, aby v se v případě odpojení od sítě automaticky znova připojily a aby se automaticky připojily na broker, pokud ztratí spojení. Výstupem kompilace C++ projektu je několik **.bin** souborů, které je nutné do flashe ESP pomocí nástroje **esptool**. Celý proces kompilace a flashování obstarává **Makefile** dodávaný s frameworkem.

2.2.1 Instalace Sming

Instalace frameworku Sming se skládá z několika kroků, nejprve je potřeba stáhnout a rozbalit předkompilované ESP SDK, poté je nutné naklonovat Git repozitář z GitHubu projektu. Následně je nutné Sming zkompilovat a také zkompilovat závislost Smingu - souborový systém ‘spiffy‘, poté již lze Sming použít v C++ projektu.

```
curl -L -O https://bintray.com/artifact/download/kireevco/
        generic/esp-alt-sdk-v1.5.0.258-linux-x86_64.tar.gz
sudo tar -zxf esp-alt-sdk-v1.5.0.258-linux-x86_64.tar.gz
        /opt/esp-open-sdk
cd /opt
sudo git clone https://github.com/SmingHub/Sming.git
cd Sming/Sming
sudo make
cd ..
cd spiffy && sudo make
```

2.2.2 Použití frameworku Sming v IDE CLion

Ačkoliv je Sming navržen tak, aby se co nejjednodušeji používal v Eclipse, lze jej velice jednoduše použít v CLion. Bohužel CLion zatím neumí používat Makefile projekty, takže je potřeba upravit ‘CMakeLists.txt‘ tak, aby IDE mělo reference na zdrojové kódy Sming frameworku. Při vytvoření každého projektu je nutné do něj nakopírovat **Makefile** a **Makefile-user.mk**, které jsou dostupné v adresáři Samples na GitHubu frameworku. Dále je nutné vytvořit v projektu adresář **app** a v něm C++ soubor **application.cpp**, který obsahuje metodu **void init() {}**. Dalším důležitým souborem v projektu je **user_config.h** umístěný ve složce **include**. Bez tohoto souboru nebudou fungovat některé operace s řetězci.

CMakeLists.txt by měl obsahovat přibližně toto:

```
cmake_minimum_required(VERSION 3.3)
project(Pathfinder)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
include_directories("/opt/Sming/Sming")
include_directories("/opt/Sming/Sming/Libraries")
include_directories("/opt/Sming/Sming/system/include")
include_directories("/opt/esp-open-sdk/sdk/include")

set(SOURCE_FILES app/application.cpp include/user_config.h)
add_executable(Pathfinder ${SOURCE_FILES})
```

app.cpp:

```
#include <user_config.h>
#include <SmingCore/SmingCore.h>

void init() {
}
```

Kompilace projektu a jeho flash do ESP se pak provádí v integrovaném terminálu pomocí příkazů:

```
# kompilace
make clean && make
# flash
make flash
# kompilace a flash
make clean && make && make flash
```

2.2.3 Životní cyklus

Po startu zařízení dojde ke konfiguraci hardware, připojení k WiFi, popřípadě konfiguraci pomocí funkce Smart Config. Následně dojde k subscribe MQTT témat a registrace MQTT callbacků. Po úspěšné inicializaci zhasne červená LED a rozsvítí se zelená.

2.2.4 Připojení k internetu

Připojení k internetu probíhá dvěma způsoby v závislosti na tom, jestli se jedná o první spuštění nebo ne. Při prvním spuštění se využije feature SmartConfig, díky které je možné nakonfigurovat připojení k internetu pomocí mobilního telefonu. Funguje to tak, že telefon zakóduje SSID a heslo do velikosti UDP paketů, které pak rozešle, ESP je ve SmartConfig módu a odposlouchává UPD packety, ve kterých hledá podobnosti, pokud je najde, dekóduje SSID a heslo a použije je k připojení. Při každém dalším spuštění se zařízení připojí pomocí těchto přihlašovacích údajů. SmartConfig lze vynutit stiskem tlačítka resetu, které smaže stávající přihlašovací údaje a restartuje zařízení. Pro zjednodušení kódu byla funkcionalita připojení k WiFi abstrahována do třídy WiFiUtils.

WiFiUtils.h

```
class WifiUtils {
public:
    WifiUtils();
    void connect(void (*onSuccess)());
    static void resetConfiguration();

private:
    void smartConfigCallback(sc_status, void *pdata);
    void onFailure();
    void (*onSuccess)();
};
```

WiFiUtils.cpp

```
WifiUtils::WifiUtils() {
    WifiAccessPoint.enable(false);
    WifiStation.enable(true);
}

void WifiUtils::connect(void (*onSuccess)()) {
    this->onSuccess = onSuccess;

    if(WifiStation.getSSID().length() == 0) {
        WifiStation.smartConfigStart(SCT_EspTouch,
            Delegate<void(sc_status status, void *pdata)>
                (&WifiUtils::smartConfigCallback, this));
    } else {
        WifiStation.waitConnection(onSuccess, 30,
            Delegate<void()>(&WifiUtils::onFailure, this));
    }
}

void WifiUtils::resetConfiguration() {
    WifiStation.config("", "", true);
}

void WifiUtils::smartConfigCallback(sc_status status, void *pdata) {
```

```
    if(status == SC_STATUS_LINK) {
        station_config *sta_conf = (station_config *) pdata;
        char *ssid = (char *) sta_conf->ssid;
        char *password = (char *) sta_conf->password;
        WifiStation.config(ssid, password, true);
    }
    if(status == SC_STATUS_LINK_OVER) {
        WifiStation.smartConfigStop();
        onSuccess();
    }
}

void WifiUtils::onFailure() {
    WifiStation.waitConnection(onSuccess, 30,
        Delegate<void()>(&WifiUtils::onFailure, this));
}
```

2.2.5 Programování zařízení

Za účelem zjednodušení kódu a zlepšení jeho čitelnosti byla vytvořena abstraktní třída **IDevice**, kterou implementují všechna zařízení. **IDevice** obsahuje všechny metody související s životním cyklem aplikace. Jednotlivé implementace jsou pak použity pomocí makroprocesoru C++ v hlavním programu.

IDevice.h

```
class IDevice {
public:
    IDevice(MqttClient* client);
    virtual void subscribe() = 0;
    virtual void onMessage(String topic, String message) = 0;

protected:
    MqttClient* mqttClient;
};
```

Metoda **subscribe()** je zavolána v momentě, kdy je úspěšně zařízení připojeno k MQTT brokeru a měla by zaregistrovat všechna témata, na která bude zařízení reagovat. Metoda **onMessage()** je volána vždy, když MQTTClient přijme zprávu, v této metodě by měly být všechny zprávy zpracovány.

Pathfinder Plug

Implementace **IDevice** pro plug subscribuje na témata **plug/toggle**, **plug/toggle/on** a **plug/toggle/off**, zpracovává tyto zprávy tak, že přepne výstupní pin v závislosti na zprávě. V konstruktoru třídy jsou nakonfigurovány vstupní a výstupní piny a jsou zaregistrována přerušení.

```
Plug::Plug(MqttClient *client) : IDevice(client) {
    lastTrigMillis = millis();
    pinMode(PLUG_POWER_PIN, OUTPUT);
    pinMode(PLUG_TRIG_PIN, INPUT);
    digitalWrite(PLUG_POWER_PIN, LOW);
    attachInterrupt(PLUG_TRIG_PIN,
        Delegate<void()>(&Plug::onManualToggle, this), FALLING);
}

void Plug::subscribe() {
    mqttClient->subscribe("plug/toggle");
    mqttClient->subscribe("plug/toggle/on");
    mqttClient->subscribe("plug/toggle/off");
}

void Plug::onMessage(String topic, String message) {
    if(topic == "plug/toggle") {
        togglePower();
    } else if(topic == "plug/toggle/on") {
        powerOn();
    } else if(topic == "plug/toggle/off") {
        powerOff();
    }
}

void Plug::togglePower() {
    digitalWrite(PLUG_POWER_PIN,
        (uint8_t) (digitalRead(PLUG_POWER_PIN) == HIGH ? LOW : HIGH));
}
```

```
void Plug::powerOn() {
    digitalWrite(PLUG_POWER_PIN, HIGH);
}

void Plug::powerOff() {
    digitalWrite(PLUG_POWER_PIN, LOW);
}

void Plug::onManualToggle() {
    long now = millis();
    if(now - lastTrigMillis > 500) {
        togglePower();
        lastTrigMillis = now;
    }
}
```

Pathfinder Bulb

Implementace třídy **IDevice** vypadá obdobně jako implementace pro Plug, liší se pouze čísla pinů.

Pathfinder Temp

Implementace pro Temp, na rozdíl od dvou předchozích, nesubscribuje na žádná témata, zato je v ní nakonfigurován Timer, který každých 10 sekund pošle zprávu s teplotou na téma **temp/temperature**. Pro obsluhu senzoru DS18b20 byla použita knihovna z frameworku Sming **DS1820.h**.

```
Temp::Temp(MqttClient *client) : IDevice(client) {
    sensor.Init(5);
    sensor.StartMeasure();
    sensorTimer.initializeMs(10 * 1000,
        Delegate<void()>(&Temp::onMeasure, this)).start(true);
}

void Temp::subscribe() {}

void Temp::onMessage(String topic, String message) {}

void Temp::onMeasure() {
    if(sensor.MeasureStatus()) {
        return;
    }
    if(sensor.GetSensorsCount() > 0 && sensor.IsValidTemperature(0)) {
        mqttClient->publish("temp/temperature", String(sensor.GetCelsius(0)));
    }
    sensor.StartMeasure();
}
```

Pathfinder Bell

Tato implementace rovněž neprovádí žádný subscribe, pouze nastavuje přerušení na pin, ke kterému je připojen výstup optočlenu. Metoda přerušení je nakonfigurována na **CHANGE**, což znamená, že by se měla volat jak při náběžné hraně signálu, tak při sestupné hraně, bohužel toto z neznámého důvodu nefunguje a metoda se volá pouze při sestupné hraně, tj. při detekci střídavého napětí.

```
Bell::Bell(MqttClient *client): IDevice(client) {
    lastBellRing = millis();

    attachInterrupt(BELL_SENSOR_PIN,
        Delegate<void()>(&Bell::onBellRing, this), CHANGE);
}

void Bell::subscribe() {}

void Bell::onMessage(String topic, String message) {}

void Bell::onBellRing() {
    long now = millis();
    if(now - lastBellRing > 200) {
        mqttClient->publish("bell/ring",
            digitalRead(BELL_SENSOR_PIN) == LOW ? "ringing" : "not ringing");
        lastBellRing = now;
    }
}
```

2.3 Android klient Sojourner

Pro jednoduché ovládání zařízení byl naprogramován android klient Sojourner. Tento klient je založen na open source platformě pro MQTT vyvíjenou Eclipse Foundation a to Eclipse Paho. Aplikace byla naprogramována v Javě s pomocí frameworku Android Annotations. Design byl vytvořen podle Material Design specifikace. Celá aplikace se skládá pouze ze tří aktivit - jedné, která zobrazuje karty reprezentující jednotlivá zařízení, aktivity pro přidávání zařízení a konfigurační aktivity. Pro ukládání zařízení byla použita ORM knihovna Torch vyvíjená společností Brightify.

MQTT klient

Pro zjednodušení práce s Paho byla vytvořena třída, která odchytává veškeré výjimky, které jsou vyhazovány většinou metod, které jsou na Paho klientu volány.

```
public class MqttClient {

    private MqttAndroidClient mqttAndroidClient;
    private HashMap<String, OnMessageArrivedListener> subscriptions
        = new HashMap<>();

    public MqttClient(Context context, String brokerURL) {
        mqttAndroidClient = new MqttAndroidClient(context, brokerURL,
                Settings.Secure.getString(context.getContentResolver(),
                Settings.Secure.ANDROID_ID));
        mqttAndroidClient.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable throwable) { }
            @Override
            public void messageArrived(String s, MqttMessage mqttMessage)
                throws Exception {
                for (String topic : subscriptions.keySet()) {
                    if(topic.equals(s)) {
                        subscriptions.get(topic).onMessageArrived(topic,
                            new String(mqttMessage.getPayload()));
                    }
```

```
                }
            }
            @Override
            public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken)
            {}
        });
    }

    public void connect(IMqttActionListener listener) {
        try {
            mqttAndroidClient.connect(null, listener);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void disconnect() {
        try {
            mqttAndroidClient.disconnect();
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void publish(String topic, String message, int QoS,
        boolean retained) {
        try {
            mqttAndroidClient.publish(topic, message.getBytes(),
                QoS, retained);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void subscribe(String topic, OnMessageArrivedListener listener) {
        try {
            mqttAndroidClient.subscribe(topic, 0);
```

```
            subscriptions.put(topic, listener);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void unsubscribe(String topic) {
        try {
            mqttAndroidClient.unsubscribe(topic);
            subscriptions.remove(topic);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public boolean isConnected() {
        return mqttAndroidClient.isConnected();
    }

    public interface OnMessageArrivedListener {
        void onMessageArrived(String topic, String message);
    }

}
```

Ukládání zařízení

Pro ukládání byla použita knihovna Torch, která zjednodušuje zápis a čtení z SQLite databáze, pro její použití je nutné ji přidat jako závislost do **build.gradle** a zároveň v **build.gradle** použít plugin **android-apt**, který se stará o generování tříd pomocí anotací. Následně bylo potřeba incializovat Torch v **SojournerApplication** tak, že ta třída bude dědit z **TorchApplication**, což provede veškerou inicializaci za programátora. Pak je už jen nutné implementovat metodu **getMetadataForRegistration** tak, aby vracela pole obsahující **NázevModelu$.create()**. Tímto způsobem se zaregistrují všechny modely, které bude databáze používat. Model pro zařízení vypadá takto:

```
@Entity
public class DeviceModel {
    @Id
    public Long id;
    public String name;
    public String deviceName;
    public int type;
}
```

Anotace **@Entity** označuje, že se jedná o POJO, které použije Torch ke generování **DeviceModel$** a **Id** označuje primární klíč.

```
// Uložení modelu
TorchService.torch().save().entity(modelKUložení);
// Načtení modelů
List<DeviceModel> models = TorchService.torch().load()
                                        .type(DeviceModel.class).list();
```

Hlavní aktivita

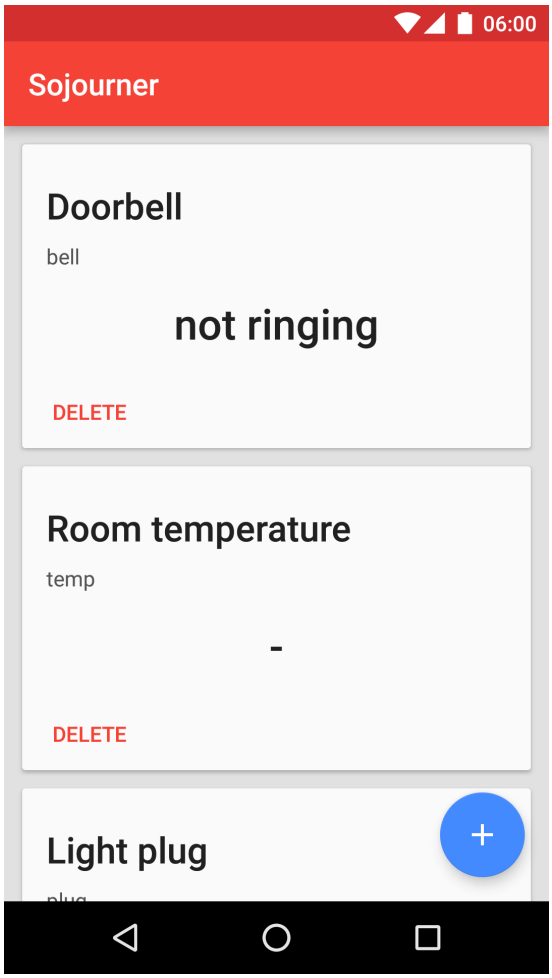
Hlavní aktivita obsahuje **RecyclerView**, což je ovládací prvek určený k zobrazování seznamů, v tomto případě ale zobrazuje karty, které slouží k ovládání jednotlivých zařízení. V pravém dolním rohu je umístěno tzv. **Floating action button** - kulaté tlačítko, přičemž po kliknutí na něj dojde k otevření aktivity pro přidání zařízení. V **ActionBaru** je v overflow menu tlačítko pro SmartConfig.

Aktivita pro přidávání zařízení

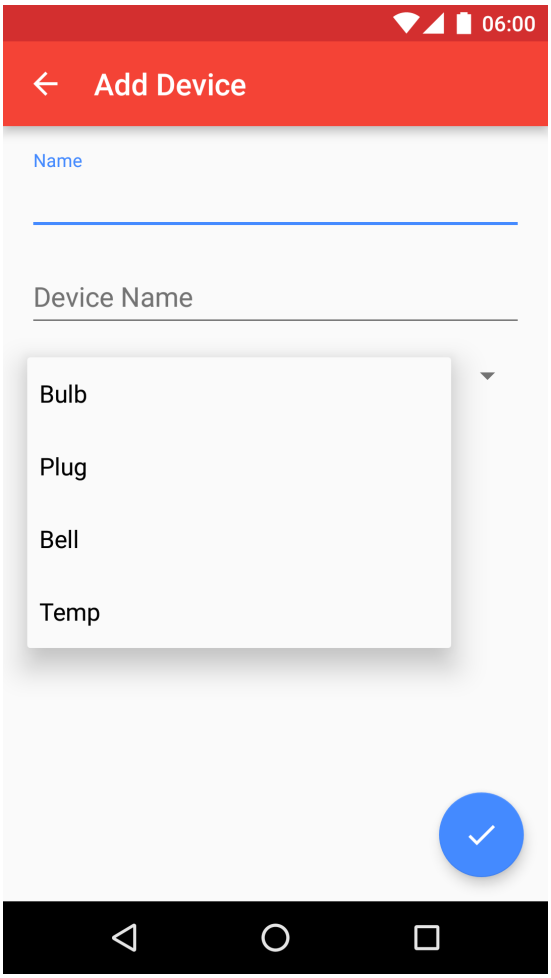
Po kliknutí na **Floating action button** se zobrazí aktivita pro přidávání zařízení, které obsahuje vstupní pole pro popisný název zařízení, který se objeví v titulku karty, a reálný název zařízení, který slouží ke generování topicu. Dále obsahuje **Spinner**, což je pop-up nabídka obsahující seznam možných zařízení. V pravém dolním rohu je **Floating action button** sloužící k uložení modelu a návratu do hlavní aktivity.

Konfigurační aktivita

Konfigurační aktivita slouží ke konfiguraci zařízení pomocí **SmartConfig**. Pro **SmartConfig** byl použit kód z oficiální aplikace pro **SmartConfig** od Espressifu.



Hlavní aktivita



Aktivita pro přidávání zařízení

Závěr

Cílem této práce bylo prokázat, jestli je možné vytvořit IoT platformu založenou na protokolu MQTT, což možné je, dokonce si troufám říci, že to funguje velmi dobře. Na druhou stranu obsahuje Pathfinder spoustu problémů ať už s hardwarem, softwarem nebo s UX. Tyto problémy je rozhodně nutné vyřešit dříve, než dojde k jakémukoliv dalšímu vývoji Pathfinderu. Pokusím se nastínit řešení některých z nich:

- Nejednotnost základního zapojení – v průběhu vývoje jednotlivých zařízení došlo nedopatřením k zapojení například tlačítka pro reset na jiný pin.
- Platforma byla zamýšlena tak, aby byl v jejím středu server, zatímco mnohem lepší bude, když v jejím středu bude mobilní zařízení, a to z důvodu mnohem snadnější konfigurace a vývoje. MQTT bude samozřejmě stále na serveru, ale veškerá konfigurace bude probíhat přes mobilní zařízení.
- Není vyřešeno jakým způsobem budou zařízení reagovat na zprávy jiných zařízení, jak je budou vyhodnocovat (porovnávat hodnoty atd.) a jakým způsobem bude fungovat konfigurace toho vyhodnocování.
- Zařízení musí být mnohem uživatelsky přívětivější. Není možné, aby zařízení zůstalo ve stavu, který uživatel nebude schopen diagnostikovat a vyřešit.
- Musí se vyřešit, jakým způsobem se přidá podpora pro použití zařízení v jiných platformách, popřípadě jakým způsobem se budou zařízení třetích stran adaptovat na zařízení vytvořená pro Pathfinder.

Osobně doufám, že se mi alespň některé z těchto problémů povede vyřešit co nejdříve, aby mohla být sepsána dobrá dokumentace a celý projekt mohl být uvolněn jako open-source.

Zdroje

- *Sming: Open Source framework for high efficiency native ESP8266 development* [online]. 2016 [cit. 2016-02-06]. Dostupné z: https://github.com/SmingHub/Sming
- *Nodemcu-firmware: lua based interactive firmware for mcu like esp8266 http://nodemcu.com* [online]. 2016 [cit. 2016-02-06]. Dostupné z: https://github.com/nodemcu/nodemcu-firmware
- *Nodemcu-devkit: A development kit for NodeMCU firmware* [online]. 2016 [cit. 2016-02-06]. Dostupné z: https://github.com/nodemcu/nodemcu-devkit
- *MQTT Essentials* [online]. 2016 [cit. 2016-02-06]. Dostupné z: http://www.hivemq.com/blog/mqtt-essentials/
- *Free Shipping New Energy-Saving 38 LEDs Lamps DIY Kits Electronic Suite 1 Set* [online]. 2016 [cit. 2016-02-06]. Dostupné z: http://www.aliexpress.com/item/Free-Shipping-New-Energy-Saving-38-LEDs-Lamps-DIY-Kits-Electronic-Suite-1-Set/32279273107.html
- *Torch: Android ORM framework.* [online]. 2015 [cit. 2016-02-06]. Dostupné z: https://github.com/brightify/torch
- *Androidannotations: Fast Android Development. Easy maintainance.* [online]. 2016 [cit. 2016-02-06]. Dostupné z: https://github.com/excilys/androidannotations