VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SERVER PRO SBĚR SENZORICKÝCH DAT A ŘÍZENÍ AKTIVNÍCH PRVKŮ

SERVER FOR COLLECTING SENSOR DATA AND CONTROL OF ACTIVE ELEMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JOZEF HALAJ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN VIKTORIN**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Halaj Jozef**
Obor: Informační technologie
Téma: **Server pro sběr senzorických dat a řízení aktivních prvků**
Server for Collecting Sensor Data and Control of Active Elements
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s projektem BeeeOn (systém pro řízení inteligentní domácnosti) a s problematikou komunikace se vzdálenými senzory a aktivními prvky.
2. Nastudujte principy komunikace pomocí technologie Web Socket s ohledem na bezpečnost a možnost obsluhovat vysoký počet vzdálených zařízení.
3. Navrhněte princip komunikace mezi serverem a vzdálenými senzorickými a aktivními prvky. Zohledněte zejm. spolehlivost doručení dat, množství různých měřených veličin, testovatelnost a škálovatelnost.
4. Implementujte server využívající navržený princip komunikace.
5. Otestujte řešení pomocí automatizovaných a poloautomatizovaných testů.
6. Zhodnoťte dosažené výsledky a diskutujte další možnosti práce.

Literatura:

- Dle dohody s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Viktorin Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Bakalárska práca sa zaoberá problematikou komunikácie so vzdialenými senzormi a aktívnymi prvkami v systéme inteligentnej domácnosti s názvom BeeeOn. Práca popisuje systém BeeeOn, jeho jednotlivé časti a princíp pôvodnej nevyhovujúcej komunikácie. Pre komunikáciu s bránami BeeeOn je využitá technológia WebSocket, ktorá umožňuje komunikovať i v sieťach s obmedzeným prístupom na privilegované porty. Implementovaný server je v princípe schopný obsluhovať vysoký počet brán BeeeOn, pomocou ktorých sú vzdialené zariadenia pripojené k systému. Komunikácia je zabezpečená pomocou SSL/TLS, používa potvrdzovacie mechanizmy pre zaručenie spoľahlivosti a je jednoducho rozšíriteľná o ďalšie potrebné správy. Prináša do systému možnosť zasielania asynchrónnych príkazov na bránu BeeeOn a pripojené zariadenia. Server je implementovaný v jazyku C++. Najbežnejšie scenáre komunikácie boli otestované automatickými testami.

Abstract

Bachelor's thesis aims on communication with remote sensors and active elements in the smart home system called BeeeOn. The individual parts of the BeeeOn system are described with respect to the current insufficient communication principle and implementation. For communication with BeeeOn gateways, the WebSocket technology is used. It allows communication on netwoks with a restricted access to privileged ports. The implemented server is in principle capable of serving a high number of BeeeOn gateways that works as a bridge among the server and the remote sensors. The communication is secured with SSL/TLS, it uses confirmation mechanism to guarantee reliability and it is easily extendable by other protocol messages. It enables the system to send asynchronous commands to the BeeeOn gateway and to the connected devices. The server is implemented in C++ language. The most common communication scenarios were tested by automated tests.

Kľúčové slová

inteligentná domácnosť, BeeeOn, WebSocket, JSON, C++, komunikácia, server, Gateway

Keywords

smart home, BeeeOn, WebSocket, JSON, communication, C++, server, Gateway

Citácia

HALAJ, Jozef. *Server pro sběr senzorických dat a řízení aktivních prvků*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Viktorin Jan.

Server pro sběr senzorických dat a řízení aktivních prvků

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jana Viktorina. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Jozef Halaj
19. mája 2017

Poďakovanie

Rád by som sa poďakoval vedúcemu bakalárskej práce Ing. Janovi Viktorinovi za poskytnuté odborné konzultácie a hlavne konštruktívne pripomienky, či už pri písaní textovej časti, návrhu alebo implementácii. Ďalej by som sa chcel poďakovať celému tímu BeeeOn za výbornú spoluprácu na projekte.

Obsah

1 Úvod **2**

2 BeeeOn **3**
2.1 Architektúra 3
2.2 Koncové zariadenia 4
2.2.1 Identifikácia koncových zariadení v systéme 5
2.3 Brána 6
2.3.1 Aplikácia AdaApp 7
2.4 Server 8
2.4.1 Komunikácia s bránou 9
2.4.2 Nový server 10

3 WebSocket **12**
3.1 Handshake 13
3.2 Prenos dát 14
3.3 Zabezpečenie komunikácie 16
3.4 Zhrnutie 17

4 Návrh a implementácia **18**
4.1 Návrh komunikácie 18
4.2 Architektúra 18
4.2.1 Trieda GatewayCommunicator 20

5 Záver **22**

Literatúra **24**

Kapitola 1

Úvod

Spolu s technickým pokrokom sa zvyšujú požiadavky ľudí na komfort a čo najkvalitnejší životný štandard. Bežné úkony v domácnosti, ktoré nám ešte donedávna boli prirodzené, nahradzujú elektronické spotrebiče, uľahčujúce a zefektívňujúce beh domácnosti. Internet už dávno neslúži len ako zdroj informácií, ale čím ďalej viac preniká do nášho života a postupne aj do bežných spotrebičov.

Vznikajú inteligentné domácnosti, ktorých cieľom je zvýšiť komfort, efektívnosť, bezpečnosť a taktiež znížiť náklady na prevádzku domácnosti. Spotrebiče je možné ovládať na diaľku a domácnosť je možné monitorovať, a taktiež automatizovať. Príkladom môže byť vypnutie osvetlenia a zamknutie dverí po odchode z domu, regulácia vykurovania na základe snímanej teploty alebo zapnutie ohrevu vody.

Inteligentnej domácnosti sa venuje aj projekt BeeeOn vyvíjaný na Fakulte Informačných technológií v Brne. Projekt sa zameriava na jednoduchosť, a zjednotenie monitorovania a ovládania zariadení od rôznych výrobcov. Zariadenia, obsahujúce senzory na snímanie vlastností prostredia a aktívne prvky na ovládanie domácnosti sa pripájajú na bránu inteligentnej domácnosti BeeeOn, pomocou ktorej komunikujú cez internet so zbytkom systému. Ústredným prvkom systému je server, ktorý zabezpečuje centralizované ukladanie dát z pripojených brán a umožňuje zasielať riadiace príkazy zariadeniam v domácnosti.

Táto práca sa zaoberá problematikou komunikácie serveru so vzdialenými senzormi a aktívnymi prvkami v systéme BeeeOn. Jej cieľom je navrhnúť princíp komunikácie medzi serverom a pripojenými domácnosťami skrz bránu BeeeOn, a následne implementovať server využívajúci navrhnutý princíp komunikácie. Navrhnutý princíp komunikácie využíva technológiu WebSocket, čím sú odstránené pôvodné problémy blokovania komunikácie rôznymi firewallmi v existujúcej sieťovej infraštruktúre. Umožňuje bezpečné pripojenie a obsluhu veľkého počtu brán, poskytuje spoľahlivosť doručenia prenášaných dát, a je jednoducho rozšíriteľný o ďalšie typy potrebných správ. Prináša do systému možnosť zasielania asynchrónnych príkazov na bránu a k nej pripojených zariadení a následné reportovanie stavu vykonávania týchto príkazov. Napríklad, zmena hodnoty nejakého aktívneho prvku je v princípe asynchrónna operácia, čo bolo v pôvodnom systéme ignorované.

Text práce je rozdelený do niekoľkých častí. Prvá časť predstavuje a popisuje systém pre riadenie inteligentnej domácnosti BeeeOn, jeho architektúru, jednotlivé súčasti systému, ich úlohy a prepojenie. Taktiež popisuje princíp a nevyhovujúci stav komunikácie so zariadeniami tretích strán v domácnosti, v podobe senzorov a riadiacich prvkov, pripojenými na bránu BeeeOn. Ďalšia časť práce sa venuje technológií WebSocket. Návrh novej komunikácie a implementácia serveru využívajúceho princíp tejto komunikácie sú popísané v poslednej časti práce.

Kapitola 2

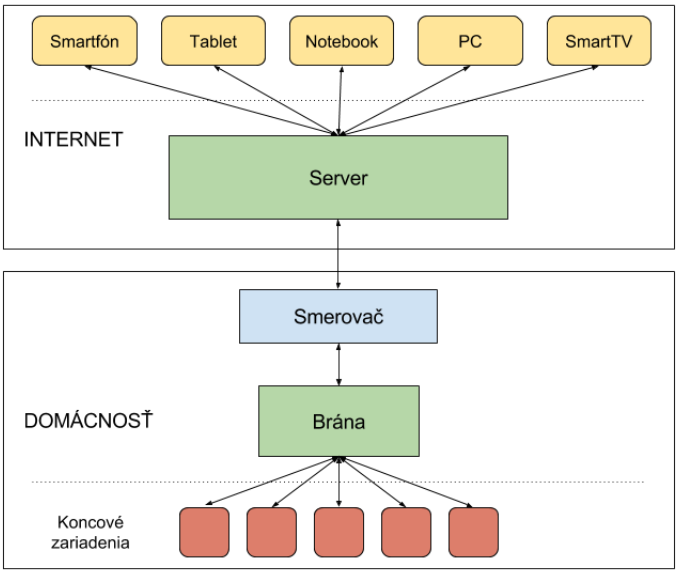
BeeeOn

BeeeOn je open-source projekt vyvíjaný na Fakulte Informačných Technológií VUT v Brne, ktorý sa zaoberá rozvojom Internetu vecí. Jeho cieľom je vytvorenie modulárneho systému pre riadenie a automatizáciu inteligentnej domácnosti, s dôrazom na prístupnosť, bezpečnosť a jednoduchú rozšíriteľnosť o ďalšie prvky.

V súčasnosti systém poskytuje všetky hardvérové komponenty potrebné pre beh inteligentnej domácnosti a rovnako ponúka softvérové riešenia. Technické špecifikácie, detaily návrhu hardvéru, ako aj schémy plošných spojov sú dostupné na oficiálnych stránkach BeeeOn [1] a v repozitároch na serveri GitHub [2], kde sú taktiež zverejňované všetky zdrojové kódy projektu. Nasledujúce informácie o systéme boli čerpané z oficiálnych stránok projektu [1], interných wiki stránok [7][8][10] a zo zdrojových kódov projektu [2].

2.1 Architektúra

Architektúra systému, sa skladá z niekoľkých samostatných komponentov, ktoré sú na najvyššej úrovni abstrakcie zoskupené do štyroch hlavných častí.



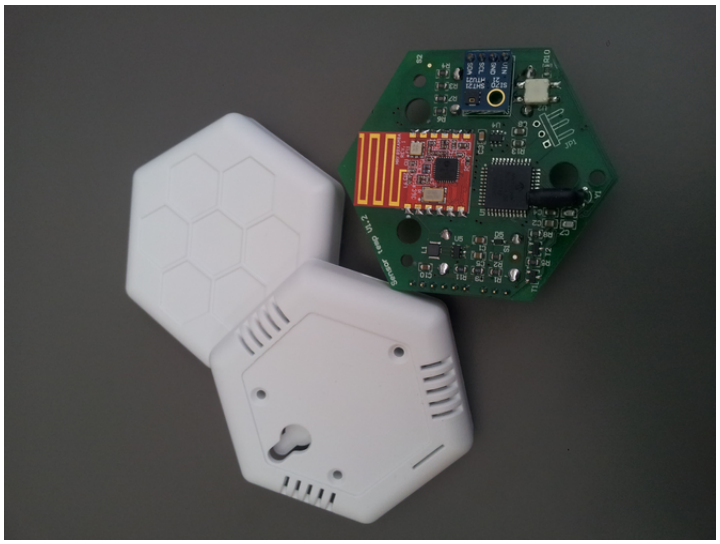
Obr. 2.1: Architektúra systému BeeeOn

Koncové zariadenia vo forme senzorov a aktívnych prvkov sú bezdrôtovo pripojené na bránu. **Brána** predstavuje centrálny prvok domácnosti a spája koncové zariadenia so zbytkom systému. Pomocou smerovača v lokálnej sieti je pripojená do siete Internet, cez ktorú komunikuje so serverom za účelom zasielania dát z koncových zariadení a prijímania riadiacich príkazov. **Server** je zodpovedný za spracovanie požiadaviek a ukladanie dát od brány, ale na druhej strane spracúva tiež požiadavky od užívateľa, ktorý so systémom komunikuje pomocou **užívateľského rozhrania** vo forme mobilnej alebo webovej aplikácie. V súčasnosti BeeeOn poskytuje android aplikáciu, ktorá užívateľovi prezentuje všetky dôležité informácie o jeho bráne a koncových zariadeniach a umožňuje zasielať riadiace príkazy. Webová aplikácia je vo vývoji.

2.2 Koncové zariadenia

Koncové zariadenia sa nachádzajú na spodnej vrstve architektúry. Sú to **bezdrôtovo** pripojené fyzické zariadenia, komunikujúce s centrálnou bránou. Koncovým zariadením môže byť akýkoľvek prístroj v domácnosti, ktorý umožňuje snímať vlastnosti prostredia alebo do prostredia zasahovať. Na základe týchto skutočností ich rozdeľujeme do dvoch skupín:

- **Senzory** slúžia na získavanie dát z okolitého prostredia. Namerané dáta, väčšinou fyzikálne veličiny (napr. teplota, tlak vzduchu, elektrická spotreba atď.), sú v určitých intervaloch zasielané na bránu. Tá ich spracúva a distribuuje do vyšších vrstiev systému.
- **Aktívne prvky (aktuátory)** naopak ovplyvňujú stav okolia na podnet riadiacich pokynov od brány. Môže ísť o jednoduché zopnutie spínača elektrickej zásuvky, ovládanie pohonu termostatickej hlavice a pod.



Obr. 2.2: BeeeOn Sensor [1]

Zariadenia v domácnosti by mali byť primárne jednoduché (jednoúčelové, cenovo dostupné, s nízkymi energetickými nárokmi), ale taktiež to môžu byť komplexné zariadenia, zložené z viacerých senzorov a aktívnych prvkov. Vďaka bezdrôtovej komunikácií s bránou

môžu byť ľubovoľne umiestnené a premiestňované, čo prináša jednoduchú integráciu do každej domácnosti.

V súčastnosti systém podporuje vlastný senzor, vyvinutý v rámci projektu a širokú škálu zariadení tretích strán (napr. Thermona [6], Jablotron [3], Open ZWave [4], Philips Hue [5]). BeeeOn senzor, zobrazený na obrázku 2.2, je schopný merať teplotu (po pripojení externého čidla aj externú), vlhkosť vzduchu a komunikuje pomocou vlastného FIT protokolu. FIT protokol je proprietárny bezdrôtový komunikačný protokol navrhnutý špeciálne pre Internet vecí.

2.2.1 Identifikácia koncových zariadení v systéme

Zariadenia tretích strán komunikujú rôznymi bezdrôtovými protokolmi na rôznych frekvenciách. Preto je nutné, aby bol k bráne rozhraním USB pripojený adaptér, zabezpečujúci komunikáciu s týmito zariadeniami pomocou vhodného protokolu.

Každé koncové zariadenie je v systéme identifikované pomocou jednoznačného identifikátoru. Identifikátor predstavuje 64 bitové číslo a je jedinečný v rámci celého systému BeeeOn. Skladá sa z 8 bitového prefixu, ktorý značí typ komunikačného protokolu a samotnej identifikácie konkrétneho fyzického zariadenia.

Okrem jedinečnej identifikácie konkrétnych zariadení je v systéme potrebné rozpoznávať rôzne typy zariadení. Na rozpoznanie a správu všetkých podporovaných zariadení v systéme BeeeOn slúži **tabuľka zariadení**. Je to XML dokument obsahujúci špecifikácie všetkých podporovaných zariadení. Každé nové podporované zariadenie musí byť pridané do tejto tabuľky, ktorá je zdieľaná medzi jednotlivými časťami systému BeeeOn. Všetky zariadenia v tabuľke sú identifikované jedinečným identifikátorom (*id*) slúžiacim na rozpoznanie typu zariadenia v jednotlivých častiach systému, menom (*name*), výrobcom (*manufacturer*) a modulmi (*modules*). Každý modul v rámci zariadenia má jedinečný identifikátor a môže to byť senzor, aktívny prvok, batéria atď.

```
1  <device id="29" name="Everspring Wireless SmokeDetector SF812">
2          <name>T:DEV_EVERSPRING_SF812_SMOKE_DETECTOR</name>
3          <manufacturer>T:MANUFACTURER_EVERSPRING</manufacturer>
4          <modules>
5                  <sensor id="0" type="0x03">
6                          <group>T:ZONE_1</group>
7                          <name>T:EVERSPRING_SF812_SMOKE_SENSOR</name>
8                          <values name="T:EVERSPRING_SF812_SMOKE_SENSOR_STATE">
9                                  <value id="0">T:OFF</value>
10                                 <value id="1">T:ON</value>
11                         </values>
12                 </sensor>
13                 <battery id="1" type="0x08" />
14         </modules>
15 </device>
```

Ukážka 2.1: Príklad špecifikácie zariadenia v tabuľke zariadení

Aby mohli koncové zariadenia komunikovať so systémom, musia byť spárované s bránou. Po úspešnom spárovaní sú súčasťou systému, teda môžu prijímať príkazy na nastavenie aktívnych prvkov a zasielať dáta bráne.

Spárované zariadenia sa môžu uspať až do času, kedy je naplánované prevedenie ďalšej činnosti – zmeranie meranej veličiny, odoslanie dát alebo prijatie riadiacich pokynov. Tento proces uspávania zvyšuje životnosť batérie. Napríklad teplotný senzor sa môže prebudiť

raz za 5 minút, zmeria teplotu, odošle namerané dáta a znova sa uspí. Podobné uspávanie sa môže diať aj u aktívnych prvkov, kde to môže narozdiel od senzorov spôsobiť určité problémy. Napríklad, užívateľ dá pokyn na nastavenie hodnoty daného prvku, ale ten je uspatý a hodnota sa nastaví až po prebudení. Túto skutočnosť je potrebné reflektovať užívateľovi.

2.3 Brána

Brána (historicky taktiež označovaná ako *adaptér*), zobrazená na obrázku 2.3, je fyzické zariadenie predstavujúce centrálny prvok inteligentnej domácnosti BeeeOn. Na jednej strane je pripojená na internet a pomocou zabezpečeného TCP spojenia komunikuje so serverom. Na druhej strane pomocou bezdrôtového protokolu komunikuje s pripojenými zariadeniami v domácnosti.

Obr. 2.3: BeeeOn Gateway [1]

Hlavnou úlohou brány je zabezpečenie obojsmernej komunikácie medzi serverom a koncovými zariadeniami. Zo strany serveru sú prijímané riadiace príkazy, ktoré sa po spracovaní delegujú na koncové prvky. Z druhej strany sú prijímané dáta zo senzorov, ktoré sú po spracovaní bránou odoslané na server.

Brána poskytuje taktiež ochranu proti strate senzorických dát v prípade výpadku internetového spojenia. Počas výpadku sú dáta ukladané najprv do vyrovnávacej pamäte a po dosiahnutí určitej hranice, sú zapisované na SD kartu. Po opätovnom spojení sú všetky uložené dáta postupne poslané na server.

Každá brána v systéme je identifikovaná jednoznačným identifikátorom. Identifikátor predstavuje 16 miestne číslo v dekadickej sústave, ktoré je v rámci systému jedinečné. Tento identifikátor je vygenerovaný pri výrobe brány spolu s odpovedajúcim QR kódom, ktorý slúži na jednoduchú registráciu brány v mobilnej aplikácií užívateľa.

Hardvér

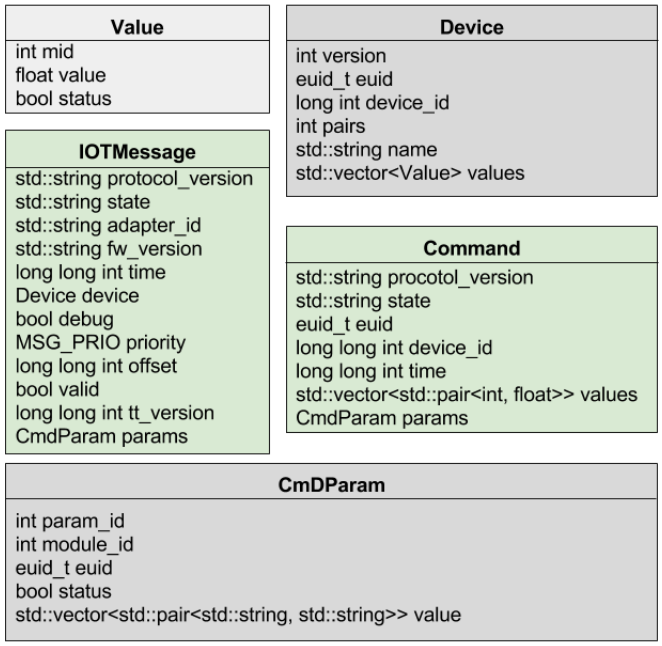
Hardvérovú realizáciu brány predstavuje doska A10-OLinuXino-LIME [13] od spoločnosti Olimex, na ktorej beží operačný systém Linux. Dôležitou súčasťou brány je aj rozširujúca

doska, obsahujúca ďalšie potrebné súčasti (napr. rádiovú vrstvu, RTC [1] a vstavaný senzor tlaku)[8]. Práve rádiová vrstva na tejto doske je zodpovedná za komunikáciu s BeeeOn senzormi pomocou FIT protokolu [9]. Pre komunikáciu so zariadeniami tretích strán, ktoré komunikujú rôznymi bezdrôtovými protokolmi, je potrebné pripojiť externý USB adaptér.

2.3.1 Aplikácia AdaApp

Hlavnú funkcionalitu brány zaisťuje aplikácia AdaApp. Okrem obojsmerného prenosu dát medzi serverom a koncovými zariadeniami plní aj ďalšie úlohy. Aplikácia po štarte, a takisto priebežne, kontroluje potrebné prostriedky, ako je nastavený správny čas a zabezpečené pripojenie na server. V prípade, že všetko funguje, prenáša dáta. V opačnom prípade, si ukladá dáta zo senzorickej siete a po obnove prostriedkov ich zasiela na server.

Aplikácia vytvára jednotné rozhranie na komunikáciu medzi serverom a koncovými zariadeniami. Jej návrh bol postavený na modularite jednotlivých častí aplikácie tak, aby bolo možné zapnúť a používať len potrebné moduly aplikácie. Hlavnými a zároveň povinnými modulmi sú agregátor, a modul pre sieťovú komunikáciu, ktoré spolu úzko spolupracujú. Na komunikáciu medzi jednotlivými modulmi slúžia interné štruktúry zobrazené na obrázku 2.4.



Obr. 2.4: Štruktúry na komunikáciu

- **Modul pre sieťovú komunikáciu (TCP)** slúži na pripojenie a následnú komunikáciu so serverom. Beží v samostanom vlákne, v ktorom prijíma dáta zo serveru v XML podobe. Po prijatí dát ich prevedie do internej štruktúry `Command` a využitím agregátoru ich deleguje modulom koncových zariadení, pre ktoré sú určené. Naopak agregátor ho využíva na prevod dát z internej štruktúry `IOTMessage` do XML podoby, odoslanie dát na server a prijatie odpovede zo serveru. Komunikácia so serverom je bližšie popísaná v kapitole 2.4.1.

[1] Real-time clock – hodiny reálneho času

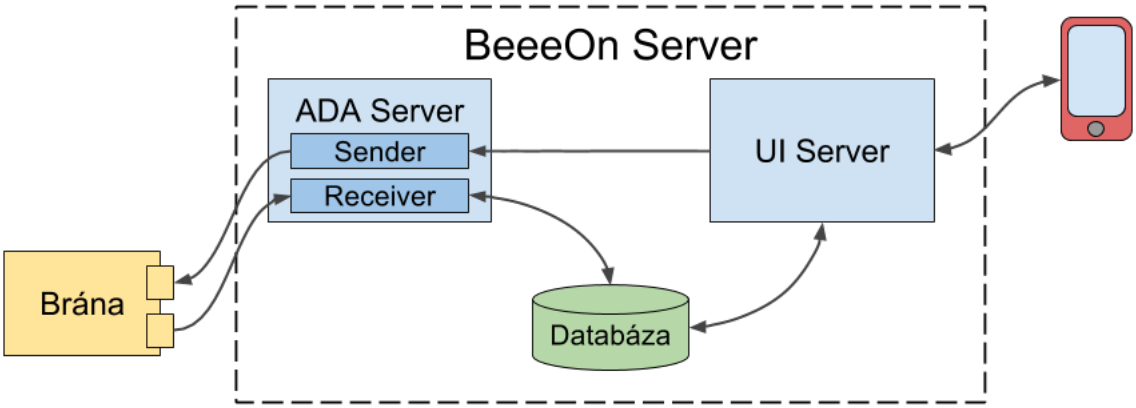
- **Agregátor** je modul tvoriaci centrálny bod komunikácie aplikácie. Beží v samostatnom vlákne, v ktorom sa stará o vyrovnávaciu pamäť a odosielanie správ z tejto pamäte na server. Moduly koncových zariadení ho využívajú na odosielanie dát v štruktúre `IOTMessage`, ktorú predáva sieťovému modulu na odoslanie. Z návratovej hodnoty zistí, či boli dáta skutočne odoslané. V prípade že nie, ukladá ich do vyrovnávacej pamäte. Modul pre sieťovú komunikáciu ho využíva na delegovanie správy v štruktúre `Command` modulom koncových zariadení, pre ktoré je určená.

- **Moduly koncových zariadení** zabezpečujú komunikáciu medzi agregátorom a koncovými zariadeniami. Jeden modul obsluhuje sadu zariadení, ktoré využívajú rovnaký komunikačný protokol. Každý modul využíva iný spôsob komunikácie so zariadeniami, ale s agregátorom komunikuje pomocou rozhrania v podobe štruktúr `IOTMessage` a `Command`.

Aplikácia nebola pôvodne stavaná na systematické pridávanie nových typov koncových zariadení. S postupným rozširovaním podpory pre zariadenia tretích strán v systéme bolo nutné zasahovať do jadra aplikácie a upravovať ho. Vytváranie nových modulov pre tieto zariadenia neustále vyžadovalo zmeny v rozhraní vo forme interných štruktúr. Vznikajúce problémy nebolo možné riešiť systematicky a preto sa navrhla nová aplikácia, ktorá sa aktuálne vyvíja spolu s novým serverom.

2.4 Server

Server je ústredným prvkom systému BeeeOn. Zaisťuje hlavnú logiku riadenia inteligentnej domácnosti a je nutný pre komunikáciu medzi užívateľom a koncovými zariadeniami v domácnosti, resp. bránou. Poskytuje dátové úložisko vo forme relačnej databázy, v ktorej sú uložené všetky potrebné informácie o užívateľoch, ich právach, pripojených bránach a koncových zariadeniach, a taktiež je ukladaná história nameraných dát. Je zodpovedný za spracovanie požiadaviek a ukladanie dát od brány, a zároveň prijíma požiadavky od užívateľa a poskytuje mu potrebné informácie alebo zasiela riadiace príkazy na bránu, resp. koncové zariadenia. Server je rozdelený na dve serverové aplikácie, ADA Server a UI Server, ako je zobrazené na obrázku 2.5. Tieto serverové aplikácie sú spustené ako služby na jednom fyzickom serveri s operačným systémom Linux, ale je možné aj ich rozdelenie na rôzne fyzické stroje.



Obr. 2.5: BeeeOn Server

ADA Server je serverová aplikácia zabezpečujúca komunikáciu s pripojenými bránami a ukladanie nameraných dát do databázy. Po pripojení brány si ukladá TCP socket do kontajnera spojení a cez toto spojenie následne zasiela na bránu riadiace príkazy prijímané z UIServera. Pre každú ďalšiu požiadavku a namerané dáta od brány sú vytvárané nové TCP spojenia a taktiež každá správa od UIServera vyžaduje nové TCP spojenie. Tieto dočasné spojenia slúžia len na prijatie správy a zaslanie odpovede a následne sú uzavreté. Aplikácia je rozdelená na dve hlavné časti, ktoré bežia v samostatných vláknach:

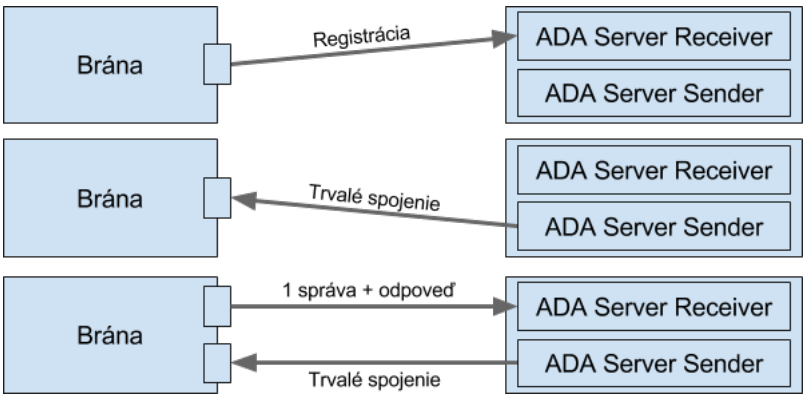
- **ADA Server Receiver** slúži na prijatie prichádzajúceho spojenia s bránou, obsluhu prijatej požiadavky a zaslanie odpovede bráne. V prípade, že sa jedná o registračnú správu, ukladá si spojenie, cez ktoré bude môcť ADA Server Sender zasielať správy bráne.
- **ADA Server Sender** slúži na prijatie prichádzajúceho spojenia s UI Serverom, spracovanie prijatej požiadavky do podoby zrozumiteľnej pre bránu a zaslanie bráne. Na správy zasielané zo serveru brána neodpovedá a preto je UI Serveru zaslaná len informácia, či bola správa odoslaná.

Spracovanie požiadaviek neprebieha vo vláknach týchto dvoch častí, ale využíva sa threadpool (fond vlákien). Vo vláknach threadpoolu sú spúšťané pracovné vlákna, ktoré obslúžia danú požiadavku a ukončia sa. Komunikácia s bránou je bližšie popísaná v kapitole 2.4.1.

UI Server poskytuje užívateľským rozhraniam (v súčasnoti len vo forme mobilnej android aplikácie) možnosť komunikovať so zbytkom systému BeeeOn. Na komunikáciu je využívané TCP spojenie so serverom a proprietárny XML protokol, umožňujúci autentifikáciu užívateľa, ukladanie a získavanie užívateľských dát z databázy a zasielanie riadiacich pokynov na bránu resp. koncové zariadenia.

2.4.1 Komunikácia s bránou

Ako už bolo spomenuté súčasná komunikácia s bránou vyžaduje vytváranie obrovského množstva TCP spojení. Prvé spojenie vytvára brána po zapnutí a cez toto spojenie posiela registračnú správu, na ktorú očakáva odpoveď označujúcu, že server akceptoval jej pripojenie a je schopný s ňou komunikovať. Spojenie je ďalej udržované, za účelom jednosmernej komunikácie zo strany serveru. Zasielanie riadiacich príkazov od užívateľa však vyžaduje vytváranie nového spojenia pre každú správu medzi UI Serverom a ADA Serverom. Na správy iniciované zo strany serveru nechodia od brány žiadne odpovede. Užívateľ sa teda nie je schopný dozvedieť, či bola správa bránou prijatá.



Obr. 2.6: Komunikácia brány so serverom

Správy od brány ako zasielanie dát alebo získanie dodatočných informácií zo strany serveru vyžadujú vytváranie nového spojenia s ADA Serverom pre každú správu. Na tieto správy server zasiela odpoveď cez vytvorené spojenie, ktorým prišla požiadavka. Obrázok 2.6 zobrazuje popísanú komunikáciu medzi bránou a serverom. Správy medzi bránou a serverom majú XML podobu a taktiež medzi UI Serverom a ADA Serverom.

Typy prenášaných správ

Správy zasielané z brány:

- **register** – Požiadavka na registráciu brány na serveri. Správa sa posiela len raz po zapnutí AdaApp.
  - **register** – Potvrdenie od serveru, že je brána zaregistrovaná a je pripravený od nej prijímať dáta. Server si ukladá socket, cez ktorý bude zasielať požiadavky na bránu.
- **data** – Správa slúži na prenos dát z koncových zariadení. Zariadenia posielajú naraz celú svoju štruktúru hodnôt zo všetkých senzorov a taktiež aktívnych prvkov.
  - **update** – Potvrdenie od serveru, že prijal nové dáta. Súčasťou správy je položka `time`, ktorá slúži na nastavenie intervalu pre získavanie hodnôt z koncových prvkov.

Správy zasielané zo servera:

- **set** – Nastavenie aktívneho prvku na novú hodnotu.
- **listen** – Prepnutie brány do párovacieho režimu, resp. aktivácia párovacieho režimu vo všetkých moduloch koncových zariadení.
- **clean** – Odpárovanie koncového zariadenia.

Pomocou spomínaných správ nebolo možné prenášať všetky potrebné informácie medzi bránou a servervom, preto bol XML protokol rozšírený o ďalšie typy správ:

- **getparameters** – Požiadavka brány na získanie dodatočných informácií. Obsahuje kód požiadavky, ktorý označuje typ požadovaných dát. Napríklad, môže ísť o získanie poslednej nameranej hodnoty nejakého modulu zariadenia alebo získanie spárovaných zariadení.
- **parameters** – Odpoveď od serveru na predchádzajúcu správu.

2.4.2 Nový server

Popísaný server je produkčne nasadený, udržiava sa, ale v súčasnosti sa vyvíja nový server založený na layered (troj-vrstvovej) architektúre s novým dátovým modelom kompletne prepísaný nad knižnicou Poco. Dôvodov pre vznik nového serveru bolo viacero. Jedným z nich bol postupný prechod z proprietárneho XML protokolu na REST API, kvôli rozširovaniu užívateľského rozhrania o webovú aplikáciu. Pôvodný server na to nebol stavaný a bolo komplikované v ňom robiť zmeny. Ďalším dôvodom bolo, že jednotlivé časti serveru implementovali rovnaké alebo podobné problémy rôznymi spôsobmi, využívali mix rôznych

technológií alebo nevyužívali dostupné technológie vôbec a implementovali problémy vlastným spôsobom. V jednotlivých častiach serveru vznikali zložité závislosti a preto nebolo možné robiť jednoduché zmeny a systém rozširovať.

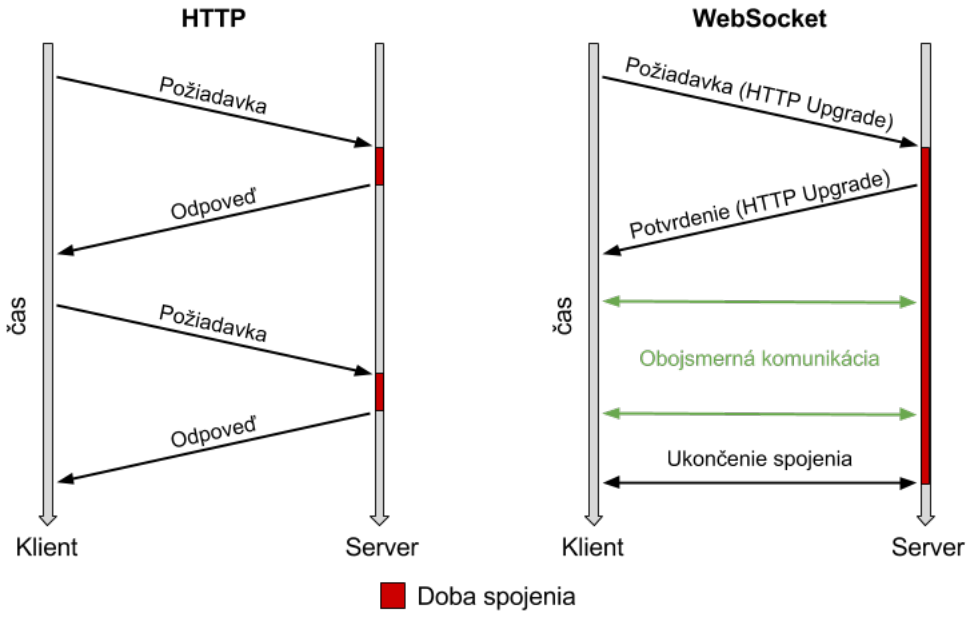
V súčasnosti je implementovaný kompletný dátový model spolu s dátovou vrstvou. Nad ňou je implementovaná serverová časť UI Server s jej servisnou vsrtvou a aplikačnou vrstvou s proprietárnym XML protokolom. Nový UI Server je chopný komunikovať s pôvodným ADA Serverom a postupne sa vyvíja aplikačná vrstva v podobe REST API.

Kapitola 3

WebSocket

WebSocket je aplikačný protokol poskytujúci full-duplex komunikáciu skrz jedno TCP spojenie. Protokol bol štandardizovaný v roku 2011 a definovaný v RFC 6455 s názvom The WebSocket protocol[12]. Primárne bol navrhnutý pre použitie vo webových aplikáciách vyžadujúcich obojsmernú komunikáciu v reálnom čase (napr. chat alebo herné aplikácie) medzi webovým prehliadačom a serverom, ale je použiteľný aj v iných aplikáciách typu klient – server. Uľahčuje prenos dát v reálnom čase zo servera a na server, čím prináša viac interakcie medzi prehliadačom a webovým serverom. Toto je možné tým, že poskytuje serveru štandardizovanú cestu na zaslanie dát prehliadaču bez toho, aby si to vyžiadal klient a umožňuje, aby boli správy prenášané tam a späť pri zachovaní otvoreného spojenia.

WebSocket je nezávislý protokol založený na TCP, avšak pre ustálenie spojenia (handshake) na prenos dát využíva HTTP[1] protokol.



Obr. 3.1: Porovnanie HTTP a WebSocket komunikácie

[1]Hypertextový prenosový protokol

Obrázok 3.1 zobrazuje porovnanie komunikácie medzi klientom a serverom u protokolov HTTP a WebSocket. Zatiaľ čo HTTP pracuje formou dotaz/odpoveď zo strany klienta a klient pre každú požiadavku typicky vytvára nové spojenie, WebSocket po ustálení spojenia umožňuje plne duplexnú komunikáciu z oboch smerov a až do požiadavky na ukončenie spojenia, zachováva spojenie otvorené.

Z historického hľadiska, vytváranie webových aplikácií, ktoré potrebujú obojsmernú komunikáciu v reálnom čase medzi klientom a serverom, vyžadovalo zneužitie HTTP na dotazovanie servera na dáta, zatiaľ čo zasielanie správ z klienta bolo realizované novými spojeniami. Toto provizórne riešenie má za následok rôzne problémy:

- Server je nútený používať množstvo rôznych TCP spojení pre každého klienta. Jedno spojenie na zasielanie dát klientovi a ďalšie pre každú prichádzajúcu správu.
- Vysoká réžia, spôsobená HTTP hlavičkou v každej správe a vytváraním nových TCP spojení.

Jednoduchšie a lepšie riešenie je použitie jedného TCP spojenia pre obojsmernú komunikáciu, čo ponúka WebSocket. Je navrhnutý pre nahradenie existujúcich neštandardizovaných obojsmerných komunikačných technológii (napr. Comet), ktoré používajú HTTP na transport kôli ťaženiu z existujúcej sieťovej infraštruktúry (proxy servery, firewall, filtrovanie, autentizácia). V kontexte využitia existujúcej webovej infraštruktúry, nasleduje existujúce technológie. Pracuje na HTTP porte 80 resp. 443 pre zabezpečenú komunikáciu pomocou SSL/TLS a taktiež podporuje HTTP proxy servery, firewall, atď. Štandard však uvádza, že WebSocket nie je viazaný na HTTP a budúce implementácie by mohli využiť jednoduchší handshake cez dedikovaný port bez nutnosti zmeny celého protokolu.

Špecifikácia WebSocket protokolu definuje `ws` a `wss` ako 2 nové URI[2] adresy pre nezabezpečené a zabezpečené spojenie v nasledujúcom formáte:

```
ws-URI = "ws://"host [ ":"port ] path [ "?"query ]
wss-URI = "wss://"host [ ":"port ] path [ "?"query ]
```

Východzia hodnota portu je 80 pre `ws` a 443 pre `wss`.

WebSocket protokol pozostáva z dvoch častí, konkrétne handshake a prenos dát, ktoré sú popísané v nasledujúcich kapitolách.

3.1 Handshake

Na nadviazanie WebSocket spojenia klient zasiela WebSocket handshake požiadavku. Jedná sa o HTTP Upgrade požiadavku z dôvodu spomínanej kompatibility. Výhodou je, že jeden port môže byť použitý ako pre HTTP, tak pre WebSocket. Server mu odpovedá Websocket handshake odpoveďou. Po úspešnom nadviazaní spojenia je vytvorený obojsmerný komunikačný kanál, kde každá strana môže zasielať dáta nezávisle na druhej strane.

Obrázok 3.2 zobrazuje príklad handshake požiadavky. Prvý riadok indikuje že sa jedná o HTTP hlavičku a povinne obsahuje metódu GET. URI adresa tejto metódy identifikuje koncový bod WebSocket spojenia, čo umožňuje, aby na jednom serveri existovalo niekoľko rôznych koncových bodov. Položka `Connection: Upgrade` v kombinácií s položkou `Upgrade: websocket` označujú, že klient chce spojenie povýšiť na WebSocket. `Host` slúži na potvrdenie názvu hostiteľa danej služby a v položke `Origin` je predaný kontext použitia.

[2] Universal Resource Identifier - jednotný identifikátor zdroja

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Obr. 3.2: WebSocket handshake požiadavka

`Sec-WebSocket-Key` je náhodne vygenerovaný base64 reťazec použitý na autentizáciu servera. Server klientovi vráti hodnotu zostavenú z 2 kľúčov. Prvým kľúčom je práve hodnota položky `Sec-WebSocket-Key` odoslanej klientom. Táto hodnota je zbavená všetkých bielych znakov pred a za textom, následne je spojená s GUID[3] pre WebSocket protokol. Spojenie týchto kľúčov je odoslané klientovi ako hash hodnota vytvorená pomocou SHA-1[4]. Klient obdrží túto hodnotu v base64 kódovaní v položke `Sec-WebSocket-Accept`, čím si overí, že skutočne odpovedá server, ktorý požiadavku prijal. Položka `Sec-WebSocket-Protocol` slúži na označenie aké subprotokoly (protokoly aplikačnej vrstvy zabalené do WebSocketu) klient podporuje. Server si vyberie jeden alebo žiadny z týchto protokolov a vráti túto hodnotu klientovi v svojej handshake odpovedi. Klient z položky hlavičky rovnakého názvu zistí, ktorý subprotokol server zvolil. Posledná položka `Sec-WebSocket-Version` označuje verziu WebSocket protokolu. V súčasnosti je štandardizovaná hodnota 13. V prípade, že server nepodporuje danú verziu protokolu, môže s klientom vyjednávať o použití inej zaslaním podporujúcich verzií v handshake odpovedi.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Obr. 3.3: WebSocket handshake odpoveď

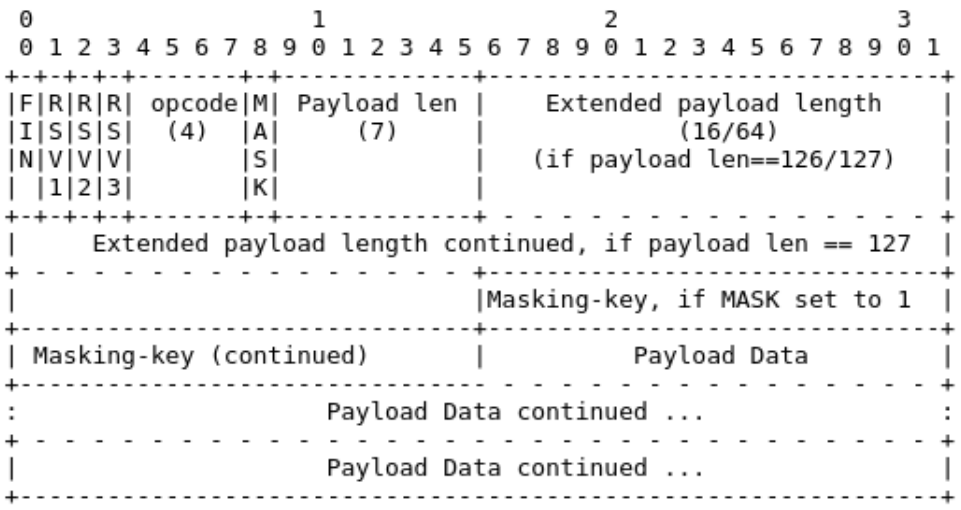
Obrázok 3.3 zobrazuje odpoveď od servera. Prvý riadok obsahuje stavový kód `101 Switching Protocols`. Každý iný stavový kód ako 101 značí, že handshake neprebehol úspešne. Po úspešnom handshaku je vytvorené WebSocket spojenie a nasleduje prenos dát.

3.2 Prenos dát

WebSocket protokol prenáša dáta použitím sekvencie rámcov. Prenosy dát sú nazývané správy, kde jedna správa môže byť rozdelená (fragmentovaná) do niekoľkých rámcov. Fragmentácia dovoľuje odosielať správy s neznámou veľkosťou v prípade odosielania bez využitia vyrovnávacej pamäti. Protokol umožňuje prenos textových dát v kódovaní UTF-8 a taktiež binárnych dát. Definuje rámec pomocou operačného kódu, veľkosti užitočných dát a samotných užitočných dát (payload). Niektoré bity a operačné kódy rámca sú rezervované pre budúcu expanziu protokolu.

[3] Globally Unique Identifier
[4] Secure Hash Algorithm

Z bezpečnostných dôvodov musí klientská implementácia protokolu maskovať všetky dáta, ktoré zasiela na server. Dáta sú maskované binárnou operáciou XOR pomocou náhodne vygenerovaného 32 bitového kľúča, ktorý je pridaný do hlavičky rámca. Maskovanie nijako nezabezpečuje prenášané dáta, keďže kľúč je zasielaný spolu s dátami, ale slúži na zabezpečenie náchylnej sieťovej architektúry pred potenciálnym útokom z klientskej aplikácie (proxy cache poisoning). Nevýhodou je, že protokol vyžaduje maskovanie aj v prípade zabezpečenej komunikácie WSS, kedy je maskovanie nadbytočné.



Obr. 3.4: Dátový rámec WebSocketu [12]

Obrázok 3.4 znázorňuje jednoduchú štruktúru rámca. Prvý bit FIN označuje, či sa jedná o finálny rámec správy. V prípade zasielania nefragmentovaných správ je prvý rámec zároveň posledným. Nasledujúce 3 bity RSV1, RSV2, RSV3 sú rezervované pre rozšírenia protokolu. Položka opcode definuje typ rámca a ako interpretovať užitočné dáta (payload). Typy rámcov s hodnotami opcode v hexadecimálnej reprezentácií:

- 0x0 - nadväzujúci dátový rámec obsahujúci dáta, ktoré nasledujú bezprostredne za predchádzajúcim rámcom v prípade fragmentácie.
- 0x1 - textový dátový rámec obsahujúci dáta v kódovaní UTF-8.
- 0x2 - binárny dátový rámec obsahujúci binárne dáta.
- 0x3-7 - hodnoty rezervované pre budúce dátové rámce.
- 0x8 - riadiaci rámec uzatvorenie WebSocket spojenia.
- 0x9 - riadiaci rámec PING slúži k overeniu živosti spojenia.
- 0x1 - riadiaci rámec PONG slúži ako reakcia na prijatie PING rámca.
- 0xB-F - hodnoty rezervované pre budúce riadiace rámce.

Ďalší bit MASK definuje, či sú užitočné dáta rámca maskované. Maskované sú všetky rámce zasielané od klienta a obsahujú 32 bitový kľúč v položke masking-key, ktorý server použije na

odmaskovanie dát. Server dáta nikdy nemaskuje. Takže rámec odoslaný zo servera masking-key neobsahuje.

Veľkosť užitočných dát (položka payload length) v bytoch je reprezentovaná pomocou 7, 7 + 16 alebo 7 + 64 bitov. V prípade, že hodnota na prvých siedmych bitoch je v rozsahu 0 až 125 táto hodnota určuje veľkosť užitočných dát. Hodnoty 126 a 127 určujú, že veľkosť reprezentuje nasledujúcich 16 resp. 64 bitov. Veľkosť dát je interpretovaná ako unsigned integer preto v prípade 64 bitov musí byť hodnota najviac významového bitu 0.

Tabuľka 3.1: Réžia WebSocket protokolu

Payload [Byte]	**Réžia[Byte]**	
	Klient–Server	**Server–Klient**
< 126	6	2
$< 2\hat{}16$	8	4
$< 2\hat{}63$	12	8

Tabuľka 3.1 zobrazuje réžiu WebSocket protokolu oproti TCP pri rôznej veľkosti prenášaných dát. Z dôvodu maskovania dát zo strany klienta je réžia väčšia práve o maskovací kľúč. Tabuľka ukazuje, že réžia je oproti prenášaným dátam minimálna, čo je veľkou výhodou WebSocket protokolu.

3.3 Zabezpečenie komunikácie

Ako už bolo spomenuté maskovanie nijako nezabezpečuje komunikáciu pomocou WebSocket protokolu. Pre zabezpečenie komunikácie je potrebné využitie WSS a teda tunelovanie cez SSL/TLS. TLS (Transport Layer Security) a jeho predchodca SSL (Secure Sockets Layer) sú šifrovacie protokoly používané na zabezpečenie sieťovej komunikácie. Pomocou kryptografie umožňujú aplikáciam komunikovať po sieti spôsobom, ktorý zabraňuje odpočúvaniu či falšovaniu správ a poskytujú koncovým bodom autentizáciu.

Pre ustavenie zabezpečeného spojenia je využitá asymetrická kryptografia a teda použitie dvojice kľúčov – verejného a súkromného. Klient zasiela požiadavku na vytvorenie zabezpečeného spojenia spolu s rôznymi parametrami spojenia. Server spolu s odpoveďou zasiela svoj certifikát, ktorý okrem iného obsahuje verejný kľúč. Zároveň si môže vyžiadať certifikát od klienta pre overenie jeho identity. Klient si overí identitu servera, vygeneruje základ šifrovacieho kľúča, ktorým sa bude šifrovať následná komunikáciu, zašifruje ho verejným kľúčom servera a pošle mu ho. Server použije svoj súkromný kľúč k rozšifrovaniu základu šifrovacieho kľúča a následne server aj klient z tohto základu a dohodnutých parametrov spojenia vygenerujú hlavný šifrovací kód. Nakoniec si navzájom overia a potvrdia, že komunikáciu bude šifrovaná týmto kľúčom a fáza handshake končí. Následná komunikácia je zabezpečená pomocou symetrickej kryptografie s využítím vytvoreného hlavného šifrovacieho kľúča [11].

Typicky, napr. pri komunikácií s webovými servermi je autentizovaný len server, zatiaľ čo klient zostáva neautentizovaný. To znamená, že totožnosť servera je overená a webový prehliadač si môže byť istý s kým komunikuje.

Pre zabezpečenie komunikácie medzi BeeeOn bránou a serverom je potrebná vzájomná autentizácia. Brána potrebuje mať zaručené, že komunikuje so správnym serverom a takisto je potrebná autorizácia brány a overenie jej identity z certifikátu.

3.4 Zhrnutie

WebSocket je aplikačný protokol využívajúci transportný protokol TCP. Poskytuje perzistentné full-duplex spojenie medzi klientom a serverom. Využíva privilegované porty HTTP a spojenie je vytvárané pomocou HTTP Upgrade požiadavky vďaka čomu dokáže prejsť firewallmi existujúcej sieťovej infraštruktúry. Bol vyvinutý primárne pre použitie v real-time webových aplikáciách, ale je vhodný aj na použitie v iných klient-server aplikáciách. Dáta sú prenášané formou správ (rámcov), ktoré môžu byť fragmentované.

Vďaka tomu, že WebSocket poskytuje zabezpečenú real-time komunikáciu, nízku réžiu prenášaných dát a nemá problémy s existujúcou sieťovou infraštruktúrou je vhodný na použitie v systéme BeeeOn pre komunikáciu medzi serverom a vzdialenými bránami v domácnostiach.

Kapitola 4

Návrh a implementácia

Táto kapitola popisuje návrh a implementáciu serverovej časti označenej ako GWServer (Gateway Server), ktorý nahradí pôvodný ADA Server a novú komunikáciu s BeeeOn bránami. Pre implementáciu je využitá knižnica Poco[1] z dôvodu hlbokej integrácie v celom projekte.

Súčasná implementácia ADA Serveru a komunikácia s BeeeOn bránami sa ukázala ako nedostačujúca. Dôvodmi sú problémy s nasadením v domácnostiach, kvôli blokovaniu komunikácie rôznymi firewallmi sieťovej infraštruktúry. Ďalej vytváranie veľkého množstva nových spojení s bránami pre každú správu prichádzajúcu od brány, zatiaľ čo jedno spojenie s bránou je stále perzistentné, ale slúži len na zasielanie riadiacich príkazov bráne. Ďalším dôvodom je nevyhovujúca architektúra pôvodného serveru, s čím súvisí nemožnosť jednoduchého pridávania ďalších potrebných správ. Keďže na správy zasielané zo serveru v podobe riadiach príkazov brána nijako neodpovedá nie je zaručená spoľahlivosť, a či boli riadiace príkazy skutočne vykonané.

4.1 Návrh komunikácie

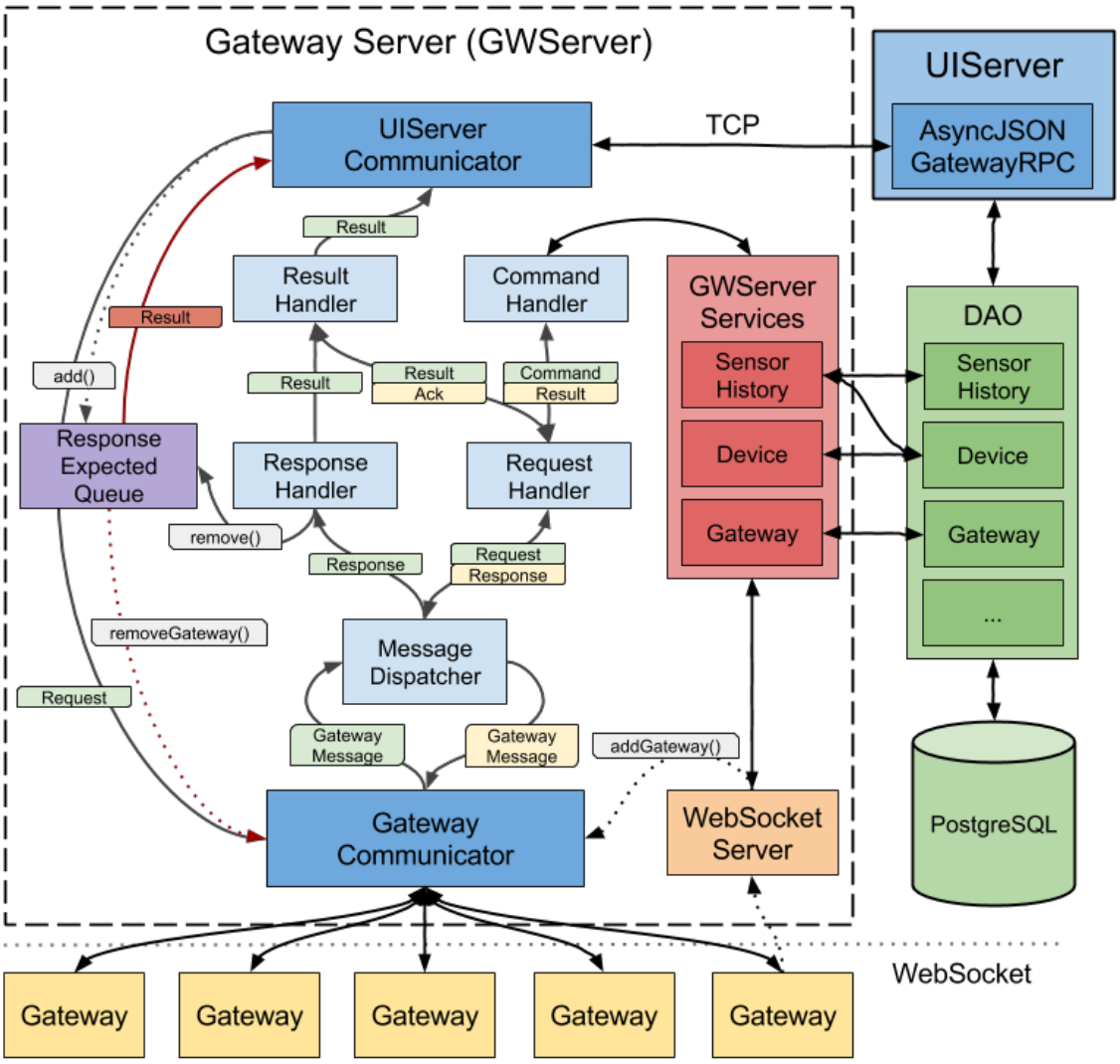
Pre komunikáciu medzi serverom a bránou som zvolil technológiu WebSocket, pretože umožňuje komunikovať i v sieťach s obmedzeným prístupom na privilegované porty. Z dôvodu šetrenia systémových zdrojov je vytváranie nových spojení pre každú správu nahradené jedným perzistentným spojením. Jednotlivé správy sú potom prenášané vo formáte JSON. Jedno perzistentné spojenie však prináša aj určité problémy. Keďže správy sú prenášané asynchrónne oboma, smermi nie je zaručené, že po zaslaní požiadavky príde odpoveď určená tejto požiadavke. Z tohto dôvodu som zaviedol unikátny identifikátor v každej správe, na základe ktorého je možné správne priradenie odpovede. Ďalším problémom je obsluha vysokého počtu perzistentných spojení a hlavne prijímanie dát z týchto spojení. Navrhnuté a implementované riešenie tohto problému popisuje kapitola 4.2.1.

4.2 Architektúra

Architektúra GWServeru zobrazená na obrázku 4.1 sa skladá z niekoľkých častí. Ohraničená časť označuje môj návrh a implementované riešenie. Využitý je spoločný dátový model spolu s dátovou vrstvou pre všetky časti serveru. Servisná vsrtva serveru bola rozšírená o služby potrebné pre registráciu brány na server, pridanie nového vzdialeného zariadenia a uloženie

[1] https://pocoproject.org/

senzorických dát. Taktiež bola navrhnutá a implementovaná komunikačná časť na druhej strane serveru v podobe triedy AsyncJSONGatewayRPC.



Obr. 4.1: Návrh architektúry Gateway Serveru

Aby mohla brána komunikovať so serverom je potrebné jej pripojenie a registrácia na server. Pre tieto účely slúži trieda WebSocketServer. Táto časť je zodpovedná za prijatie nového WebSocket spojenia a overenie identity brány z jej certifikátu. Následne je brána registrovaná využitím príslušnej služby a ak všetko prebehlo v poriadku spojenie je pridané do hlavnej časti pre komunikáciu s bránami GatewayCommunicator 4.2.1.

Trieda UIServerCommunicator slúži na prijatie pripojenia z druhej časti serveru a následnú komunikáciu s ňou. Prijíma príkazy, ktoré je potrebné poslať na bránu. V prípade, že je brána pripojená ich obalí do požiadavky pre bránu (request) a zasiela využitím komunikátoru s bránami. Ak brána pripojená nie je, oznámi tento stav UIServeru. Taktiež sú cez tento komunikátor s UIServerom odosielané odpovede od brán.

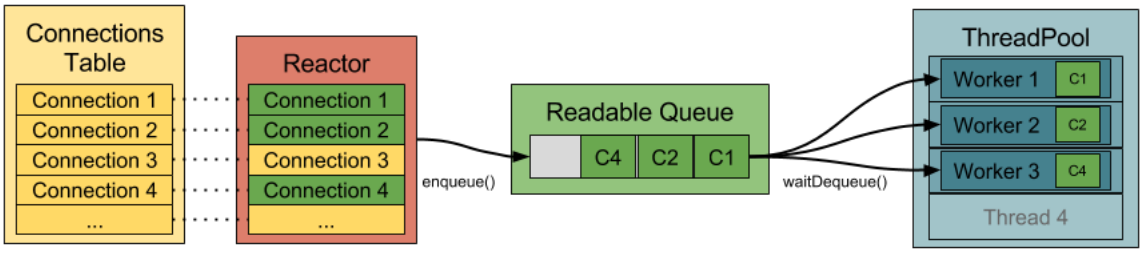
Na zistenie, že neprišla odpoveď od brány na nejakú požiadavku slúži fronta očakávaných odpovedí (ResponseExpectedQueue). Identifikátor každej požiadavky zasielanej na bránu, spolu s identifikátorom konkrétneho príkazu sa pridáva do tejto fronty. Fronta vy-

užíva časovač a v prípade, že odpoveď nedorazí do požadovaného časového limitu zasiela túto informáciu cez komunikátor s UIServerom. Brána je týmto považovaná za neaktívnu a spojenie je uzatvorené.

Správy medzi bránou a serverom majú 2 úrovne. Prvá úroveň je určená na priradenie odpovede (response) k požiadavke (request) v serverovej časti GWServer a detekciu živosti spojenia. Táto úroveň obaľuje konkrétny typ správy. Môže to byť príkaz (command), výsledok (result) alebo potvrdenie (ack). Z dôvodu, že príkazy zo serveru môžu mať asynchrónnu povahu, brána zasiela viacero odpovedí. Aby bolo zaručené spoľahlivé doručenie všetkých týchto odpovedí je potrebné ich potvrdzovať. Potvrdzovanie funguje na základe unikátneho identifikátoru v požiadavke a odpovedi. GWServer by preto musel duplikovať logiku nachádzajúcu sa na UIServeri a poznať presný počet odpovedí na každý príkaz. Preto v správe typu request nemusí byť zabalený len príkaz, ale taktiež výsledok vykonávania príkazu. V tomto prípade je bráne zasielaná správa typu response, ktorá obaľuje správu ack. Preto vznikla hierarchia postupného spracovania úrovní správ zobrazená na obrázku 4.1.

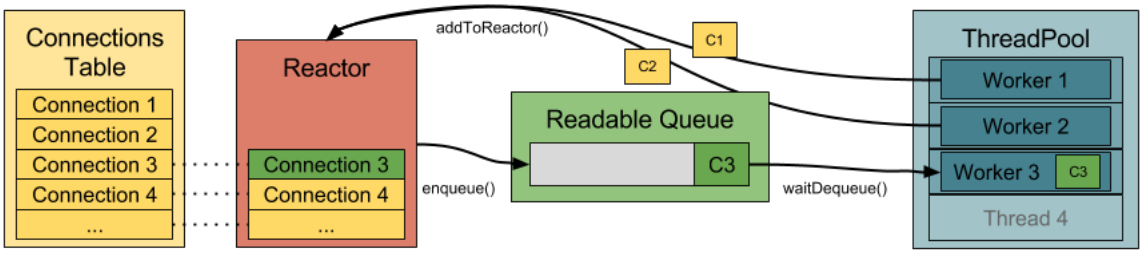
4.2.1 Trieda GatewayCommunicator

Trieda GatewayCommunicator je zodpovedná za komunikáciu s bránami po ich úspešnom pripojení a zaregistrovaní. V internej tabuľke si uchováva všetky spojenia s aktívnymi bránami. Ostatným častiam architektúry poskytuje metódy na pridanie, odobranie a vyhľadanie spojenia s konkrétnou bránou. Pomocou tohto spojenia môžu ostatné časti zasielať správy bráne, ale za prijímanie správ a následnú obsluhu je zodpovedná táto trieda. Okrem tabuľky spojení, obsahuje reaktor (konkrétne SocketReactor z knižnice POCO) bežiaci v samostatnom vlákne. Reaktor na pozadí využíva systémové volanie epoll na zistenie prichádzajúcich dát alebo uzavretie spojenia. Pre túto detekciu je potrebné spojenie zaregistrovať do reaktora. Čítanie dát v mojom riešení však neprebieha v jeho vlákne, pretože môže byť blokujúce, kým sa neprečíta celá prichádzajúca správa. Funguje v iteráciách a v každej iterácií vyvolá sériovo obsluhu spojení, z ktorých je možné čítať dáta. Tieto spojenia sú postupne zaraďované do fronty a odregistrované z reaktora. Poslednou časťou je konfigurovateľný threadpool, v ktorom sa podľa zátaže spúšťajú pracovné vlákna. Tieto pracovné vlákna si vyberajú spojenia s fronty obslúžia ich a spojenie zaregistrujú naspäť do reaktora. Nasledujúce obrázky 4.2, 4.3, 4.4 znázorňujú popísaný princíp obsluhy spojení a nadväzujú na seba.



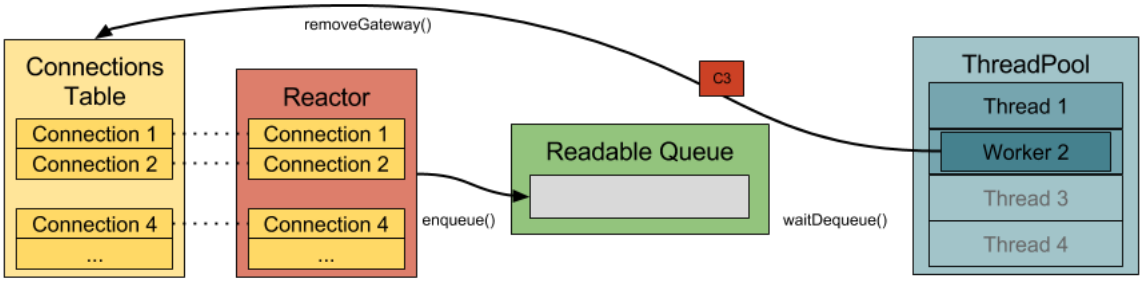
Obr. 4.2: Princíp fungovania triedy Gateway Communicator

Obrázok 4.2 znázorňuje, že reaktor detekoval možnosť čítania dát zo spojení 1, 2 a 3. Tieto spojenia zaradil do fronty, spustili sa pracovné vlákna a začali ich obsluhovať.



Obr. 4.3: Princíp fungovania triedy Gateway Communicator

Obrázok 4.3 znázorňuje stav, kedy pracovné vlákno 3 stihlo obsluhu oveľa skôr ako ostatné vlákna preto začalo obsluhovať ďalšie pripravené spojenie vo fronte. Pracovné vlákna 1 a 2 dokončili obsluhu a vracajú spojenia do reaktora.



Obr. 4.4: Princíp fungovania triedy Gateway Communicator

Posledný obrázok 4.4 znázorňuje, že pracovné vlákno 2 zistilo, že spojenia sa uzavrelo a preto ho nevracia do reaktora, ale odstráni záznam z tabuľky spojení. Okrem toho znázorňuje stav kedy bol server spustený s nastavením minimálneho počtu pripravených vlákien v threadpoole triedy GatewayCommunicator na 2 vlákna. V tomto prípade sa vlákno neukončí ale ostáva pripravené.

Kapitola 5

Záver

Cieľom bakalárskej práce bolo navrhnúť princíp komunikácie medzi serverom a vzdialenými senzormi a aktívnymi prvkami v systéme BeeeOn, ktorý zohľadňuje možnosť obsluhovať vysoký počet zariadení, spoľahlivosť doručovania dát, zabezpečenie prenášaných dát, množstvo rôznych meraných veličín, testovateľnosť a škálovateľnosť, a následne implementovať server využívajúci navrhnutý princíp komunikácie.

Prvá časť práce spočívala v zoznámení sa so systémom BeeeOn, s jeho jednotlivými časťami, ich prepojením, a s problematikou komunikácie so vzdialenými senzormi a aktívnymi prvkami v tomto systéme. Následne bolo potrebné naštudovať princípy zabezpečenej komunikácie pomocou technológie WebSocket, s možnosťou obsluhovať vysoký počet zariadení na serverovej strane. Naštudované poznatky sú zhrnuté v prvých dvoch kapitolách bakalárskej práce.

Na základe naštudovaných informácií som navrhol princíp komunikácie pomocou technológie WebSocket medzi serverom a bránami BeeeOn v domácnostiach, ktoré komunikujú so zariadeniami obsahujúcimi senzory a aktívne prvky. Následne som navrhol a implementoval server, využívajúci navrhnutý princíp komunikácie. Technológia WebSocket bola zvolená z dôvodu možnosti komunikovať na privilegovaných portoch, čím sa odstránili pôvodné problémy blokovania komunikácie rôznymi firewallmi v sieťovej infraštruktúre.

Schopnosť obsluhovať vysoký počet zariadení som dosiahol využítím systémového volania epoll v kombinácií s frontou spojení, z ktorých prichádzajú dáta a konfigurovateľným threadpoolom. Toto riešenie zaručuje tiež férovú obsluhu pripojených brán a pôvodné vytváranie nových spojení pre každú správu bolo možné nahradiť jedným perzistentným spojením. Z dôvodu obojsmernej asynchrónnej komunikácie, bolo potrebné zaviesť identifikáciu jednotlivých požiadaviek, aby k nim bolo možné priradiť správne odpovede. Pre zaručenie spoľahlivosti doručenia dát som implementoval systém potvrdzovania všetkých správ.

Keďže server BeeeOn je rozdelený na časť komunikujúcu s užívateľom (UIServer) a časť, ktorá bola úlohou tejto práce, komunikujúcu s bránami v domácnostiach (GWServer), navrhol a implementoval som nad rámec zadania aj komunikáciu medzi týmito časťami servera. Ďalej som nahradil vytváranie nových TCP spojení jedným perzistentným spojením s asynchrónnou komunikáciu. Oproti pôvodnej implementácií kedy na riadiace príkazy zasielané od užívateľa neprichádzali žiadne odpovede od brány, nová implementácia umožňuje viac úrovňové potvrdzovanie príkazov. Keďže niektoré príkazy (napr. nastavenie hodnoty aktívnemu prvku) majú asynchrónnu povahu, je možné serveru reportovať stav vykonania danej úlohy.

Najčastejšie možné scenáre výsledného riešenia komunikácie, som otestoval pomocou automatických a poloautomatických testov. Pre realizáciu testov bola vytvorená sada skriptov. S ich pomocou potom bola overená funkčnosť vytvorenej implementácie.

Literatúra

[1] *BeeeOn - Main Page*. [Online; navštívené 15.05.2017].
URL https://www.beeeon.org

[2] *BeeeOn - Repositories*. [Online; navštívené 15.05.2017].
URL https://www.github.com/BeeeOn

[3] *Jablotron*. [Online; navštívené 15.05.2017].
URL https://www.jablotron.com/cz/

[4] *Open ZWave*. [Online; navštívené 15.05.2017].
URL http://www.openzwave.com/

[5] *Philips Hue*. [Online; navštívené 15.05.2017].
URL http://www2.meethue.com/

[6] *Thermona*. [Online; navštívené 15.05.2017].
URL http://www.thermona.cz/

[7] BeeeOn: *Architektúra systému*. [Online; navštívené 15.05.2017].
URL https://antdev.fit.vutbr.cz/redmine/projects/iot/wiki/architektura

[8] BeeeOn: *Brána*. [Online; navštívené 15.05.2017].
URL https://antdev.fit.vutbr.cz/redmine/projects/adapter/wiki

[9] BeeeOn: *Brána*. [Online; navštívené 15.05.2017].
URL https://beeeon.org/wiki/Gateway

[10] BeeeOn: *Server*. [Online; navštívené 15.05.2017].
URL https://antdev.fit.vutbr.cz/redmine/projects/server/wiki

[11] Bouška, P.: *Protokol SSL/TLS - slabé šifry, zranitelnosti a jejich testování*. [Online; navštívené 15.05.2017].
URL http://www.samuraj-cz.com/clanek/protokol-ssl-tls-slabe-sifry-zranitelnosti-a-jejich-testovani/

[12] Fette, I.; Melnikov, A.: *RFC 6455 - The WebSocket Protocol*. ietf.org, Dec 2011, [Online; navštívené 15.05.2017].
URL http://tools.ietf.org/html/rfc6455

[13] Olimex: *A10–OLinuXino–LIME*. [Online; navštívené 15.05.2017].
URL https://www.olimex.com/wiki/A10-OLinuXino-LIME