VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 Martin Přistal

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

FRAMEWORK IOTIVITY A JEHO POUŽITÍ NA RŮZNÝCH HARDWAROVÝCH PLATFORMÁCH
IMPLEMENTATION OF IOTIVITY FRAMEWORK ON VARIOUS HARDWARE PLATFORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Přistal**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jiří Pokorný**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Martin Přistal ***ID:*** 173732
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Framework IoTivity a jeho použití na různých hardwarových platformách

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce bude proveden popis nově vznikajícího frameworku IoTivity, který je zaměřen na propojení komunikujících zařízení (drátových či bezdrátových) v ekosystému Internetu věcí. V praktické části bude provedena kompilace na zvolených architekturách (x86, ARM) s cílem vytvoření komunikačního scénáře mezi mobilním telefonem a výkonově omezeným zařízením.

DOPORUČENÁ LITERATURA:

[1] GUBBI, Jayavardhana, et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 2013, 29.7: 1645-1660.

[2] OUSTERHOUT, John K. Scripting: Higher level programming for the 21st century. Computer, 1998, 31.3: 23-30.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Jiří Pokorný
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá studiem projektu IoTivity. V práci jsou popsány metody komunikace pro různé druhy přenosů dat a architektury na niž je tento projekt vystavěn. Zmíněny budou i podobné projekty. Práce cílí na aplikaci v praxi a tedy i praktický výstup v podobě sestavení komunikace z různých prvků založených na odlišných architekturách v komunikačním scénáři Klient - Server.

KLÍČOVÁ SLOVA

Internet of Things, IoTivity, Rozhraní, Architektura, CoAP, DTLS, Zdroje, Klient-Server, URI, Wi-Fi, Ethernet, C, C++, Java, Python, Linux, Android, Intel Galileo , Raspberry Pi, Yocto

ABSTRACT

This thesis concerns with a research of project IoTivity. In this work are described methods of communication for a different ways of data transmission and an architecture, on which this project is based. In the next step thesis is aiming for usage in real life and practical output in the form of building communication from different elements based on different architectures in the Client - Server communication scenario.

KEYWORDS

Internet of Things, IoTivity, Framework, Architecture, CoAP, DTLS, Resources, Client-Server, URI, Wi-Fi, Ethernet, C, C++, Java, Python, Linux, Android, Intel Galileo, Raspberry Pi, Yocto

PŘISTAL, Martin *Framework IoTivity a jeho použití na různých hardwarových platformách*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Rok. 68 s. Vedoucí práce byl Ing. Jiří Pokorný

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Framework IoTivity a jeho použití na různých hardwarových platformách“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Jiřímu Pokornému, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

V současné době dochází k exponenciálnímu nárůstu počtu chytrých zařízení a automatizace všedního života. Komunikace představuje mezi jednotlivými zařízeními (často označovaná jako M2M (Machine-to-Machine) či D2D (Device-to-Device)) aktuální otázku v ekosystému Internetu věcí (IoT (Internet of Things)). Problémem je, že většina těchto zařízení pochází od různých distributorů a jejich hardwarové a softwarové základy se mnohdy i výrazně odlišují. Tato zařízení využívají ke svým komunikacím především prostředí Internetu, realizovaného zejména bezdrátovými technologiemi jako jsou Wi-Fi nebo LTE (Long Term Evolution). Jelikož na trhu působí různé společnosti využívající odlišných telekomunikačních technologií s řešeními jak hardwarově tak i softwarově rozličnými, vzniká tím stále větší potřeba pro softwarové sjednocení jejich vzájemné komunikace a interoperability. Řešení se nabízí v podobě open-sourcových projektů, které se snaží tuto otázku vzájemné komunikace vyřešit. Cílem těchto projektů je přenositelnost, čímž je myšleno, že jsou vytvářeny tak, aby jakékoliv zařízení nezávisle na použité architektuře čipu (SoC) a operačním systému bylo schopné provozu a propojení se s ostatními nezávislými zařízeními. V souladu s definovanými požadavky vznikly organizace zabývající se standartizací této problematiky jako OCF (Open Connectivity Foundation). Mezi tyto projekty se řadí i Iotivity, který bude v této práci prozkoumán. Bude otestována jeho přenositelnost pro různé SoC a operační systémy. Následně bude realizován komunikační scénář mezi těmito zařízeními. Výsledky a poznatky budou uvedeny v praktické části práce a shrnuty v závěru.

1 INTERNET OF THINGS

Internet of Things, v překladu Internet Věcí, dále jen IoT představuje koncept inteligentních bezdrátových zařízení schopných sběru dat a jejich předání skrze Internet, kde se dále zpracovávají. k inteligetnímu řízení přispívá implementace embedded zařízení. k vzájemné komunikaci mezi sebou využívají především komunikační technologie Ethernet, Wi-Fi, Bluetooth, LTE (Long Term Evolution) atd. Zařízení nefungují pouze jako M2M, nýbrž tvoří především větší funkční celky. Jejich přítomnost si kolikrát neuvědomujeme, avšak jejich nasazení vede k enormní automatizaci prostředí kolem nás. IoT je často vnímáno z pohledu výkonově omezených zařízení z hlediska jejich energetického napájení (životnost baterií) a dalších minimalizací těchto nároků. IoT uvádí pojmy zařízení a věc, kde věcí je myšlen fyzický objekt umožnující připojení k internetu a zařízení by mělo být schopné komunikovat s ostatními zařízeními nebo možnosti samotného sběru dat a optimálně jejich dalším zpracováním.

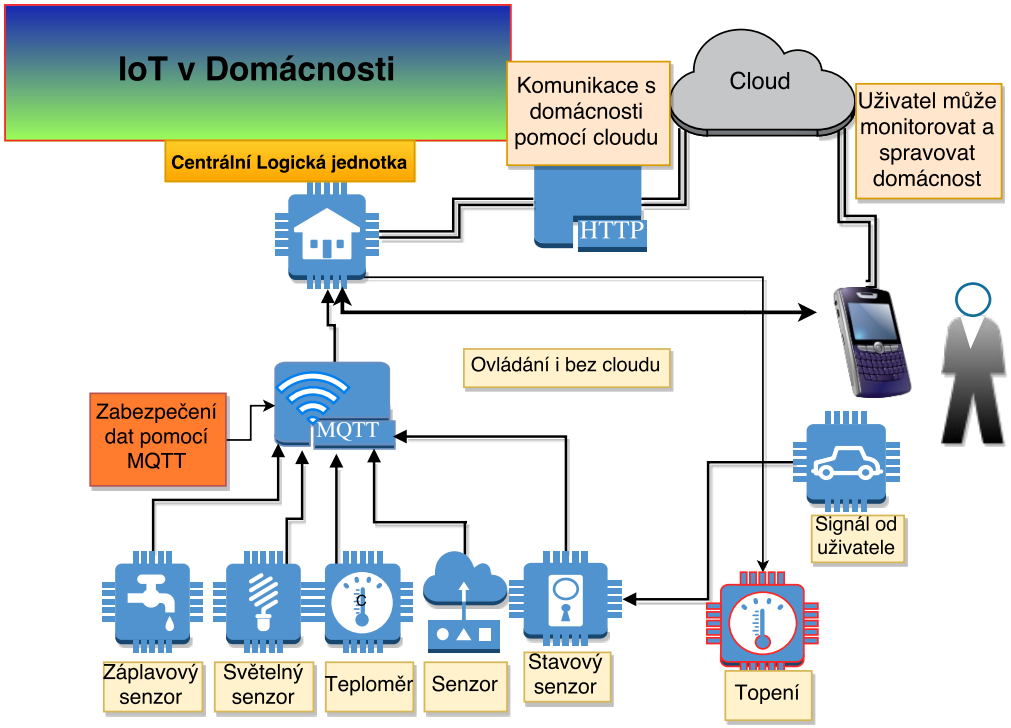
1.1 Využití IoT

IoT působí ve velkém měřítku a imlementace stále proniká do mnohých sfér užití. Tyto oblasti mohou mít komerční využití, zastoupení v průmyslu, či mohou být pro neziskové vývojáře v rámci testování vlastních prostředků. Hlavní zastoupení IoT můžeme nalézt v průmyslu, tedy automatickém řízení výroby a monitoringu, bezpečnostních firmách, domácnostech, zdravotnictví, automobilovém průmyslu – například čidla parkování, rozvíjející se komunikace automobilů navzájem v provozu a další oblasti.

1.1.1 Řízení chytré domácnosti

Nejjednodušší pro úvod a pochopení základní logiky, bude reprezentace kolekce a použití získaných dat v domácnosti(obr.1.1). Data jsou snímána senzory, které komunikují skrze vzdušné médium, reprezentované zde pomocí technologie Wi-Fi, s centrální logickou jednotkou monitorující provoz systému. Jednotka má přehled o všech zařízeních k ní připojených. Získaná data zpracovává a následně předává i mimo místní síť uživateli dle preferencí. Reprezentuje tak komunikační střed mezi uživatelem a budovním systémem. Získaná data mohou být samostatně vyhodnocena na základě vnitřího programu. Například udržení teploty v domácnosti lze realizovat za pomoci klimatizace, topení či tepelně izolačních žaluzií. Tedy přijme informaci o teplotě a na základě této teploty může vyslat signál pro aktivaci topení, nebo jeho neaktivitu. Uživatel si může zvolit, ve kterou hodinu se topení zapne, či se pozastaví

jeho aktivita z jiných důvodů. Tato správa v domácnosti slouží nejen k monitorování aktuálního stavu domácnosti, nýbrž má za cíl především snížení energetických nákladů objektu, a tím i ekonomického dopadu.



Obr. 1.1: Chytrá domácnost

1.1.2 Senzory

Užitečná data, které mohou být získána jsou například: **teplota, vlhkost, úroveň osvětlení, energetický odběr, tlak** - v zabezpečení snímání tlaku tíhou předmětu, **stavové** - například světla zapnuto/vypnuto, dveře otevřeny/zavřeny, **záplavové** - snímají pomocí spínacích kontaktů, dalším senzorem může být senzor pro otevírání vjezdové brány z dálkového ovladače. Senzory představují nutný článek, který získává data potřebná pro další zprácování a vyhodnocení. Bez senzorů by celá architektura nejen v případě chytré domácnosti ztrácela na významu a postrádala logickou realizaci celého řízení. Senzory jsou nejčastěji výkonově omezená zařízení.

1.2 Zabezpečení

Kritickým parametrem je zabezpečení předávání dat, před zneužitím třetí stranou. Jelikož počet IoT zařízení neustále vzrůstá a jejich použití je stále širší, jsou tak uživatelé vystavování nebezpečí úniku osobních dat. Předcházet lze užitím šifrované komunikace, či jiných metod zabezpečení přenosů, podle zvoleného protokolu UDP / TCP. Každý má své vlastní bezpečnostní metody přenosu. Na základě této problematiky se různé projekty liší užitím konkrétních komunikačních protokolů a metod, ze kterých vyplývá i jejich následná možnost realizace zabezpečení.

Autentizace U zařízení dochází zprvu k autentizaci, tedy procesu podobnému jako elektronický podpis. Tím se zajistí prvotní spojení.

Heslo Další bezpečnostní úskalí je nedostatečná znalost problematiky IT uživatelem a zařízení je tak ponecháno původní heslo.

Kryptografie Užití šifrování dat v přenosu pomocí kryptografických metod.

Tunel Vytvoření bezpečného spojení mezi zařízení např. SSH(Secure Shell),SSL (Secure Sockets Layer)

Firewall Užitím firewallu(systém zabezpečení) na komunikačním rozhraní sítě.

Protokoly Nasazení komunikačních protokolů, jenž mají v sobě implementováné řešení ochran (2.3.2).[24]

1.3 Embedded systémy

Neboli vestavěné systémy jsou založeny na mikroprocesorech nebo mikrokontrolerech. v obou případech je přítomen integrovaný obvod, navržený pro operace běžící v realném čase. Mikrokontroléry neobsahují pouze výpočetní jednotku - CPU (Central Processing Unit), ale také RAM (Random Acces Memmory) paměti, Flash paměti a různé I/O porty. Využívají se pro více komplexní úkoly. Vyšším stupněm realizace mikrokontrolérů je označení SoC (System on the Chip) - systém na čipu. Co se týká obdoby operačního systému, ten je minimalizován a nahrán do pamětí Flash nebo ROM (Read-Only Memory). Tyto operační systémy jsou upraveny přímo pro embedded zařízení a jejich velikost zabírající v paměti je minimální (od stovek kB). Nese označení Firmware. Systém může být nahrán na jednoduchém modulu pro sběr dat, až po technologie chytrých telefonů [25].

1.4 Yocto Project

Open-source projekt, který nabízí šablony, nástroje a metody s cílem vytvořit systém s linuxovým základem pro embedded zařízení nezávisle na hardwarové architektuře. Výhodou je, že neobsahuje žádné přebytečné části kódu, které nejsou potřebné k vystavění systému pro dané zařízení. Zaručuje spolehlivost a jednoduchost implementace linuxového systému pro cílová zařízení [9].

1.5 Brillo

Představuje IoT platformu embedded zařízení se systémem Anrdoid. Cílí na paměťově a zdrojově omezená zařízení. Umožňuje jednoduchý vývoj aplikací. Implementuje protokoly Weave (API pro IoT) a Thread (síťový protokol). Brillo může být vystavěno ze zdrojového kódu pro architektury ARM (Acorn RISC Machine), Intel a MIPS (Microprocessor without Interlocked Pipeline Stages). Systém potřebuje k běhu minimálně 128MB paměti statické a 32MB paměti dynamické (RAM). Projekt nabízí i vývojové desky, pro které existují balíčky softwaru dle cíleného použití. Google spolupracuje s HW vývojáři, aby jejich desky byly kompatibilní s Brillem a ulehčili developerům implementaci jejich softwaru [15].

1.6 OSGI

Open Services Gateway Initiative je světová asociace technologických inovátorů, která cílí na vytvoření otevřených specifikací, které umožní standartní zapojení softwaru s technologií Java (objektově orientovaný programovací jazyk). Specifikace popisují modulární systém a servisní platformu pro jazyk Java, jenž implementuje kompletní a dynamický model komponent. Aplikace nebo komponenty jsou vydávány v balíčcích pro rozvíjení, a mohou být vzdáleně instalovány, spuštěny, zastaveny, odinstalovány, rozšířeny bez nutnosti resetování systému.[20]

1.7 Alljoyn

Jedná se o softwarový open-source projekt skupiny AllSeen Alliance, který umožňuje vývoj aplikací pro objevení zařízení a jejich komunikaci mezi sebou. Cílí na jednoduchou implementaci pro IoT zařízení, nezávisle na jediném operačním systému nebo architektuře. Používá TCP (Transmission control protocol) i UDP (User Datagram Protocol), avšak pouze ve verzi IPv4, IPv6 je ve vývoji. Zabezpečení komunikace probíhá na aplikační vrstvě. Pakety jsou šifrovány až aplikační vrstvou. Komunikace

probíhá v binárním formátu, přes D-BUS serializaci. Alljoyn využívá RMI (Remote Method Invocation) architekturu s subscribe / publish mechanismem (komunikace inicializovaná el. podpisy). Dodržuje OIC standardy [2].

1.8 IoTivity

Podobným projektem jako Alljoyn je IoTivity. Projekt se stejnými cíly jako Alljoyn a je vytvářen, (dnes již nezávislý) s pomocí OCF (Open Connectivity Foundation) organizace, jež určuje standardy komunikace IoT. Použité technologie a metody se odlišují u každého projektu (viz.2.7.4).

Parametr	IoTivity	Alljoyn
Architektura	RESTful+Notify	RMI+Pub/Sub
Datový formát	JSON, přes CBOR	Binární řízení přes D-BUS, XML
Transport	UDP (IPv4 a IPv6),multicast	UDP a TCP (IPv4)
Objevení zařízení	CoAP	MDNS
Zabezpečení	ECC, AES, X509	ECC, AES, X509
	DTLS - Linková vrstva	ACL - Aplikační vrstva

Tab. 1.1: Přehled základních rozdílů projektů IoTivity a Alljoyn [2]

2 IOTIVITY

Jedná se o open-sourceový projekt obsahující kompletní sadu nástrojů umožňující vývojáři implementaci jeho softwaru do různých zařízení pomocí dostupných knihoven. IoTivity je podporováno organizací Linux Foundation. Vývoj probíhá v souladu standardů OIC (Open Interconnect Consortium). IoTivity je projekt, který cílí na univerzální rozhraní zařízení, aby byly splněny základní požadavky v komunikaci zařízení-zařízení. Dále zahrnuje API knihovny pro různé systémy a zařízení, čímž umožňuje vývojářům snadnější implementaci jejich kódů a usnadnit jim jejich budoucí práci. Iotivity obsahuje podpory pro různé SoC architektury i operační systémy, a pomocí těchto předpřipravených platforem, je pro vývojáře propojení jeho softwaru s IoTivity rozhraním, které zajišťuje komunikaci mezi zařízeními, velmi odlehčující. IoTivity je především strukturován na vrstvě aplikační [2].

2.1 Rozhraní

Rozhraní neboli framework reprezentuje mechanismy pro vývoj aplikací komunikujících přímo s hardwarem. Na této úrovni se inicializují a ladí vlastnosti komunikace zahrnující podklady vývoje aplikací. Rozhraní zahrnuje řadu knihoven. v IoTivity tvoří rozhraní základní sadu nástrojů a vrstvu pro implementaci OCF standardů pro komunikaci. Psané jsou převším v programovacím jazyku C a C++ [10].

2.1.1 Rozhraní v IoTivity

Hlavní mechanizmy rozhraní v IoTivity.

1. **Identifikace zařízení:** Podporuje násobné mechanismy pro objevení zařízení a zdrojů a to jak vzdáleně tak přímo.
2. **Správa nad zařízením:** Zařizuje konfiguraci, diagnostiku a opatření zařízení.
3. **Předávání dat**: Výměna informací a kontrola na základě streamovacích a informačních modelů.
4. **Správa nad daty:** Spravuje kolekci dat, jejich uložení a analýzů nezávisle na zdroji informace [2].

2.2 Architektura

Architektura reprezentuje vystavění systému, respektivě uložení jednotlivých komponent a jejich vrstvení na sebe. Především se jedná o vlastnosti a nastavení vzájemné interoperability. v projektu IoTivity se řeší architektura programové implementace, která se stará o řízení a vystavění nastavení všech nutných komunikačních

protokolů pro správnou funkci komunikace [1]. Pokud mluvíme o architektuře operačního systému, mluvíme o způsobu komunikace systému s procesorem. Procesory se liší ve velikostech pamětí, registrů, šířkách sběrnic a velikostí datových typů, do kterých se uládají výsledky. Menší procesory 8/16-bitové pro méně náročná zařízení jako kalkulačky, ale především 32/64-bitové použitých od chytrých telefonů po desktopové počítače. Vždy tedy mluvíme o způsobu utilizace rozhraní a jeho spuštění. Projekt IoTivity je založen na REST komunikaci .

2.2.1 Representational State Transfer

Zkráceně REST je typem komunikace pro distribuované systémy především World Wide Web, fungující na základě HTTP (Hypertext Transfer Protocol). Pracuje se zdroji, které nesou formát typu URL (Uniform Resource Locator). V RESTu jsou přenášeny i přidané datové vrsty HTTP cookies (uložená data na straně klienta od serveru). Datově orientovaný, podle identifikátoru URI (Uniform Resource Identifier). Vhodný pro jednoduchou práci server-klient. RESTful je označení, které značí užití architektury REST. REST architektura se orientuje na zdroje, což znamená, že je založena na resource konceptu (zdrojů). Jakýkoliv objekt v kategorii architektury REST uložený na síti, který lze v rámci systému identifikovat je považován za zdroj. Základní myšlenkou je, že se nezabývá tolik způsobem, jak data měnit a zpracovávat. Zabývá se spíše jejich strukturou, identifikací a reprezentací. Operace prováděny nad daty jsou považovány vždy za standartní. Komunikace v této architektuře probíhá vždy mezi klientem a serverem. Vložení dalších vrstev mezi ně musí zůstat realizovatelné, mezilehlé servery a brány jsou na sobě nezávislé a pro ostatní vrstvy neviditelné. Tyto neviditelné vrstvy můžou být znázorněny jako vrstvy ukládající průchozí data do mezipaměti (cache) a snižují tak odezvu celé trasy [22].

2.2.2 JavaScript Object Notation

Zápis JavaScriptovou objektovou notací, zkráceně JSON. Zápis sledující syntaxi Javascriptu objektů, který je jazykově nezávislý. Jedná se o textový formát pro výměnu dat, nejčastěji při komunikaci prohlížeč-server. Data jsou zapisovány formou čitelnou pro člověka, díky tomu se velmi popularizuje. Může být implementován do komunikačních protokolů jako je MQTT, CoAP a další [26].

2.2.3 SDK

Software development kit - sada vývojových nástrojů případně i úplné prostředí, ve kterém je možno vyvíjet aplikace. Obsahuje API knihovny, pro práci mezi jednotlivými operačními systémy.

2.2.4 Concise Binary Object Representation

CBOR - Datový formát založen podle formátu JSON. Oproti JSONu může popsat komplexnější struktury dat. Podporuje také možnost streamovat. Mezi jeho cíle patří možnost extrémně malé velikosti kódu, malé zprávy a rozšiřitelnosti bez potřeby závislosti na verzi. To činí CBOR odlišným od předchozích binárních serializací (ASN.1 and MessagePack). Jedná se o standartizovaný formát binární reprezentace strukturovaných dat. Některé z nich specificky pro hlavní informace, zatímco ostatní jsou libovolná data. Prvky, které CBOR musí splnit jsou

1. Reprezentace musí být schopna jednoznačně zakódovat většinu běžných datových formátů použitých v Internetových standardech.
2. Kodéry pro kodéry nebo dekodéry musí být kompaktní souladně k podpoře systémů s velmi omezenou pamětí, výkonem procesoru a sadou instrukcí.
3. Data musí byt schopna být dekódování bez popisného schématu.
4. Serializace musí být smysluplně kompaktní, ale datová kompaktnost je druhořadá ke kódové kompaktnosti pro kodér a dekodér.
5. Formát musí být aplikovatelný na oba omezené uzly a vysoce úrovňové aplikace.
6. Formát musí podporovat všechny typy JSONu pro konverzi do i z JSONu.
7. Formát musí být rozšiřitelný. Rozšířená data musí být dekódovatelná dřívějším dekodérem [3].

2.2.5 RAML

RESTful API Modeling Language je pro lidi a stroje čitelný jazyk podle definice RESTful aplikačního programového rozhraní. RAML byl navržen ke zlepšení specifikací rozhraní poskytnutím formátu, který může sloužit jako jako mezičlánek mezi API poskytovatele a klienta. Například k demostraci při poskytování uživatelské dokumenace a zdrojového kódu pro klientské a serverové implementace, čímž se zrychluje a zlepšuje definice a vývoj vnitřních aplikací, které užívají RESTful API. RAML není prozatím striktní. Soustředí se čistě na popis zdrojů, metod, parametrů, odpovědí, typy médií a další HTTP struktury, které formují základ pro moderní API, které splňují mnohé RESTful požadavky [27].

2.3 Protokoly

Přenos dat je podporován násobnými protokoly, což znamená, že může využívat různé komunikační protokoly. Dále pomocí API se můžou pod něj navazovat další protokoly dle potřeby [1]. Protokol v obecném znění reprezentuje speciální seznam

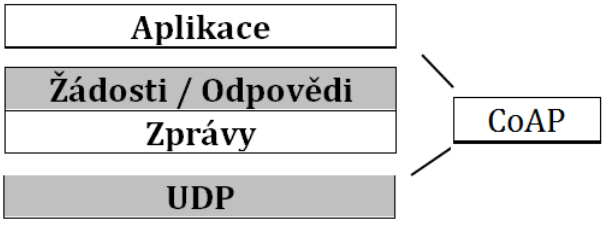
pravidel, které používají koncové body v telekomunikačním spojení při vzájemné komunikaci. Protokoly specifikují pravidla interakce mezi komunikujícími entitami. Protokoly existují na různých úrovních spojení. Například, existují protokoly pro výměnu dat na hardwarové úrovni a jiné na úrovni aplikační. Standartním modelem komunikace po internetu známým jako *Open Systems Interconnection – OSI*, kde jsou na každé vrstvě jeden nebo více protokolů pro realizování úspěšné telekomunikační výměny. Protokoly jsou popsány podle mezinárodních standardů [5].

2.3.1 Constrained Application Protocol

Zkráceně CoAP je webově orientovaný protokol aplikační vrstvy založeném na architektuře REST, určený pro vázané uzly a výkonově omezená zařízení. Uzly většinou mívají 8-bitové mikrokontroléry s malým množstvím ROM a RAM paměti, zatímco vázané sítě jako IPv6 a Low-Power WPAN –*6LoWPANs* často mají vysokou chybovost paketů a typickou propustnost řádech *10kbps*. Protokol je navržený pro M2M aplikace jako chytré energetické a budovní automatizace. v současné době využívá CoAP definovaný podle IETF (Internet Engineering Task Force) jako hlavní komunikační protokol IoT. CoAP lze přirovnat k odlehčenému hypertextovému přenosovému protokolu (HTTP), kde model interakce je podobný modelu HTTP klient-server. Oproti HTTP, CoAP využívá ke komunikaci datagramově orientovaného transportního protokolu UDP. CoAP je považován za koncept dvouvrstvého přístupu obr.2.1.

1. Sdělovací vrstva vytvořena pro kooperaci přes UDP. Charakterizuje se asynchronní povahou interakcí.
2. Vrstva pro interakci požadavek/odpověď použitím metod a reakčních kódů.

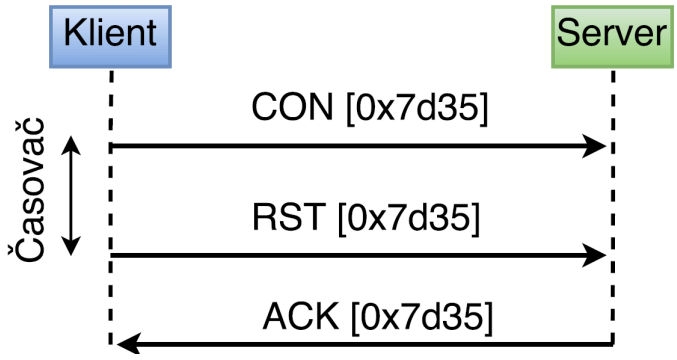
Nicméně CoAP je jednotný protokol a žádost/odpověd je pouze vlastností CoAP hlavičky.



Obr. 2.1: Abstrakce vrstev CoAP [16]

Dále skrze tento protokol se zařízení mezi sebou objevují a není potřeba dalšího protokolu s *URI* identifikátory. Pracuje na základě modelu žádost/odpověď mezi

koncovými aplikacemi. Podporuje vestavěné objevovací služby a zdroje. Žádost je odeslána klientem. Server odpoví kódem odpovědi. Tato odpověď může obsahovat zdrojovou reprezentaci. CoAP používá hlavičku zkrácené fixní délky *(4 Bajty)*, která může být doplněna kompaktními binárními možnostmi a užitečnými daty. Každá zpráva obsahuje *ID Zprávy-16 bitů* pro detekování duplikátů a pro volitelnou spolehlivost. Ta je zaručena zavedením označováním zpráv jako *Potvrditelné-CON*, kde tento mechanismus je označován jako *Piggybacked* reakce. Tento typ zprávy je odesílán opakovaně na základě interně přednastaveného časovače a odstupné funkce pro nastavení na původní hodnotu, dokud příjemce nepotvrdí zprávou pro *Potvrzení-ACK* se stejným *ID*, jako na obrázku s příkladem (0x7d35) obr.2.2, jinak odpoví zprávou o restartování *RST*. Zprávy mohou mít kromě charakteru *CON* i možnost *Nepotvrzující-NON*, ty nevyžadují potvrzení o doručení. Příkladem může být nedoručená informace teplotního senzoru v domácnosti.



Obr. 2.2: Zprávy ACK-RST-CON [16]

Sémantika CoAP žádodostí/odpovědí jsou přenášeny s CoAP zprávami, které zahrnují rovněž *Metodický kód* nebo *Response Code*(druh odpovědi na základě vyžádané operaci, pouze informativní). Volitelné nebo spíše přednastavené informace žádosti a odpovědi, jako jsou *URI* a typy datových médií, jsou přenášeny jako vlastnosti CoAPu. Pro rozlišení konkrétní odpovědi na konkrétní žádost jsou používány *Tokeny* - neboli značky *(*není to samé co ID zprávy).

Obsahuje základní 4 metody: GET, PUT, POST a DELETE, kterými přistupuje ke zdrojům. Existují i další metody s odlišnými specifikacemi. Tyto metody nemusí nezbytně používat logiku žádost/odpověď párovaně. i jedna žádost může vyvolat mnohonásobné odpovědi (pro žádost o multicast nebo možnost *OBSERVE*). Podpora URI na straně serveru je zjednodušena, protože klient již rozloží URI a rozděluje ji do: hostu, portu, cesty a query. Využívá implicitní hodnoty kvůli efektivitě. Zařízení mezi sebou komunikují pomocí metodiky zdrojů. Objevování pomocí zdrojů je důležité pro M2M interakce.

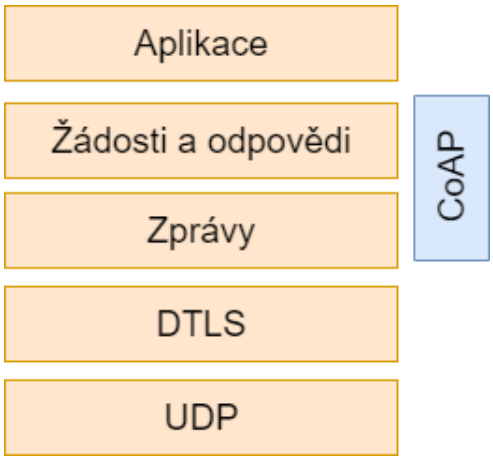
- **GET** metoda vrací reprezentaci informace, která aktualně korespondje zdroji identifikovanému dle žádosti URI. Pokud žádost obsahuje *Accept Option*, která indikuje preferovaný formát odpovědi. Tato metoda je bezpečná a idempotentní. Odpovědný kód (Response) by měl byt zahrnut s odpovědí
- **POST** metoda vyžaduje zpracování dle přiložené reprezentace v žádosti. Základní funkce této metody je určena původem serveru a závislá na cílovém zdroji. Běžně nastává situace vytvoření nebo aktualizování cíleného zdroje. Pokud byl dán zdroji vznik na straně serveru, vrácená odpověď serverem by měla (kontroluje data z originálního datagramu) mít Odpovědný Kód a měla by zahrnovat URI nového zdroje v sekvenci jedné či více lokačních cest nebo dotazových možností. Nejedná se o metodu bezpečnou ani idempotentní.
- **PUT** metoda, kde zdroj je identifikován žádostí URI o aktualizování nebo vytvoření s přiloženou reprezentací. Její formát je specifikován typem média a obsahem kódování. v případě, že zdroj existuje pro danou URI, Přiložená reprezentace by měla být považována za upravenou verzi tohoto zdroje a kód odpovědi by měl být vrácen (změněn). Pokud žádný zdroj neexistuje, potom server může vytvořit nový zdroj s touto URI, utvořen do odpovědního kódu. Pokud zdroj nemohl být vytvořen nebo modifikován, potom odpoví pomocí *error* odpovědním kódem. Metoda je idempotentní, ale ne bezpečná.
- **DELETE** metoda požaduje, aby server smazal zdroj dle *URI*. Tato metoda je idempotentní, ale ne bezpečná.

MULTICAST

CoAP podporuje vytváření žádostí IP multicastových skupin. Pro komunikaci přes multicast musí všechna zařízení naslouchat na přednastaveném portu pro CoAP (5663-5664 pro DTLS) a zároveň musí být připojeny k jednomu, či více náležitým CoAP - uzlovým multicastovým adresám. Namísto adresy koncových bodů se odesílá na multicastovou adresu *0.0.0.0*. Jedná se o proces, kdy se jedná o komunikaci nespolehlivou *(NON-ACK)*. Komunikace může být vedena pouze skrze UDP nikoliv přes DTLS [16].

2.3.2 DTLS

Datagram Transport Layer Security je protokol zajištující ochrany komunikace protokolu CoAP přes UDP. Rozšiřuje protokol TLS (použit v TCP) při použití datagramu. Nedochází ke zpoždění, ale jsou možné ztráty paketů a jejich nové vyžádání. Působí na linkové vrstvě, kde je nutné při přechodech mezi různými linkovými vrstvami zajistit prověřený vztah mezi těmito dvěma různými body. Využívá zabezpečovací algoritmy ECC, AES a X509 [2].



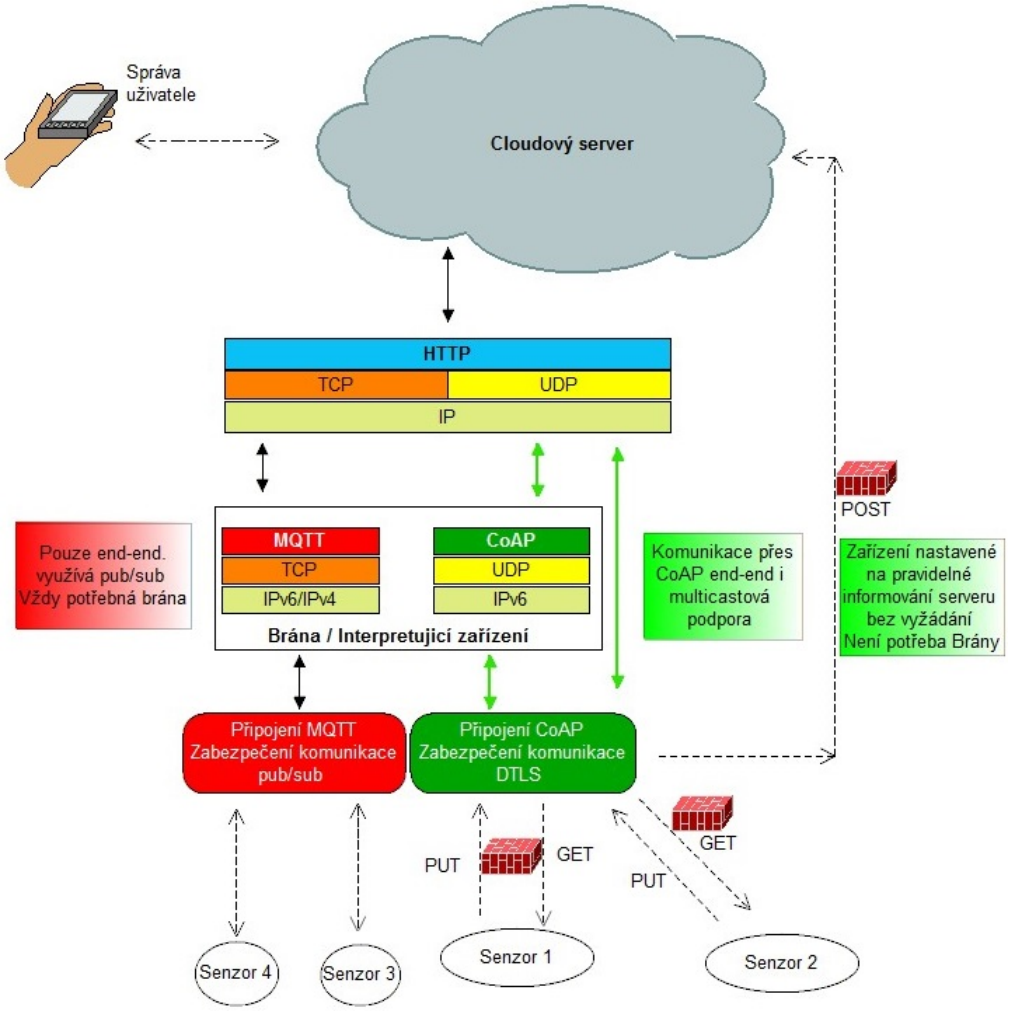
Obr. 2.3: Abstraktní vrstvení DTLS s CoAP, převzato z [16]

2.3.3 MQTT

IoTivity je připraveno používat MQTT neboli *Message Queue Telemetry Transport*, což je další nenáročný komunikační protokol mezi dvěma zařízeními. Využívá metody publisher/subscriber, ještě před započetím samotného přenosu užitečných dat. k přenosu je potřebný MQTT zprostředkovatel, jenž řídí přenos informací z více klientů přes centrálního zprostředkovatele. Využívá komunikační protokol TCP. MQTT umožňuje šifrování pomocí TLS/SSL (Transport Layer Security / Secure Sockets Layer) metod, avšak v aplikaci pro IoT probíhá komunikace pouze mezi dvěma zařízeními na základě publisher/subscriber a jiné zabezpečení nevyužívá. Vhodný na vzdálené propojení, kde je zapotřebí omezit množství přenášených dat. Jedná se o extrémně jednoduchý protokol určený pro omezená zařízení s úzkým pásmem nebo ve vysoko-latenčních nespolehlivých sítí. Rozdíl komunikací protokolů MQTT a CoAP (viz Obrázek 2.4) v IoT je rozlišný v zabezpečení a typu doručení. Nevýhodou užití TCP je udržování komunikace s klientem, což je kritické pokud zařízení disponuje omezeným energetickým zdrojem - baterií [17].

2.3.4 UDP

User Datagram Protokol, nezaručuje doručení datagramu pomocí *ACK* paketu jako u TCP. UDP je vhodný díky svému jednosměrnému vysílání pro přenos v reálném čase. Poskytuje rozhraní mezi aplikační a síťovou vrstvou. Protokol však nezaručuje správné pořadí při doručení. v případě přehlcení směrovače jsou nadbytečné UDP datagramy zahazovány, pokud však je jejich doručení nutné, je nasazen DCCP protokol, který je pro případy zahlcení, kde koriguje provoz a zacházení s datagramy,



Obr. 2.4: Komunikace CoAP [16] vs MQTT [**?**]v IoT převzato z [14]

anebo si druhá strana vyžádá o poslání stejného datagramu znovu. V projektu IoTivity využívá internetový protokol IPv4 i IPv6. Dále je možné pracovat v multicast režimu.

2.3.5 TCP

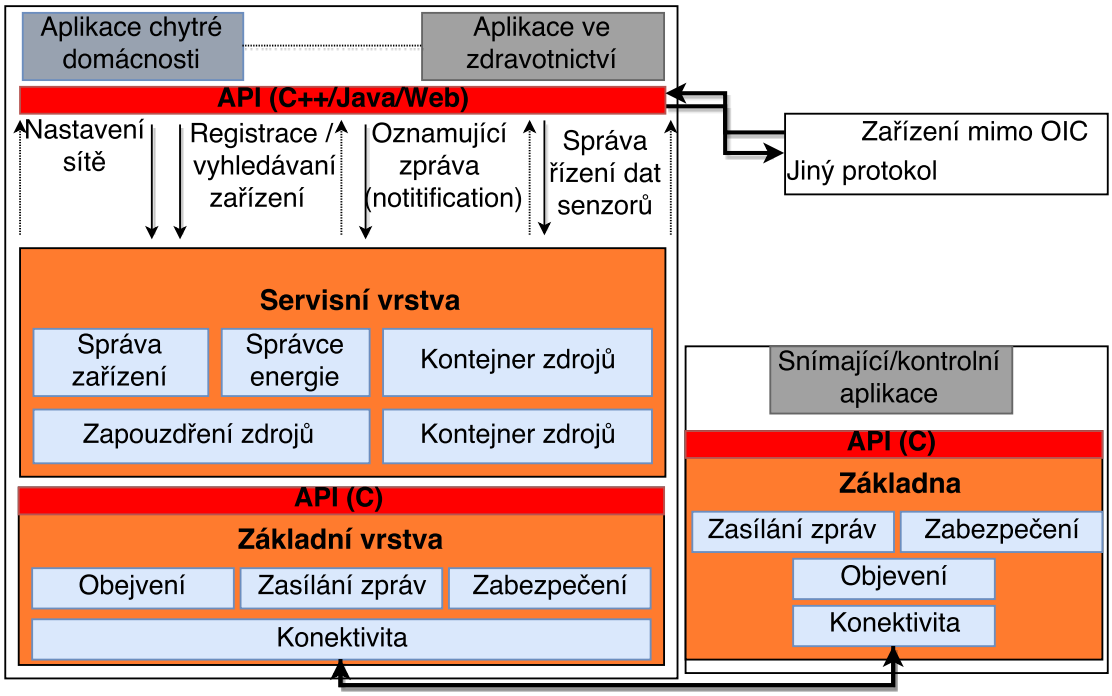
Transmission Control Protocol je spojově orientovaný protokol. Jedná se o protokol transportní vrstvy zajištující obousměrnou komunikaci. Zajištuje doručení datagramu ve správném pořadí a jeho neztrátovost. Mezi klientem a servem proběhne three-way handshake. v průběhu navazování spojení se obě strany dohodnou na číslu sekvence a potvrzovacím čísle. Pro navázání spojení se odesílají datagramy s nastavenými příznaky *SYN / ACK*. Protokol je náročnější na datovou režii než UDP.

2.4 Open Connectivity Foundation

Open Connectivity Foundation (OCF) je organizace, která si klade za cíl vyvinout standardy a certifikace pro IoT pracující na komunikačním protokolu CoAP. Její vliv je v globálním měřítku a spolupracuje s velkým množstvím firem jako jsou Cisco, MediaTek, Samsung, Intel, Microsoft, Qualcomm atd. Definuje jednotné komunikační rozhraní, který připojuje a inteligentně zpravuje tok dat mezi zařízeními v IoT. Zakládá na interoperabilitě open-source systému - Tizen, Linux a Android. Vytváří specifikace, které popisují svět v IoT, certifikační programy včetně zabezpečení a sponzoruje open-source projekty jako je např. IoTivity. Licencovaný pod Apache License verzí 2. Architektura je založena na vyhledávání zdrojů a zde využívá RAML pro specifikaci zdrojů s pomocí knihoven a protokolu JSON pro předávání obsahu dat [19].

2.5 Application Programming Interface

API neboli Rozhraní pro programování aplikací, respektive soubor knihoven, který může být použit na úrovni softwaru, hardwaru, webu, aplikaci, operačního systému nebo práci s databází. Usnadňuje vývojářům jejich práci tím, že obsahuje celé bloky sad příkazů, i ukázky vytváření jednotlivých metod, tříd či sekvencí příkazů. Pro implementaci API nemusí vývojář plně rozumnět celé sadě příkazů. Stačí mu ji implementovat a použít přesně to, co je potřeba z ní použít. v případě potřeby lze API použít v jiném programovacím jazyce, než je původně napsána a to pomocí tzv. wrapperů. API v rámci IoT umožnuje širokoúhlou aplikovatelnost a interoperabilitu IoT zařízení. IoTivity používá pro API jazyky C, C++ a Java(obr.2.5) [2].



Obr. 2.5: IoTivity API architektura [1]

2.6 Podporované systémy

Projekt IoTivity podporuje portabilitu pro různé architektury i různé operační systémy jako jsou systémy **Linux, Android, Tizen, Windows** a **iOS**. Jedná se o kompletní softwarové prostředí, které umožnuje uživateli správu mezi hardwarovým a softwarovým prostředím.

2.6.1 Linux

Jedná se o open-sourcový operační systém založený na Unixovém jádru. Hlavní komponentou systému je Linuxový Kernel (jádro systému). Původně byl vyvinut pro osobní počítače na architektuře intelu x86, avšak byl nahrán i na spoustu dalších platforem. Díky dominanci systému Android na smart telefonech, má Linux nejširší obecné uplatnění operačního systému. Patří také k předním operačním systémům pro servery a velké systémy jako mainframe(velké výpočetní celky pro kritické operace). Linux rovněž je využíván na embedded systémech v úzkém spojení s jejich firmwarem. Linux reprezentuje skvělý případ spolupráce komunity na vývoji jedneho systému. Základ linuxu se používá a modifikuje pro různé distribuce at komerčně za-

ložené či nikoliv, avšak podlehájícím obecným licenčním podmínkám. Nejznámějšími modifikacemi jsou CentOS, Debian, Fedora, Gentoo Linux, Linux Mint, openSUSE a Ubuntu. s kombinací určitých softwarových nástrojů a knihoven jsou systémy přizpůsobovány pro různé záměry. [4].

2.6.2 Android

Nejrozšířenější open-source operační systém především pro chytrá zařízení. Hlavní hardwarovou platformou pro android je ARM architektura vedle s x86 a MIPS architekturami pro které byla vytvořena podpora později. Všechny 64-bitové verze všech platforem jsou podporovány dodatečně k 32-bitovým, a to od verze Android 5.0. Zařízení s tímto systémem se často spojují s ruznými hardwarovými komponentami jako: GPS(*Global Position System*), akcelerometry, gyroskopy, barometry, senzory vzdalenosti, tlakové čidla, teploměry, orientační senzory, GPU, magnetometry, audio rozhraní, Wi-Fi, dotyková obrazovka atd.
Primární vývoj Androidu společností Google, je směřován přímo pro *Smart Zařízení* jeho vlastní distribuce. V této skupině Google vydává aktualizace systému poměrně často, a v rozmezí mezi 6-9 měsíci se snaží přijít s novým nástupcem systému. U distribucí všech ostatních firem tyto aktualizace podléhájí modifikaci, pro konkrétní zařízení. Proto je jejich podpora několik měsíců vždy za distribucí společnosti Google zpožděná. Systém pracuje na linuxovém jádře. Verze jádra se odvíjí od konkrétní hardwarové platformy, kdy pro Android 1.0 se jednalo o jádro verze 2.6.25, nejnovější využívají verzi 3.4. Nad linuxovým jádrem stojí middleware(rozhraní mezi jádrem a aplikacemi), knihovny a API psané v jazyce C a software běžící na rozhraní s Java kompilátorem. Pomocí SDK lze vytvářet i vlastní aplikace na cílené zařízení. [7].

2.6.3 Tizen

Open-sourceový a flexibilní operační systém od základu designovaný pro všechny účastníky mobilních a připojených zařízení, včetně zařízení výrobců, mobilních operátorů, vývojářů aplikací a nezávislých vývojářů. Nabízí různé profily systému pro různá odvětví nasazení systému. Vývojářům umožňuje vzít takto přednastavený profil a přidávat či ubírat z něj určité příspěvky kódu. Společnost Samsung nahradila svůj vlastní projekt Bada Tizenem, na kterém pracují nyní veškeré produkty distribuované Samsungem. Spolupracuje stejně jako IoTivity s Linux Foundation. Stejně jako u Androidu obsahuje linuxové jádro. Cílí na použitelnost mezi různými zařízeními navzájem. Jeden z hlavních cílů je užívání komunikačního protokolu HTML5 [8].

2.7 Podporovaná zařízení

Zařízení zde chápeme jako hardwarové výpočetní celky, schopné naučení se příkazů schopné vykonávat sadu instrukcí. Dále jsou to adaptivní zařízení, schopná komunikace s ostatními zařízeními a možností připojovat různé komponenty, např. senzory, Wi-Fi modul. Tato zařízení jsou schopna fungovat podobně jako desktopové počítače, avšak s omezeným výpočetním výkonem, který je limitován operační pamětí, pamětí procesoru, pamětí sady instrukcí, zdrojem energie a výpočetní jednotkou.

2.7.1 Arduino

Jedná se o open-source elektronickou platformou. Nabízí velkou řadu vývojových desek disponujícími různymi výpočtními výkony a pamětí. Jednoduše programovatelné a rozšiřitelná o další moduly. Nabízí vlastní vývojové prostředí Arduino Softwafe IDE pro psaní kódu. Operující na operačních systémech Linux, Windows a OSX. Knihovny jsou psány v jazyce C++, a mohou být dále rozšiřovány. Prostředí není limitováno pouze na desky vydávané distribucí Arduino, může být aplikováno na další mikro-kontroléry jako třeba ESP [6].

2.7.2 ESP

Jedná se minimalizovanou obvodouvou desku s omezenou operační pamětí a výpočetním výkonem od firmy *Espressif*. Jednotlivé modely se odlišují od ostatních především v počtu programovatelných pinů. ESP jsou obsazeny Wi-Fi moduly, které jim umožnují komunikovat po Internetu v pásmu 2,4 GHz Wi-Fi IEEE 802.11 b/-g/n, podporující WPA/WPA2. Nejznámější a nejrozšířenější je modul ESP8266 s 32-bitovým procesorem RISC s pracovní frekvencí 80-160MHz. Tento modul je ideálním reprezentantem IoT a to díky rozměrům a schopnostem plně dostačujícím pro běžné užití. Modul je vybaven 10-bitovým ADC převodníkem a je schopen pulsně kódové modulace. Programovatelný skrz knihovnu v prostředí Arduino IDE. Programovatelný v jazyce Lua, Python a dalších [13].

2.7.3 Chytré telefony

Chytré telefony neboli Smartphone je pokračování mobilních telefonů s pokročilými operačními systémy, které kombinují možnosti osobního počítače a telefonu. Operační systémy se mohou různit. Mohou to být Windows, Tizen, Ubuntu Touch, Android a iOS. Mohou komunikovat pomocí různých technologií (Wi-Fi, LTE, 3G, BLE, Bluetooth a jiné), a fungovat jako klient tak i jako server. Většina těchto zařízení je ovládána uživatelem pomocí dotykových obrazovek (dnes již kapacitních,

dříve i odporových). Bývají osazeny různými hardwarovými komponentami jako fotoaparát, GPS, pohybový senzor, svítící dioda, paměťová úložiště (micro-SD karty), GMS modul, Wi-Fi modul atd. Výkonnost těchto zařízení stále roste, a to především díky novým technologickým procesům zpracování, kdy se uplatňuje vyšší počet jader i frekvence, RAM paměť, rozlišení obrazovek a fotoaparátů a v neposlední řadě i postupné zvyšování kapacit baterií.

2.7.4 Intel Galileo Generace 2

Intel Galileo Generace 2 představuje desku osazenou procesorem Intel Quark SoC x1000 s 32-bitovou architekturou SoC pracující na frekvenci 400 MHz. Procesor podporuje verzi linuxové distribuce Yocto 1.4 Poky Linux. Deska disponuje již vestavěným Ethernet socketem a podporou Power over Ethernet - PoE (napájení po Ethernet kabelu), USB 2.0 porty, micro-SD slotem, PCI Express mini-card slotem, 20ti digitálními I/O piny, mikro USB připojením pro komunikaci se zařízením, ICSP (In-Circuit Serial Programming) hlavou pro přímé zapsání programu na desku, JTAG (Joint Test Action Group) k testování hardwarových funkcí a 2 tlačítka pro resetování. Dále obsahuje RTC (Real Time Clock), pro udržení informace o aktuálním čase s volitelnou 3V baterií. Pracovní napětí je možné volit mezi 3,3V a 5V a to pomocí regulátoru osazeném již na desce [28].

Procesor	SoC Quark X1000
Pracovní napětí	3,3V / 5V
Vstupní napětí	7-15V
Digitální I/O piny	14
Analogové vstupní piny	6
Flash paměť	512kB
RAM paměť	256MB DDR3
SRAM paměť	512kB
Flash úložiště	8MB
EEPROM	8kB
Frekvence jádra	400MHz
Počet jader	1
Architektura	x86
Podpora PoE	Ano
Externí úložiště	mikro-SD až 32GB

Tab. 2.1: Přehled parametrů [28]

2.7.5 Raspberry Pi 2 B

Jedná se o variantu modelu z řady Raspberry Pi, známé svou energetickou nenáročností kapesního počítače. Oproti předchozí verzi jsou změny omezeny na výkon a to z přechodu jednojádrového procesoru o taktu 700MHz na 4-jádrový procesor o taktovací frekvenci 900MHz pro každé jádro. RAM paměť se navýšila na kapacitu 1GB. Deska je vhodná pro zkoušení a naučení se práci s projekty na bázi linuxu i vyzkoušení práce s I/O periferiemi připojitelných pomocí GPIO pinů. Model 2 zaručuje zpětnou kompatibilitu se starší verzí 1 a jejím softwarem. se zájmem společnosti Microsoft byl vyvinut systém Windows 10 IoT Core.

- *Raspberry Pi 2 B* odpovídá rozměrově první generaci z této řady. Je osazen čtyřmi USB 2.0 porty, audio výstupem v podobě konektoru *jack 3,5mm*, HDMI portem pro audio/video výstup, ethernetovým portem a slotem pro mikro-SD kartu, na kterou je možné nahrát jakýkoliv kompatibilní systém, jelikož tato deska nedisponuje žádnou vnitřní pamětí. Dále je vybavena i GPIO piny pro připojování různých periferií. Napájení je realizováno pomocí micro-USB vstupu o napětí 5V.
- Raspeberry Pi 2 disponuje procesorem Broadcom BCM2836 32-bitové architektury se čtyřmi Cortex-A7 jádry o frekvenci 900MHz s možností přetaktování na 1,1GHz.
- RAM paměť se navýšila na 1GB. Grafický čip zůstal stejný jako u minulé řady a to Broadcom VideoCore IV. Jeho možnosti vyplývající z Benchmark testů znamenají velkou problematiku běhu systému Android 5.0, jelikož Android spoléhá od verze 4.0 na značný výkon Grafických akcelerátorů. Doporučený optimální výkon je tak operační systém Raspian, kde se nejedná o nic jiného než o upravenou verzi Debianu, která se často modifikuje i pro jiné desky jako výchozí podporovaný systém [29].

3 SPUŠTĚNÍ IOTIVITY

Pro praktickou část práce budou uvedeny závislosti a postupy nutné pro spuštění rozhraní IoTivity. Bude rozebrána komunikace na příkladu včetně ilustrace a ukázek kódu. Spuštění IoTivity bude prezentováno na různých hardwarových platformách. Poté na konci kapitoly budou popsány nezmíněné poznatky a problematika praktické části.

3.1 Nastavení prostředí a spuštění

Zde bude uveden pouze stručný postup instalace a nastavení prostředí, zbytek je k nalezení v příloze. Prostředím pro tvorbu a editaci spustitelných souborů zde bude probíhat v operačním systému linuxové distribuce Debian, konkrétně podporovaná verze `Ubuntu 12.04LTS` v souladu webových stránek projektu, ze kterých bylo v postupu vycházeno [1].

1. Stažení souboru obsahujícího všechny zdrojové kódy k projektu z oficiální distribuce [1] nebo nerevizované verzi od kontributorů.
2. Instalace balíčků knihoven, které jsou uvedeny [21][23].
3. Pro kompilaci, tedy převedení zdrojových kódů jazyka C++/Java do spustitelné podoby slouží nástroj pro kompilaci `scons`.
4. Při prvních spuštěních kompilačního programu scons se objeví požadavek pro stažení knihovny `Tinycbor` do složky `extlibs` - externích knihoven, s nimiž kompilační nástroj pracuje.
5. Příkazu scons je nutné předat parametry, podle kterých bude kompilace probíhat pro určitou SoC, dále OS, případně typy komunikace jako zabezpečená a veřejná, které však v této práci nejsou předmětem testování. V případě bezparametrické syntaxe budou nastaveny výchozí parametry pro spuštění kompilace a sestavení. Přehled parametrů v tabulce: 3.1.
6. Dále po úspěšném průběhu scons-scriptu je možné v cílové složce, do které byly soubory přeloženy, spustit skrz terminál. Konkrétně pro Linuxse nachází odkaz zkompilovaných souborů s konkrétní cestou.

   ```
   ~/iotivity-master/out/linux/x86_64/release/resource/examples
   ```

7. Pro demonstraci a popis bude spuštěn SimpleServer z konzolového prostředí počítače a SimpleClient z telefonu se systémem Android.

Tab. 3.1: Scons parametry pro kompilaci [1]

TARGET_ARCH	**Podporované architektury**
	x86 (přednastavený)
	arm
	armeabi
	armeabi-v7a
TARGET_TRANSPORT	**Podporované druhy transportu**
	ALL (přednastavený)
	IP
	BT
	BLE
RELEASE	**Podporované módy buildu**
	1 (přednastavený-relase mód)
	0 (debug mód)
SECURED	**DTLS je povoleno / zakázáno**
	0 (DTLS vypnuto - přednastavelo)
	1 (DTLS zapnuto)
ADNROID_HOME	Zde se nastaví cesta pro Android SDK
ADNROID_NDK	Zde se nastaví cesta pro Android NDK
GRADLE_HOME	Zde se nastaví cesta pro *Gradle/bin/*

3.2 Server a klient

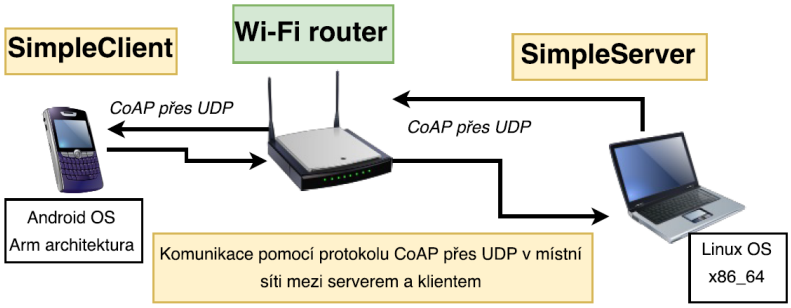
Jedná se o základní síťovou architekturu modelu komunikace 2 a více účastníků. Služby jako hostování, objevování a užití GET, PUT, POST a OBSERVE operací na zdroji. Komunikace probíhá na základě dotazu a odpovědi. Tuto logiku používají *HTTP* servery (přístup na webové stránky) obr. 3.1.

3.2.1 Navázání komunikace

1. **Vytvoření a konfigurace platformy**
   Prvně je potřeba vytvořit a nastavit platformu pro komunikaci, kde se nastaví typ služby, její kvalita, a naslouchání, které je zde formou multicastu. Velmi podobně se nastaví i klient. V jiných případech se pro CoAP nastavuje:
   - `Multicast Port: 5683`
   - `Secure Port: 5684`

Výpis 3.1: Vytvoření platformy serveru IoTivity

```
1 //nahlednutí do databáze jestli již klient není registrován
```



Obr. 3.1: Komunikace SimpleServer - SimpleClient

```
2  OCPersistentStorage ps {client_open,fread,fwrite,fclose,unlink};
3  PlatformConfig cfg {
4  OC::ServiceType::InProc,//průběžná služba
5  OC::ModeType::Server,//typ zdroje - server
6      "0.0.0.0", //naslouchání pro všechny prvky v~síti
7      0,         //Použije náhodný port (nezabezpečeno)
8      OC::QualityOfService::LowQos, //kvalita síťove služby
9      &ps //odkaz 1.komentář
10 //Pro klienta HighQos(volitelné), ModeType::Both
11 };
```

2. **Vytvoření zdroje**

   Zdroj vytváří prvek server. Server referuje jednotku (implementovanou ve zdroji), která reprezentuje aktuální stav například senzoru nebo jiný vhodný údaj. Tento zdroj je sestaven v následující podobě.

Výpis 3.2: Vytvoření zdroje serverem IoTivity

```
1  //registerResource značí metodu zaregistruj
2  //m_resourceHandle typ komunikační metody (GET,SET,POST..)
3  //resourceURI URI která specifikuje zdroj
4  //resourceTypeName označení - naše pojmenování zdroje
5  //resourceInterface zdrojové rozhraní, cb volá handle žádost
6  //resourceProperty bitová maska, udává možnosti pro zdroj:
7  //Observable-zdroj podporuje notyfing, Secure-zabezpečení
8  //Discoverable-Odpový na Discovery žádost - viditelnost
9
10 OCStackResult result = OCPlatform::registerResource
11 (m_resourceHandle,resourceURI,resourceTypeName,
12 resourceInterface,cb,resourceProperty);
```

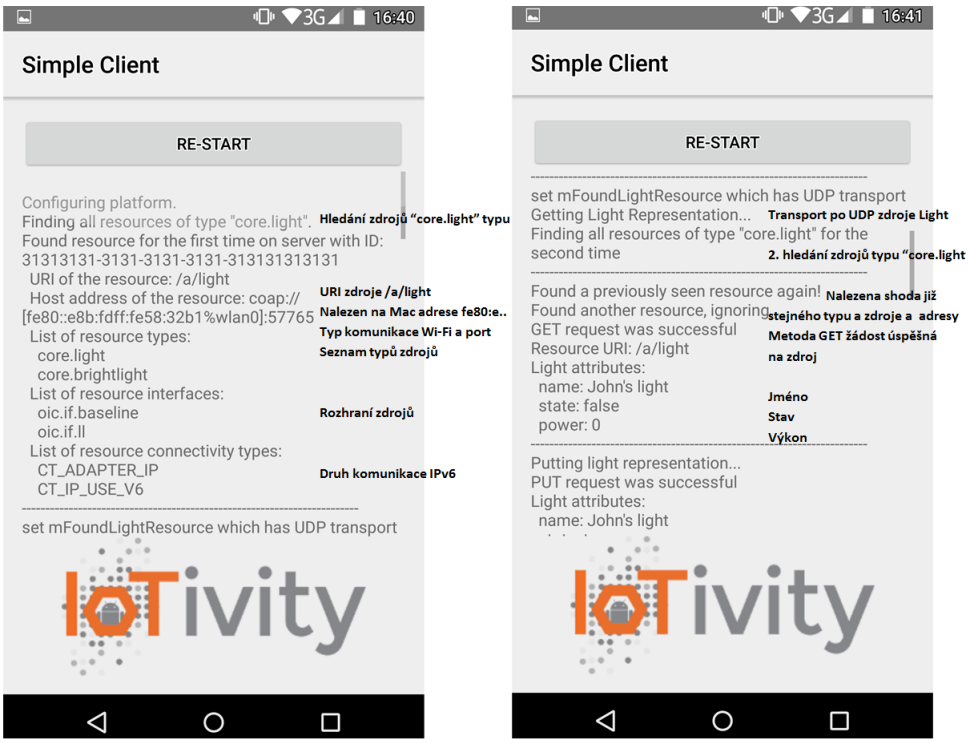
3. **Vyhledání zdroje**

   Klient nyní začne vyhledávat dostupné zdroje pomocí multicastové adresy, aby se k nim mohl přihlásit v lokální síti. Vyhledávání probíhá pomocí parametru

cesty, URI - specifického označení. Přes tento identifikátor k serveru může být připojeno více klientů se stejným URI. Ilustrováno na výpisu 3.3.

Výpis 3.3: Vytvoření zdroje serverem IoTivity

```
1 //přednastavené "", a~URI indentifikátor
2 OCPlatform::findResource("", requestURI.str(),
3       CT_DEFAULT, &foundResource);
4     //typ připojení, adresa zdroje
```



Obr. 3.2: Zahájení komunikace klienta

4. **Žádost/odpověď**
   Server použije implementovanou `Entity handler` funkci pro registraci sady `CRUDN` žádostí (obrázek 3.2). Pomocí těchto žádostí si cíleně obě strany vyměňují data. Jedná se o tzv. `callback` funkci, což znamená odpověď na vyžádání. Jedna strana pošle žádost GET-dostaň a druhá odpoví metodou obvykle POST.

Tab. 3.2: Shrnutí metod CRUDN [19]

Termíny dle OCF	**IoTivity termíny**
Create-vytvoř	POST
Retrieve-získej	GET
Update-nahraj	PUT
Delete-smaž	DELETE
Notify-sleduj	OBSERVE

Výpis 3.4: Odpověď serveru na žádost PUT klienta

```
1  private:OCEntityHandlerResult entityHandler(std::shared_ptr
2          <OCResourceRequest> request){
3          OCEntityHandlerResult ehResult = OC_EH_ERROR;
4      if(request){
5          std::string requestType = request->getRequestType();
6          int requestFlag = request->getRequestHandlerFlag();
7      if(requestType == "GET"){
8          put(rep);//Nastaví hodnotu pro server
9          pResponse->setResponseResult(OC_EH_OK);
10         pResponse->setResourceRepresentation(get());
11         if(OC_STACK_OK == OCPlatform::sendResponse(pResponse))
12           {ehResult = OC_EH_OK;/*potvrzení vše v~pořádku*/ }}}
```

5. **Ukončení komunikace**

   Server nedostává další zprávy typu požadavku od klienta. Server zůstává připraven na další komunikaci s klientem. Server zůstává spuštěň dokud není vypnut. V tomto případě uživatelem. Pro ukončení komunikace se nyní zavolají destruktory, které se postarají o smazání objektů a jejich reprezentací. V tomto bodě dochází k přerušení komunikace. Pokud je spuštěn klient dřív, než-li server vytvoří zdroj, klient své hledání zdrojů při druhém pokusu ukončí a na vytvořený server už nereaguje, dokud není uživatelem znovu podnícen k aktivitě obrázek 3.8.

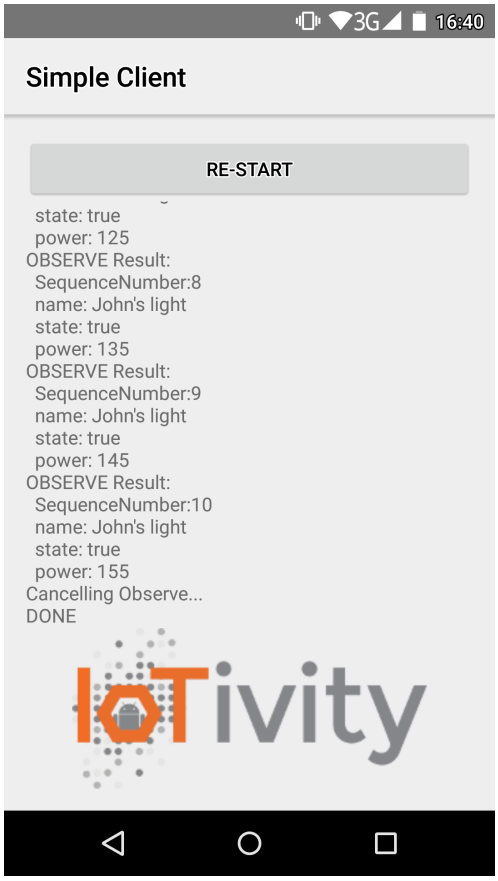
Výpis 3.5: Konec komunikace SimpleServer

```
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : GET
    requestFlag : Observer

Power updated to : 165
Notifying observers with resource handle: 0x2028c50
```

```
No More observers, stopping notifications
```



Obr. 3.3: Konec komunikace SimpleClient

3.3 Konfigurace IoTivity pro Intel Galileo

V této části bude popsána práce na zařízení Intel Galileo Gen. 2, na které bude spuštěn server. Pro Intel Galileo zatím nebyla vyvinuta podpora, což se v budoucnu bude měnit.

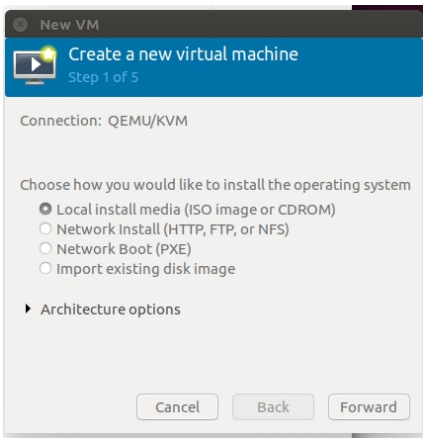
3.3.1 Virtuální stroje

Pro práci byla zvolena cesta virtuálních strojů, z hlediska možnosti vytvořit tzv. *snap-shoty* - za účelem uložení systémové konfigurace v určitém stavu. Prvním testovaným virtualizačním strojem byl **VM Virtualbox** [30].

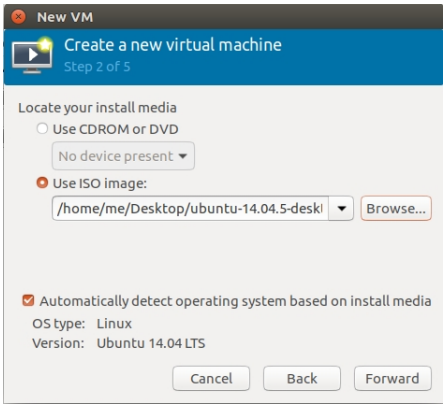
- **VM Virtualbox**
  Jako první byla stažena a použita nejnovější verze `5.1.21` včetně rozšiřovacího modulu (extension pack) ze stejné domény. Hostující systém byl zvolen OS Windows 10. Systém pro instalaci byl zvolen dle doporučení Ubuntu 14.04 LTS. Instalace s touto verzí virtuálního stroje nefungovala a byla odkoušena řada několika starších vydání. U nich instalace proběhla v pořádku, nicméně systém i přes dostatek přidělených prostředků pro práci reagoval s velkou odezvou a působil dojmem nedostatku prostředků pro zpracování procesů. Později stávalo, že systém za běhu zamrznul a neodpovídal na jakékoliv žádosti přes vstupní terminály.
- **QEMU**
  Jedná se o prostředí umožňující spuštění a běh dalšího operačního systému v již spuštěném systému. Tento emulátor je dostupný pro linuxové systémy. Postup konfigurace je následující.
  Stažení *qemu emulátoru* se spouštěcím manažerem a *virt-viewer* ke správě systému.

  ```
  sudo apt-get install qemu-kvm qemu virt-manager \
  virt-viewer libvirt-bin
  ```

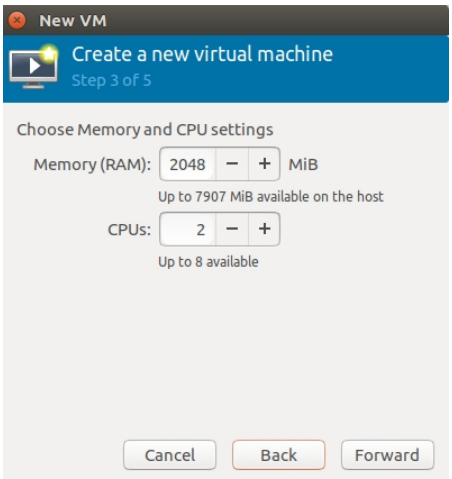
  Instalace systému se realizovala pomocí virt-manageru, ve kterém byly nastavovány postupně parametry výpočetních prostředků. Po skončení instalace se zavolá v konzoli virt-manager a z něj se spouští daný systém, který funguje v okně nebo na celé ploše. Výkon přiřazený zde virtuálním strojům je vyšší, než by se mohl zdát potřebný. Tyto virtuální systémy však slouží ke kompilacím a později jim byly přiděleny prostředky o 4 jádrech z 8 celkových. Hostující procesor je `Intel i7 4700MQ` s 4 fyzickými jadry a každé s možností virtualizace vláken. V tomto virtualním stroji probíhala práce v systému oproti předešlému *Virtualboxu* velmi plynulá skoro jako v nativním systému. Systémy byly vytvořeny dva. Oba `Ubuntu 14.04LTS` s rozdílem `32/64bit` architektur.



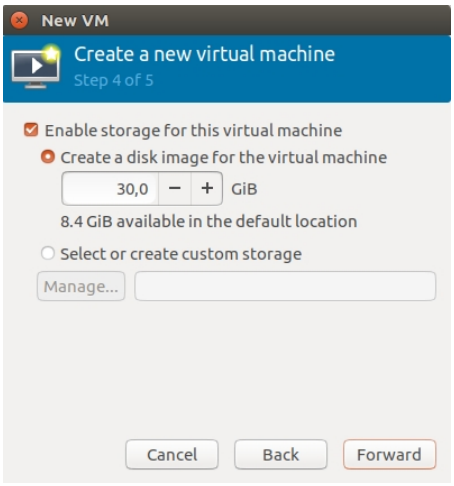
Obr. 3.4: Vybrání typu média pro instalaci



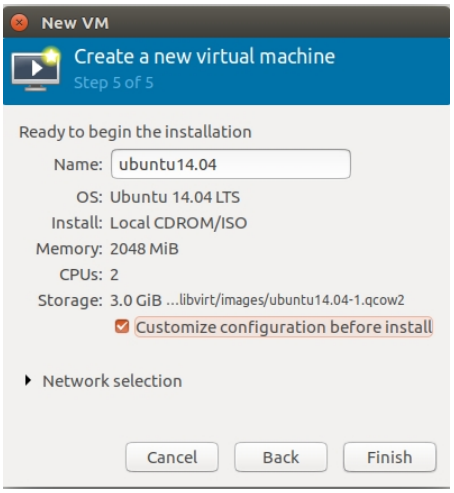
Obr. 3.5: Zvolení konkrétní cesty k médiu



Obr. 3.6: Rezervace prostředků



Obr. 3.7: Alokace místa na disku



Obr. 3.8: Rekapitulace nastavení

3.3.2 Instalace Galilea

Jak bylo zmíněno v teorii, toto zařízení dipsonuje mikro SD kartou, na kterou se uloží operační systém – v tomto případě bylo zkoušeno několik verzí *Ubilinuxu*, což je operační systém s linuxovým jádrem odvozeným pro výkonově omezená zářízení jako je například zařízení Galileo. Není zde implementované grafické uživatelské rozhraní *GUI*, je zde použita pouze konzole.

- **Firmware**
  Pro správné fungování nejnovějšího operačního systému je nutné mít i poslední verzi firmawaru. Původní verze s níž byla deska přijata byla `v1.0.4` a následně upgradovaná na `v1.1.1`. Toto proběhlo pomocí aplikace z prostředí Windows

10 kdy přes komunikační USB kabel byl do desky bez SD karty a dalšího příslušenství nahrán nový firmaware. Byl následován tento návod [43].

- **Příprava SD karty**
  Kartu je nutno pro nahrání operačního systému vhodně formátovat, a to na formát `EXT3`. Pro tuto změnu formátu sloužila nativní aplikace v Ubuntu Disk Manager. s tímto formátem a prázdná je vložena do paměťového slotu. Nyní je nutné stáhnout Ubilinux [31] a extrahovat jeho obraz na USB disk. Vypnuté zařízení s oběma komponentami zapojenými se připojí ke zdroji a začne interní instalace bez nutného zásahu uživatele. Signalizací, že se opravdu systém instaluje je indikace zápisu / čtení u diody SDkarty. Samotná instalace trvá cca 15 minut, poté se zařízení vypne. Prvotní přihlašovací údaje jsou `root` pro uživatele s totožným heslem.
- **Práce v systému**
  Samotný Ubilinux je orientován na konzolové prostředí, který je jediným grafickým výstupem. Pro komunikaci byly použity programy Tera Term, Putty a příkazový řádek. Prvotní komunikace probíhala výhradně přes USB serial port, později byla přidáná Wi-Fi karta na mini-PCIe sběrnici s následnou konfigurací v systému. Stažení ovladače pro Wi-Fi při připojení ethernetového kabelu.

  ```
  apt-get update && apt-get install firmware-iwlwifi
  ```

  Povolení instalace modulu pro kartu

  ```
  modprobe -r iwlwif of; modprobe iwlwifi
  ```

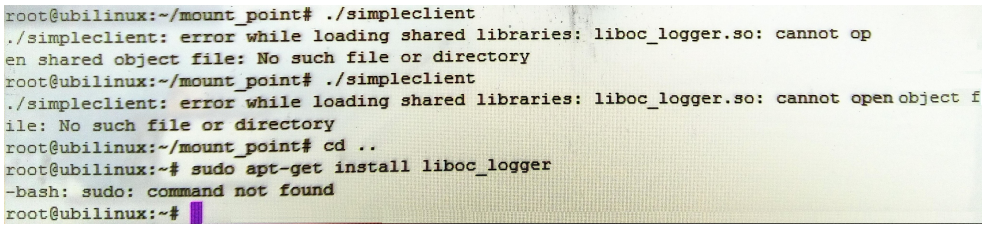
  Editace v systému, kde bylo třeba přidat aktivní režim pro wlan0 rozhraní a nastavit konkrétní připojení k síti, její SSID a heslo.

  ```
  nano\,/\,etc\,/\,network\,/\,interfaces
  auto wlan0
  iface wlan0 inet dhcp
  wpa-ssid Jmeno_Site
  wpa-psk heslo
  ```

  a nakonec restartování Wi-Fi.

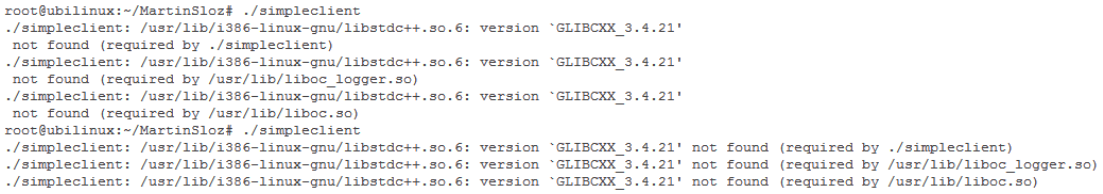
  ```
  ifconfig wlan0 down
  ifconfig wlan0 up
  ```

  Poté bylo na základě přesné IP adresy možné přistupovat k zařízení pomocí SSH spojení v místní síti.
- **IoTivity**
  Jako první byl vyzkoušen již zkompilovaný soubor *simpleclient* pro architekturu x86. Bohužel se u zařízení nepodařilo soubor spustit viz obr. 3.9.



Obr. 3.9: Spuštění binárního souboru simpleclient na Galileu

Výpisy odkazovaly na různé knihovny. Později byla tato knihovna vysledována a tyto nekompatibility způsoboval stroj, na kterém byl tento kod kompilován. Jelikož kompiloval s knihovnami potřebnými i pro následný běh aplikace, konkrétně knihovna Glib3.41 obr3.10. Přešlo se z *Ubuntu 16.1* na *Ubuntu*



Obr. 3.10: Chyba v knihovně Glib

14.04LTS. Později při spuštění IoTivity na Galileu se vypisovala zpráva do konzole

```
"Illigal instruction"
```

Tato situace byla osvětlena na internetových diskuzních fórech, kde chyba odkazovala na interní chybu pro komunikaci s procesorem. Později byla odzkoušena i kompilace na desce Galileo. Prostředí bylo obdobné a nastavení různých podkladů pro běh IoTivity se řešilo podobně jako na plnohodnotném systému. Bohužel ani tady se nepovedlo chybu odstranit. Při pokusu přejít na vyšší verzi Ubilinuxu došlo ke zvláštní chybě, kdy se nedalo přihlásit do systému. Systém uživatele ihned odhlásil a v této smyčce již zůstalo zařízení zaseknuté. Další krok směřoval k nápravě této nekompatibility strojového kódu s procesorem ve spolupráci s projektem Yocto.

3.3.3 Yocto image

Yocto je projekt zabývající se vývojem a úpravami linuxu pro různá zařízení například jako je deska Galileo. Na internetu je k nalezení několik verzí jak vytvořit Yocto pro intel Galileo, kde 2 z nich jsou distribuovány společností *Intel*.

- První vystavění obrazu následovalo tento návod [41]. Postup spočívá ve stažení git repozitářů. Poté vnoření se do prostředí pro spuštění kompilace Yocto obrazu.

  ```
  source oe-init-build-env
  ```

  Vytvoření nového konfiguračního souboru do složky,

  ```
  conf/auto.conf
  ```

  ve kterém jsou konfigurační parametry pro konkrétní třídu zařízení. Jak je vidět, parametr MACHINE se rovná typu série procesoru, který nese zařízení Galileo.

  ```
  DISTRO = "iot-devkit-multilibc"
  PACKAGE_CLASSES = "package_ipk"
  MACHINE = "quark"
  ```

  Poté stačí spustit příkaz na vytvoření obrazu dle parametrů.

  ```
  bitbake iot-devkit-prof-dev-image
  ```

  Tento průběh nikdy nedoběhl do konce. Vždy byla obdržena nějaká chyba při kompilaci. Ke konci již stálá chyba

  ```
  OE-core's config sanity checker detected
  a~potential misconfiguration.
  Either fix the cause of this error or at
  your own risk disable the checker (see sanity.conf).
  ```

  Soubor sanity.conf se nepovedlo dohledat ani vypnout checker.
- Z jiného vlákna byl následován opět postup ze stránek společnosti *Intel* [42].
  1. Zkontrolování všech závislostí pro systém linux

     ```
     sudo apt-get install build-essential gcc-multilib\
     vim-common uuid-dev iasl subversion
     ```

  2. Stáhnutí prostředí pro takzvaný BSP neboli board support package, což je balíček obsahující různé soubory pro vystavění systémového obrazu pro desku [36].
  3. Stažení patche a nahraní do složky se stejným názvem. Verze patche nebyla označena konrkétně, proto byl zvolen patch s co nejbližším datem vydání dokumentu.
  4. Nainstalování potřebných nástrojů a rozbalení balíčku meta-clanton

     ```
     sudo apt-get install diffstat gawk chrpath
     tar -xvf meta-clanton*.tar.gz
     cd meta-clanton*
     ./setup.sh
     ```

     Poslední příkaz setup.sh je bezparametrický a vytváří root systém pro kernel Intel Galilea

5.
```
source poky/oe-init-build-env yocto_build
bitbake image-spi-galileo
```

Tato instrukce neproběhla do konce. Skončila s výpisem chyb

Výpis 3.6: Výpis chyb při bitbake

```
ERROR: Function failed: do_patch (see/home/me/Galileo-
Runtime/meta-clanton/yocto_build/tmp/work/clanton-poky-
linux-uclibc/linux-yocto-clanton/3.8-r0/temp/
log.do_patch.2202 for further information)
ERROR: Logfile of failure stored in: /home/me/Galileo-
Runtime/meta-clanton/yocto_build/tmp/work/clanton-poky-
linux-uclibc/linux-yocto-clanton/3.8-r0/
temp/log.do_patch.2202
ERROR: Function failed: do_patch (see /home/me/Galileo-
Runtime/meta-clanton/yocto_build/tmp/work/clanton-poky-
linux-uclibc/linux-yocto-clanton/3.8-r0/temp/
log.do_patch.2202 for further information)
ERROR: Task 239 (/home/me/Galileo-Runtime/
meta-clanton/meta-clanton-bsp/recipes-kernel/linux/linux
-yocto-clanton_3.8.bb, do_patch) failed with exit code '1'
```

6. Dle prvního návodu by následovala po úspěšném bitbaku tato syntaxe. Podle [32] návaznost byla jiná. Proto bylo vyzkoušeno toto pořadí.

```
cd Quark_EDKII*
./svn_setup.py
svn update
```

Tato operace se opět nepovedla uskutečnit do konce. Objevila se následující chyba:

Výpis 3.7: Výpis chyb při svn_setup.py

```
INFO:
 Ad-hoc SVN repo:
 /home/me/Desktop/edk2-staging/Quark_EDKII_v1.2.1.1-
 svn_externals.repo
 INFO:   SVN version:
 OSError: [Errno 2] No such file or directory
 ERROR:  Failed to run Subversion command line,
 please make sure it is installed and in the PATH
```

Tento problém je vyřešen po dohledání na internetových fórech s pomocí instalace balíčku [37].

```
sudo apt-get install subversion
```

7. Následovně inspirováno dle [34].

8. Dalším krokem je nutnost mít ovladače iASL pro práci s rozhraním a tvoření kernelu nebo jeho úprav. příkazem

```
sudo apt install apcica-tools
```

9. Jelikož hlásí chybu ve verzi acpica-tools, je třeba manuálně stáhnout a extrahovat náležitou verzi - `4.1` z [35] do složky

Výpis 3.8: BSP

```
*/BSP_Sources_and_Docs_for_Intel_Quark_v1.2.1.1\$
tar -xzvf Quark_EDKII_v1.2.1.1.tar.gz
```

10. Spuštění skriptu:

```
./buildallconfigs.sh GCC46 QuarkPlatform
```

Kde bylo prvotně testování různých GCC verzí, se kterými bylo dosaženo stejného výsledku. Výpis požadoval úpravu v OpenSSL. V dokumentu `HOWTO.txt`, který je umístěn v současné složce EDK odkazoval na stažení balíčku z [38].

11. Extrakce do

```
~/Desktop/edk2-staging/Quark\_EDKII\_v1.2.1.1/
CryptoPkg/Library/OpenSslLib/openssl-0.9.8zb
```

12. Stáhnutí patche z [39]. Použita byla verze 2.7 s přesměrováním na stránku [40].
13. Vstup do složky

```
cd \$(WORKSPACE)/CryptoPkg/Library/OpensslLib/
openssl-0.9.8zb/patch*
```

v této složce je textový dokument `INSTALL` a dle něj následující instrukce

```
  cd ..\,/\,./configure
make (spusti kompilaci)
sudo make install (nainstaluje zavislosti)
  patch -p0 -i ../EDKII_openssl-0.9.8zb.patch
  cd ..
./Install.sh
```

14. Následuje vrácení se do hlavní složky, ze které je třeba spustit příkaz:

```
./buildallconfigs.sh GCC46 /usr/bin/ QuarkPlatform
```

již upravený o parametry `/usr/bin/`, což je cesta k binárním souborům GCC. Build neproběhl úspěšně opět pro Ubuntu 16.1 s obdržením nové chybové zprávy:

```
 build.py...
:error 000E: File/directory not found in workspace
/home/andre/tmp/galileo/new_BSP/
Quark_EDKII_v1.2.1/Pkg/Pkg.dsc
```

15. Částečné řešení bylo v těchto krocích později realizováno

    ```
    sudo apt-get install nasm
    ./buildallconfigs.sh GCC48 /usr/bin/ QuarkPlatform
    ```

    S výsledkem neúspěšným pro Ubuntu 16.1, ale pro Ubuntu 14.4 ano. Avšak bylo nutné dotvářet uměle složky `GCC_DEBUG, PLAIN, GCC_48 GCC_47` poté build proběhl úspěšně. To byly kroky k nakonfigurování prostředí pro vytvoření obrazu, který zatím vytvořen nebyl. Jeho kompilace proběhla s chybami, pro které už nebylo nalezeno řešení. V této části bylo rozhodnuto pro přechod od zařízení Intel Galileo Gen 2. Přechází se na testování podporovaného pro Raspberry Pi 2B.

3.4 Server Raspberry Pi

Pro aplikaci typu server bylo zvoleno zařízení Raspberry Pi 2B, které podle serveru IoTivity má funkční podporu. Nastavení zařízení probíhá v následujících krocích:

1. Stažení operačního systému Raspbian [44].
2. Nahrání systémového obrazu na mikroSD kartu [45]. Pomocí příkazu dd (disk destroyer), který zapisuje byte po bytu, konkrétně po 4 MB blocích. Výstupní disk of=/dev/sdX bylo nahrazeno patřičným označením v systému a to sdc2.

   ```
   dd bs=4M if=2017-04-10-raspbian-jessie.img of=/dev/sdX
   ```

3. Poté se paměťové médium vložilo do zařízení Raspberry a při zapnutí se spustila systémová instalace v grafickém prostředí.
4. Dalším krokem bylo aktualizování systému.

   ```
   sudo apt-get update && upgrade
   ```

5. Instalování závislostí pro Raspberry Pi [46].
6. Vytvoření vývojového prostředí IoTivity na počítači pro Raspberry. Všechna data byla přenesena do virtuálního stroje Ubuntu 14.04LTS 64-bit. Pomocí příkazu chroot se přeplo do prostředí systému Raspberry Pi a mohl se využít výpočetní výkon počítače ke kompilacím [46].

   ```
   sudo chroot /home/developer/rpichroot/
   ```

7. Kompilace pro Raspberry má obměněné parametry pro cílovou architekturu. Procesor je řady ARM, upraví se parametry následovně:

   ```
   scons TARGET_ARCH=arm /resources/examples
   ```

3.4.1 Server

Základní šablonou pro serverovou aplikaci slouží zdrojový kód:

```
*/iotivity-1.2.1/resources/examples/simpleserver.cpp
```

Zdrojový kód byl analyzován, odstranili se metody pro Windows, které nebudou pro Raspberry Pi užity. Poté se změnily parametry hlavičky zdroje, přidala se metoda observe, pro průběžné odesílání dat na klienta.

```
void * ObserveResource (void *param);
```

Uri pro identifikaci bylo změněno:

```
std::string resourceTypeName = "core.raspberry";
```

Metoda PUT byla upravena pro komunikaci s klientem ve režimu, kdy klient může zadat režimy:

- **Heating mode: Auto / Manual**

- **Set: Vložení minimálních a maximálních teplotních limitů**
- **Heating: Zapnutí / Vypnutí topení**

Pomocí metody get jsou jejich hodnoty od uživatele změněny pro vnitřní stav serveru. Dalším objektem přidaným je metoda pro čtení dat ze senzorů.

```
void readSensors()
```

V této metodě probíhá čtení z textových souborů, do kterých jsou průběžně přepisována data. Data reprezentují tyto hodnoty:

- **Teplotu - Temperature**
- **Vlhkost - Humidity**
- **Světlo - Light**

Teplota a vlhkost jsou čteny ze souboru ve formátu *array* - pole znaků. V textovém souboru jsou tato data dohromady, proto pomocí kombinace cyklu *for* a podmínek *if* jsou tyto řetězce rozděleny a jejich obsahy jsou převedeny na *string* a ten je pomocí knihovny *std::stod* převedeny na číslo datového formátu *double*. Vyčítání z pole do řetězce je řešení, které bylo možné implementovat. Například funkce *putc() getc()* nebylo možné zkompilovat. Obdobné řešení je užito i pro čtení hodnoty světla. Ta je realizovaná s pomocí fototranzistoru a vysílá pouze stav, kdy intenzita ozáření přesáhne určitou nastavenou mez a sepne do stavu *pravda*, jinak je vypnutá ve stavu *nepravdy*. Je zde užita *booleovská* logika. Text se nepřevádí do stringu ani se nevyčítá, pouze se provede podmínka pro čtení znaku který reprezentuje stav 1 / 0. Metoda *readSensors()* obsahuje také ovládání *topení* dle vnitřní změny stavu provedené přes *entityHandlerPut* - tedy změnách na základě *PUT* požadavku, a to ve dvou stavech:

- **Auto** - V tomto režimu se teplota nastaví a udržuje dle teplotních intervalů.
- **Manual** - Zde se zapne ohřívání a vypne opětovným příkazem.

Topení se spíná pomocí zavoláním skriptu v jazyce *Python*, který přivede na digitální výstup logickou hodnotu *HIGH* o velikosti 3,3V, což sepne tranzistor do sepnutého stavu. Zápisy do textových dokumentů jsou realizováný spuštěním skriptů v jazyce *Python*, které jsou volány jako subprocesy v hlavní funkci ***main***.

```
if(fork() == 0){
int status = system("/home/pi/Desktop/teplota.py 22 4");
        exit(0);
  }
```

Server začíná spuštěním funkce main(), kde se inicializuje prostředí pro IoTivity. Platforma pro IoTivity funkce se implementuje s pomocí *OCPlatform::Configure(cfg);*. Obecněji sumarizováno do:

```
#include "OCPlatform.h"
```

```
#include "OCApi.h"
#include "ocpayload.h"
```

K registraci zdroje je potřeba dvou základních položek:

- Handler - Zpracovává žádosti
- URI path - Cesta k registraci zdroje

Registrace zdrojů (resources) - řeší konkretní třídy zdrojů přes funkci *createResource()*. Dále je hlavní IoTivity funkce OCPlatform::registerResource(), která zaregistruje zdroje. Ta předá funkci "callback"vnitřní stav, který zavolá v případě dotazu na daný zdroj. Callback funkce je tvaru:

```
EntityHandler cb = std::bind(&BaseResource::entityHandler,
                   this,PH::_1)
```

Je definovaná tak, že bude volat funkci entityHandler. V mainu jsou zaregistrované dva zdroje představující senzory a topení. Dále v mainu bylo vytvořeno další vlákno, které vyčítá ze senzoru, nastavuje a ukládá data do lokalních proměnných. Smyčka běží ve funkci *sensorLoop()*, která volá funkci *readSensors()*.

```
pthread_t sensorThreadId;
pthread_create (&sensorThreadId, NULL, sensorLoop, NULL);
```

Hlavní funkce pro zdroj je callback funkce a je volána podle požadavku na zdroj, který přichází od klienta.

```
OCEntityHandlerResult entityHandler(std::shared_ptr
<OCResourceRequest> request)
```

V této funkci se testuje typ požadavku, jestli jde o RequestFlag nebo ObserverFlag. V případě RequestFlag - se dále testuje, zda jde o dotaz GET nebo PUT. Předání parametrů probíhá následovně:

```
pResponse->setResourceRepresentation(get());
```

Předává se funkce *get()*, která naplní objekt *OCRepresentation m_rep.* daty: ve stylu klíč, hodnota:

```
m_rep.setValue("temperature", m_temperature);
m_rep.setValue("humidity", m_humidity);
m_rep.setValue("light", m_light);
```

V případě, že byl zaslán požadavek na *OBSERVE*, v *callbacku entityHandler* se rozhodne, zda jde o registraci pozorovatele (OBSERVER) a na tomto základě je přidán nebo odebrán z listu pozorovatelů. Následně se vytvoří vlákno, které volá funkci na níž ukazuje a předává ji parametr this. Následně se vytvoří vlákno, které volá funkci na níž ukazuje *m_observe_fce* a předává jí parametr *this* (instanci objektu). Objekt

m_observe_fce je nastaven pro funkci *ObserveResource*, která běží pořád v případě, že existuje pozorovatel posílá pořád info o zdrojích.

3.4.2 Klient

Klient byl zvolen v podobě aplikace pro mobilní telefon operačního systému *Android 5.1*. IoTivity uvádí na svých stránkách podporu tohoto zařízení. Je zajímavé i svou širokou aplikací a rozšířením mezi lidmi. Z tohoto důvodu je dobré jej pro naše účely otestovat. Pro začátek je třeba dodržet následující kroky:

1. Operační systém Linux, v tomto případě *Ubuntu 16.1 64-bit*
2. Instalace platformy pro práci s jazykem JAVA ve verzi *openJDK 1.7* nebo vyšší. Zvolena je doporučená verze *1.7*.

   ```
   sudo apt-get install openjdk-7-jdk
   ```

3. *Android Studio* - Prostředí pro vývoj aplikací se systémem Android [47].
4. Instalace *Gradle 2.2.1* nebo vyšší. Použitá verze je *3.2*. Je vhodné, aby gradle byl instalován společně do složky, kde je instalováno *android-Studio*
5. Vytvoření složky s binárními soubory pomocí

   ```
   scons TARGET_OS=android TARGET_ARCH=armeabi-v7a
   ```

6. Importování aplikace *Android* pro další úpravy do *Android Studia*. Musíme najít binární distribuci buildu, která je identifikována koncovkou *.aar*.

   ```
   /iotivity-1.2.1/android/android_api/base/build/
   outputs/aar/iotivity-base-armeabi-v7a-release.aar
   ```

7. v Android Studiu vytvoříme nový projekt. V sekci *Configure* následujících oken vybereme patřičné schémata a *API 21: Android 5.0 (Lollipop)* pro telefon a tablet.
8. Nyní je importováno binární API jako nový modul.

   ```
   File>New Module> Import .JAR or .AAR Package
   ```

   je vložena cesta k *.aar* souboru.
9. Vlevo je označena *app*, a pravým tlačítkem se otevře okno a je vybrán *Open Module Settings*. V záložce *Dependencies* se přidá *Module dependency* a vybere se *iotivity-base-armeabi-release*.
10. Po vzoru zdrojových kódů v:

    ```
    iotivity-1.2.1/android/examples/simpleclient/
    src/main/java/org/iotivity/base/examples/
    ```

    Vytvoří se hlavička *SensorsRes*, sloužící k vytvoření reprezentací zdrojů a jejich aktualizací hodnot v závislosti na činnosti serveru.

11. V hlavičce inicializujeme ve veřejné třídě *Klíče*, podle nichž jsou hodnoty při komunikaci mezi sebou rozpoznány. Klíče jsou upraveny dle našich parametrů.

```
public class SensorsRes {
public static final String JMENO_KLIC =
   "jmenoKlice";
```

Jak je možné vidět v importovaných knihovnách, využívá se knihoven např:

Výpis 3.9: Importování knihoven

```
import org.iotivity.base.OcException;
import org.iotivity.base.OcRepresentation;
import org.iotivity.base.OcResource;
import org.iotivity.base.OcPlatform;
```

pro spolupráci s rozhraním IoTivity, které předává OC reprezentace. Ty umožňují komunikaci v IoTivity. Klient začíná ve funkci *onCreate()*, kde se inicializují *GUI* objekty v propojení s *layoutem* v *activity_mail.xml*. Tlačítku start je přiřazena funkce:

```
private void startClient(OcConnectivityType type)
```

V této funkci proběhne inicializace IoTivity a hledání zdrojů odpovídajících *URI* v našem případě: *core.raspberry*. Další funkce pro hledání zdrojů implementují rozhraní *OnResourceFoundListener*, kde

- **onResourceFound()** v případě úspěšného nalezení bude získán objekt *resource*, se ktrým se dále pracuje.
- **onFindResourceFailed()** v případě neúspěšného nalezení.

Pro tlačítka jsou namapované funkce:

- **Update**
  Posílá ***GET*** žádost na Server pro oba nalezené zdroje

  ```
  getSensorsResourceRepresentation();
  getTempLimitsResourceRepresentation();
  ```

  Ty se vypisují do GUI pomocí *textView_sensors*. Pro zpracování GET requestu je potřeba implementovat rozhraní ***OnGetListener***:
  onGetCompleted úspěšný dotaz *GET* server poslal odpověď s reprezentací dotazovaného zdroje, který je rozlišen pouze podle *URI*

Výpis 3.10: Identifikace URI

```
String uri = ocRepresentation.getUri();
if(uri.contentEquals("/raspberry/sensors")) {
 ...
```

```
}
if(uri.contentEquals("/raspberry/templimits")){
  ...
}
```

Žádost *onGetFailed GET* selhala.

Další tlačítka jsou pro nastavení režimu topení *AUTO / MANUAL*, kde pro manuální režim je tlačítko *HEATING* neboli topení povoleno používat. Tlačítko *set* pro nastavení teplotních limitů. Tyto tlačítka používají žádost *PUT*, pomocí které předávají zdroje serveru. Pro zpracování *PUT* žádosti je potřeba implementovat rozhraní ***OnPutListener:***

- **onPutCompleted** - *PUT* metoda byla úspešná, vyčteme aktuální údaje z klienta.
- **onPutFailed** při neúspěchu metody vyvolá chybový výpis.

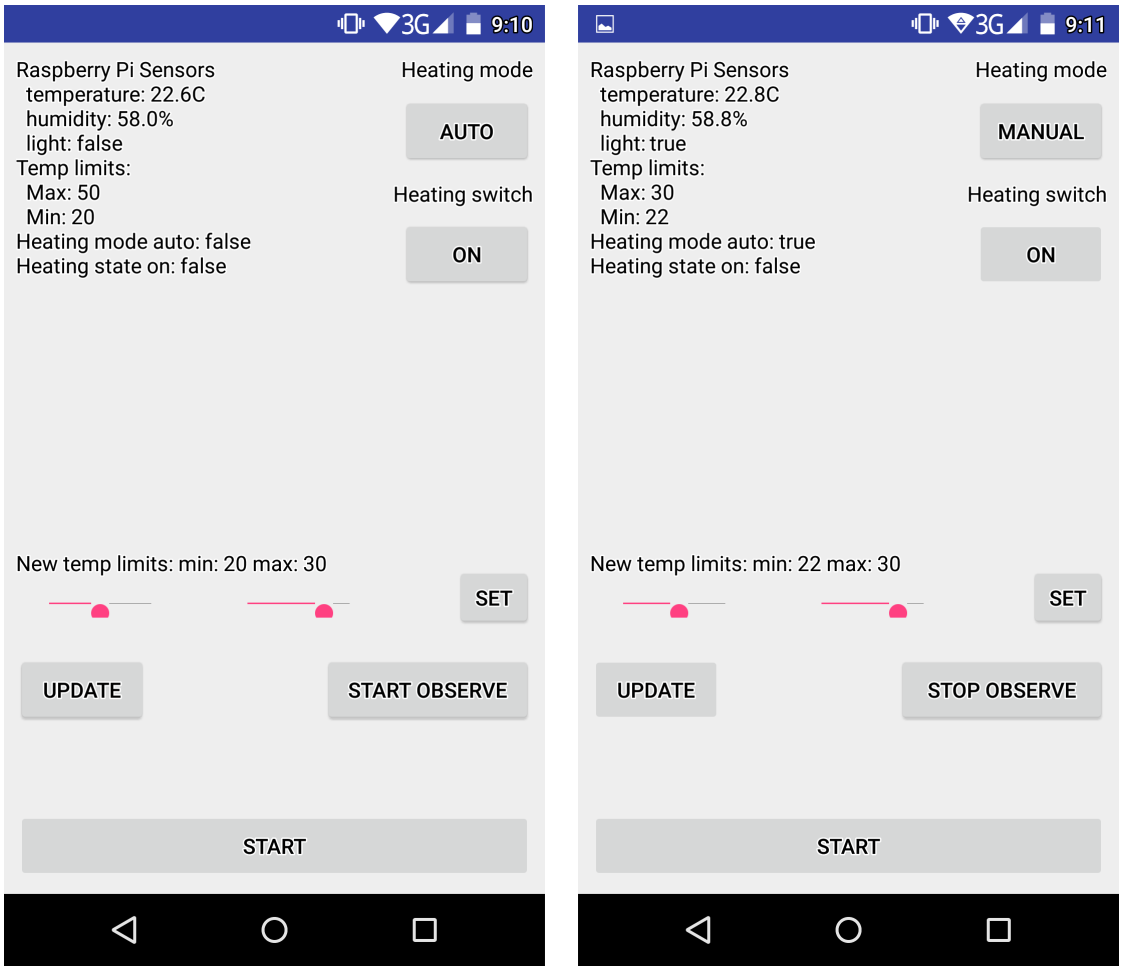
Pro spuštění *OBSERVE* režimu je zde funkce: *startObserve()*, která volá nad danými zdroji funkci pro pozorování.

Výpis 3.11: Nalezení podle klíčů pro OBSERVE

```
new HashMap<String, String>(), this);
mFoundHeatingResource.observe(ObserveType.OBSERVE,
new HashMap<String, String>(), this);
```

A rovněž je potřeba implementovat rozhraní pro observe funkcionalitu: *OnObserveListener*

- onObserveComplete ze jsou serveru přijímány informace o zdroji
- onObserveFailed při neúspěchu vypíše chybové hlášení



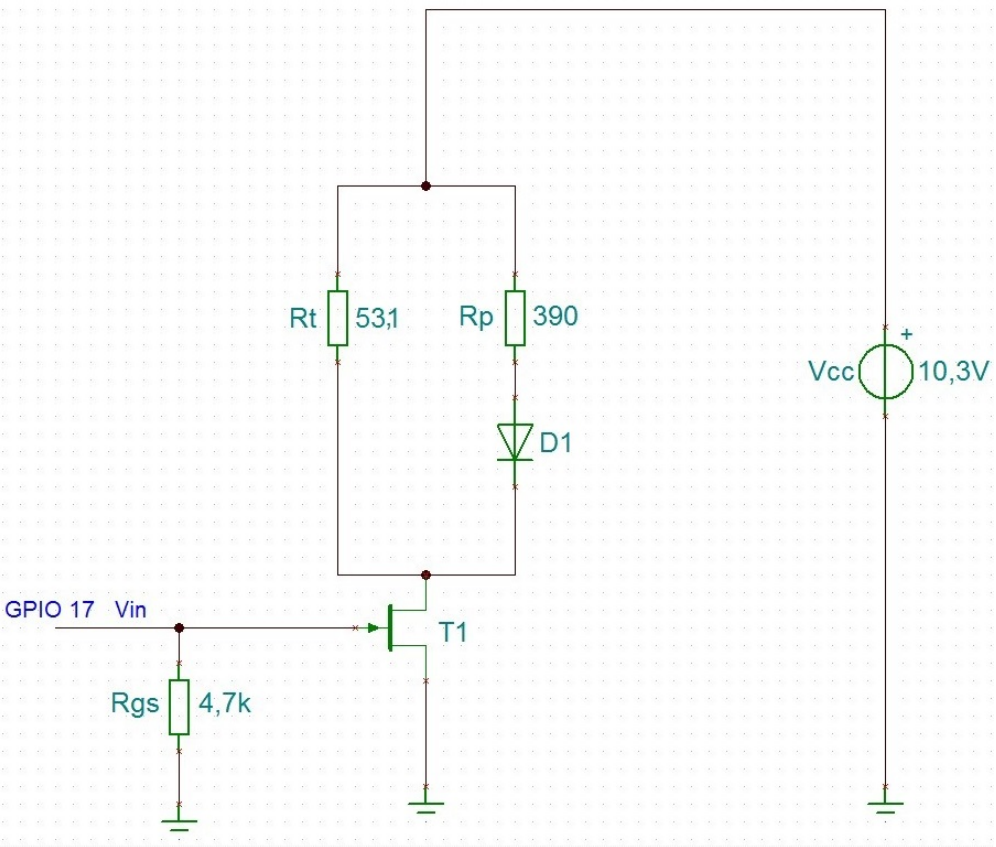
Obr. 3.11: Komunikace klienta Android

3.5 Zapojení Raspberry Pi

Raspberry Pi disponuje *GPIO* piny pro I/O komunikaci s rozhraním. Využito bude napájení o hodnotě 3,3V a stejná hodnota napětí signalizuje stav *HIGH* a 0V *LOW*. Pro práci bylo zapojení realizováno následujícím schématem 3.14

- *DHT22* neboli *AM2302*, což je senzor snímající teplotu od *-40-80°C* a vlhkost vzduchu v rozsahu *0-100%* s odchylkou *0,5°C a 2%*. Datový výstup senzoru byl realizován na *GPIO 4* neboli *pin7*. Napětí je dodáváno přes *VCC*=3,3V pinem *1* a uzemněn.
- Fotorezistor pro snímání intenzity světla s nastavitelnou citlivostí byl zapojen na datový pin *GPIO 27*. Napájen opět *VCC*=3,3V.
- Topení je realizováno odporovým drátem o celkové hodnotě 53,1Ωovinutého kolem tepelného senzoru. Topení je spínáno pomocí tranzistoru *MOSFET bts117* typu *N-channel* zapojeného jako spínač. Spínací napětí je přivedeno na *Gate*,

který má spínací napětí 2,2V. Zároveň je stav zapnutého topení realizováno červenou diodou s předřadným odporem o hodnotě 390Ωv paralelním zapojení k topení. Tento obvod je napájen externím zdrojem o hodnotě 10,3V stejnosměrného napětí, realizované *NiCd* článkovou baterií. Spínání tranzistoru je hlídáno pomocí Pulldown rezistoru o jmenovité hodnotě 4.7kΩ. Schéma zapojení obvodu topení pomocí spínače obr. 3.14



Obr. 3.12: Schéma zapojení topného obvodu pomocí N-tranzistoru

Ovládání těchto I/O periférií se odehrává prostřednictvím žádostí klienta na server, kde server vnitřní funkcionalitou tyto úkony vykonává.

- **Topení**

  Spouštění topení vykonává binární soubor napsaný v jazyce *C*. Pro ovládání *GPIO* výstupu se ovládá prostřednictím knihovny *wiringPi.h* [49]. Příkazem se ověří, zda se na zařízení nachází tato podpora

  ```
  gpio -v
  ```

  Je obdržen výpis, že je již instalovaná. Pro výpis přehledu pinů a jejich funkcí je příkaz:

  ```
  gpio readall
  ```

Použije se volný pin označen ve *wiringPi* indexem *0* pro *OUT* výstup spínáním. Na desce je to pin *GPIO 17*.

```
pi@raspberrypi:~/Desktop $ gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |  OUT | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
```

Obr. 3.13: wiringPi GPIO

```
#include <wiringPi.h>
int main (void)
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  digitalWrite (0,  HIGH) ;
  return 0 ;
}
```

Nyní je třeba skript zkompilovat pomocí kompilačního nástroje *gcc* pro jazyky *C/C++* s parametry odkazující na knihovnu *wiringPi*.

```
gcc -Wall -o vystup vstup.c -lwiringPi
sudo ./vystup
```

- **Čtení dat - AM2302**

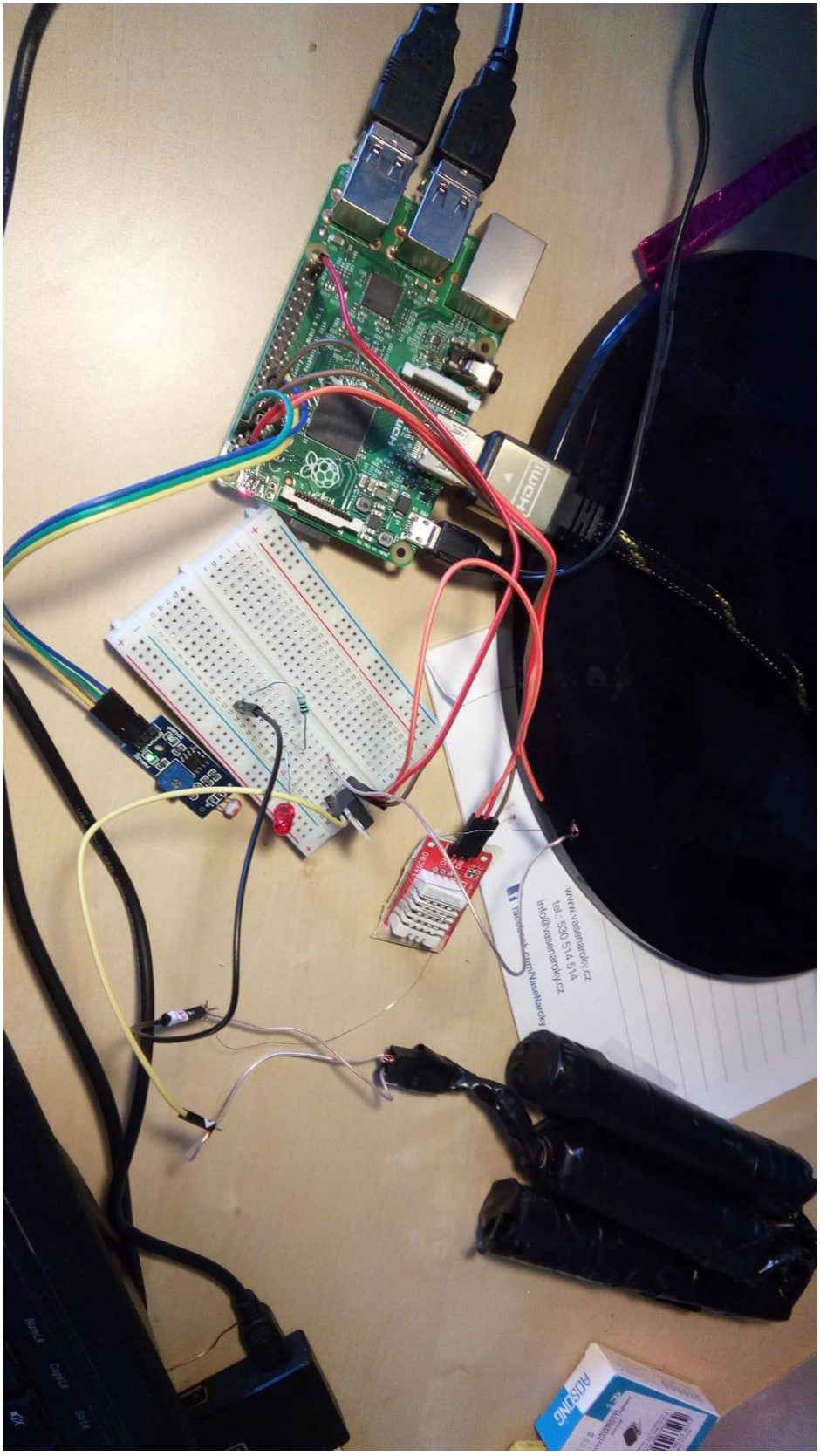
  Čtení ze senzorů je realizováno pomocí skriptů v jazyce *Python*. Pro čtení z tepelného senzoru *AM2302* je implementována knihovna pro čtení dat [48]. Pomocí spuštění skriptu *setup.py* uvnitř složky se nalinkuje cesta pro konkrétní knihovny. Vyčtená data se zapíšou do souboru pomocí parametru *w+*, který

vytvoří / edituje existující soubor pro zápis a čtení. z tohoto souboru probíhá výčet serverem.

- **Čtení dat - fotorezistoru**
  Čtení je realizováno pomocí knihovny *GPIO*.

```
import RPi.GPIO as GPIO
GPIO_PIN = 27
GPIO.setmode(GPIO.BCM)
GPIO.setup(GPIO_PIN, GPIO.IN)
VALUE_PATH = "/home/pi/Desktop/svetlo.value"
if GPIO.input(GPIO_PIN):
        try:
            with open(VALUE_PATH,'w+') as file:
                file.write('0')
```

  Čtení probíhá logikou true / false a zápis se podle toho formátuje pro textový soubor *svetlo.value* pro server.



Obr. 3.14: Zapojení obvodu na Raspberry Pi

4 ZÁVĚR

V práci byl popsán framework IoTivity včetně jeho implementací jako jsou komunikační protokoly na nichž staví různá podporovaná rozhraní a možnosti použití. IoTivity funguje na komunikačním protokolu CoAP, který je obdobou HTTP. Z tohoto důvodu je nejčastěji využito zapojení typu Klient - Server. IoTivity umožňuje vývojářům soustředit se na vývoj aplikací bez nutnosti zabývat se vzájemného navázání spojení. Sestavení komunikace probíhá skrze rozhraní IoTivity na transportní vrstvě. Využívá pro to knihovny typu *OC* a implementuje jejich funkce. Nalezení a komunikace včetně klíčů je vedeno v podobě tzv. textitResources neboli zdroji, které slouží jako identifikátory. Zařízení se k sobě registrují na základě *URI* klíče. Takto spárované zahájí přenos užitečných dat.

V praktické části byla otestována komunikace modelu Server - Klient, pro niž byla vytvořena aplikace simulující jednoduché domácí prostředí pomocí senzorů pro teplotu, vlkhost a světelného čidla. Realizováno bylo i vytápění pomocí odporového drátu. Tyto parametry dostává klient (telefon s OS Android, Soc ARM). Jejich ovládání je řízeno serverem platformy zařízení Raspberry Pi 2B s operačním systémem Linux a SoC ARM. Pro testování sloužil rovněž počítač s procesorem řady *intel core i7* se systémem Linux.

Během práce byl pozorován vývoj projektu IoTivity na jejich domovských stránkách, kde byly přidány další podporovaná zařízení, upraveny seznamy balíčků pro spuštění a sjednoceny postupy pro nastavení závislostí pro operační systém.

Seznámení s rozhraním a jeho implementací probíhá pomocí zdrojových kódů, příkladů přiložených v aktuálních verzích vydání a dokumentovaných knihoven.

Z vypracování této práce lze konstatovat funkčnost projektu IoTivity a jeho možnou implementaci na různých hardwarových platformách, na kterých byla úspěšně spuštěna komunikace.

LITERATURA

[1] Iotivity. *Iotivity* [online]. 2016 [cit. 2016-11-12]. Dostupné z URL: www.iotivity.org

[2] Linux Foundation. *Linux Foundation* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.linuxfoundation.org

[3] CBOR. *CBOR-protokol* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.tools.ietf.org/html/rfc7049

[4] Linux. *Linux* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.linux.com

[5] Protokol. *Protokol* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: http://searchnetworking.techtarget.com/definition/protocol

[6] Arduino. *Arduino* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.arduino.cc

[7] Android. *Android* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.android.com

[8] Tizen. *Tizen* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.tizen.org

[9] Yoctoproject. *Yoctoproject* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.yoctoproject.org

[10] Techterms. *Techterms* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.techterms.com

[11] Techtarget. *Techtarget* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.techtarget.com

[12] Tutorialspoint. *Tutorialspoint* [online]. 2015 [cit. 2016-11-12]. Dostupné z URL: www.tutorialspoint.com

[13] ESP. 2015. *ESP* [online]. [cit. 2016-11-17]. Dostupné z URL: www.esp8266.net

[14] Electronicdesign. 2015. *Electronicdesign* [online]. James Stansberry [cit. 2016-11-20]. Dostupné URL: www.electronicdesign.com

[15] Brillo. 2015. *Brillo* [online]. [cit. 2016-11-20]. Dostupné z URL: www.brillo.com

[16] CoAP. 2014. *Coap.technology* [online]. [cit. 2016-11-17]. Dostupné z URL: www.datatracker.ietf.org/doc/rfc7252/?include_text=1

[17] MQTT.2015. *MQTT* [online]. [cit. 2016-11-17]. Dostupné z URL: www.electronicdesign.com/

[18] SIP.2015. *SIP* [online]. [cit. 2016-11-27]. Dostupné z URL: www.pyqt.sourceforge.net/Docs/sip4/introduction.html

[19] OCF.2015. *OpenConnectivityFoundation* [online]. [cit. 2016-11-27]. Dostupné z URL: www.openconnectivity.org/resources/specifications

[20] OSGI .2015. *Open Services Gateway initiative* [online]. [cit. 2016-12-2]. Dostupné z URL: www.osgi.org

[21] IoTivity .2015. *Instalační balíčky* [online]. [cit. 2016-12-12]. Dostupné z URL: www.iotivity.org/documentation/linux/getting-started

[22] Fielding, Roy Thomas Representational State Transfer [online]. 2000 [cit. 2016-11-12]. Dostupné z URL: www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.html

[23] IoTivity .2015. *iconlabs* [online]. [cit. 2016-12-12]. Dostupné z URL: www.wiki.iotivity.org/build_for_your_system

[24] Security .2015. *Instalační balíčky* [online]. [cit. 2017-02-11]. Dostupné z URL: www.iconlabs.com/prod/internet-secure-things-%E2%80%93-what-really-needed-secure-internet-things

[25] Embedded .2015. *Embedded* [online]. [cit. 2017-02-11]. Dostupné z URL: www.internetofthingsagenda.techtarget.com/definition/embedded-system

[26] JSON .2015. *JSON* [online]. [cit. 2017-02-11]. Dostupné z URL: www.json.org/json-cz.html

[27] RAML .2015. *RAML* [online]. [cit. 2017-02-11]. Dostupné z URL: http://www.slideshare.net/ShankyGupta7/raml-55283821

[28] Galileo .2015. *Galileo* [online]. [cit. 2017-05-11]. Dostupné z URL: https://www.arduino.cc/en/ArduinoCertified/IntelGalileoGen2

[29] Raspberry 2 .2015. *Raspberry* [online]. [cit. 2017-05-11]. Dostupné z URL: http://www.trustedreviews.com/raspberry-pi-2-review

[30] VirtualBox .2015. *Virtualbox* [online]. [cit. 2017-05-12]. Dostupné z URL: https://www.virtualbox.org/wiki/Downloads

[31] Ubilinux .2015. *Ubilinux* [online]. [cit. 2017-05-17]. Dostupné z URL: https://owncloud.cesnet.cz/index.php/s/N21k6Cl55PBvSQg

[32] Intel-galileo .2015. *Intel-Galileo* [online]. [cit. 2017-05-17]. Dostupné z URL: http://downloadmirror.intel.com/23962/eng/Quark_BSP_Buildand-SWUserGuide_329687_007.pdf

[33] Edk2-tiancore .2015. *Edk2-git* [online]. [cit. 2017-05-17]. Dostupné z URL: https://github.com/tianocore/edk2

[34] Edk2-tiancore .2015. *Edk2-git* [online]. [cit. 2017-05-17]. Dostupné z URL: https://github.com/tianocore/edk2-staging

[35] Acpica-tools .2015. *Acpica-tools* [online]. [cit. 2017-05-17]. Dostupné z URL: https://www.acpica.org/downloads/linux

[36] BSP .2015. *BSP* [online]. [cit. 2017-05-17]. Dostupné z URL: https://downloadcenter.intel.com/download/23197

[37] Ask Ubuntu svn .2015. *SVN missing* [online]. [cit. 2017-05-17]. Dostupné z URL: http://askubuntu.com/questions/55546/how-do-i-install-svn

[38] OpenSSL .2015. *OpenSSL* [online]. [cit. 2017-05-17]. Dostupné z URL: http://www.openssl.org/source/openssl-0.9.8zb.tar.gz

[39] OpenSSL-patch .2015. *OpenSSL-patch* [online]. [cit. 2017-05-17]. Dostupné z URL: http://directory.fsf.org/project/patch/

[40] OpenSSL-patch-ftp .2015. *OpenSSL-patch-ftp* [online]. [cit. 2017-05-17]. Dostupné z URL: ftp://ftp.gnu.org/gnu/patch/

[41] Software-intel-yocto .2015. *Intel-yocto* [online]. [cit. 2017-05-17]. Dostupné z URL: https://software.intel.com/en-us/blogs/2015/03/04/creating-a-yocto-image-for-the-intel-galileo-board-using-split-layers

[42] Software-intel-návod .2015. *Bitbake-galileo* [online]. [cit. 2017-05-17]. Dostupné z URL: https://downloadmirror.intel.com/24355/eng/BSP-Patches-and-Build_Instructions.1.0.8.txt

[43] Software-intel-návod-pdf .2015. *Bitbake-galileo* [online]. [cit. 2017-05-17]. Dostupné z URL: https://downloadmirror.intel.com/24748/eng/IntelGalileo FirmwareUpdaterUserGuide-1.0.4.pdf

[44] Raspibian-download .2015. *Raspibian* [online]. [cit. 2017-05-20]. Dostupné z URL: https://www.raspberrypi.org/downloads/raspbian/

[45] Raspibian-format .2015. *Raspibian* [online]. [cit. 2017-05-20]. Dostupné z URL: https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

[46] IoTivity Raspberry .2015. *Raspberry Dep* [online]. [cit. 2017-05-20]. Dostupné z URL: https://wiki.iotivity.org/build_iotivity_for_raspberry_pi

[47] SDK instalace .2015. *SDK-Instalace* [online]. [cit. 2017-05-17]. Dostupné z URL: https://developer.android.com/studio/install.html

[48] Adafruit-DHT22 .2015. *Adafruit-DHT22* [online]. [cit. 2017-05-18]. Dostupné z URL: https://github.com/adafruit/Adafruit_Python_DHT

[49] Wiring-Pi .2015. *wiringPi.h* [online]. [cit. 2017-05-18]. Dostupné z URL: http://wiringpi.com

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API	Application Programming Interface
XML	Extensible Markup Languagen
OIC	Open interconnect consortium
IoT	Internet of Things - zařízení komunikující skrz internet
OCF	Open connectivity foundation
HTTP	Hypertext Transfer Protocol - zajištuje přenos HTML
HTML	HyperText Markup Language - jazyk pro tvoření webu
SOAP	Simple Object Access Protocol)- protokol pro výměnu XML zpráv
URI	Uniform Resource Identifier - označení indetifikátoru zdroje
ACK	Acknowledgement - u paketů se jedná o potvrzení přijetí packetu
C	Programovací jazyk
C++	Programovací jazyk
Java	Objektově orientovaný programovací jazyk
ESP	Programovatelný modul s výpočetní jednotkou
CoAP	Constrained Application Protocol - odlehčené HTTP
SoC	System on Chip - Označení architektury procesoru

SEZNAM PŘÍLOH

A UKÁZKA KÓDŮ

Výpis A.1: Výpis příkladů zkompilovaných pro linux

```
~/iotivity-master/out/linux/x86_64/release/resource/examples\$ ls
devicediscoveryclient                presenceclient
devicediscoveryclient.o              presenceclient.o
devicediscoveryserver                presenceserver
devicediscoveryserver.o              presenceserver.o
directpairingclient                  rdclient
directpairingclient.o                rdclient.o
fridgeclient                         roomclient
fridgeclient.o                       roomclient.o
fridgeserver                         roomserver
fridgeserver.o                       roomserver.o
garageclient                         simpleclient
garageclient.o                       simpleclientHQ
garageserver                         simpleclientHQ.o
garageserver.o                       simpleclient.o
groupclient                          simpleclientserver
groupclient.o                        simpleclientserver.o
groupserver                          simpleserver
groupserver.o                        simpleserverHQ
lightserver                          simpleserverHQ.o
lightserver.o                        simpleserver.o
oic_svr_db_client.dat                threadingsample
oic_svr_db_client_directpairing.dat  threadingsample.o
oic_svr_db_server.dat
```

Výpis A.2: Výpis komunikace simpleserver

```
./simpleserver
Usage : simpleserver <value>
    Default - Non-secure resource and notify all observers
    1 - Non-secure resource and notify list of observers
    2 - Secure resource and notify all observers
    3 - Secure resource and notify list of observers
    4 - Non-secure resource, GET slow response, notify all observers
Starting server & setting platform info
Created resource.
Added Interface and Type
Waiting
```

Výpis A.3: Výpis příkladů zkompilovaných pro linux

```
0:
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : GET
0:
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : PUT
        state: 1
        power: 15
0:
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : POST
        state: 1
        power: 55
0:
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : GET
    requestFlag : Observer
0:
In entity handler wrapper:

  In Server CPP entity handler:
    requestFlag : Request
      requestType : GET
    requestFlag : Observer

Power updated to : 165
Notifying observers with resource handle: 0x2028c50
No More observers, stopping notifications
```

Výpis A.4: Část kódu funkce readSensor()

```
int status;
if(heatingResPtr->m_state_on)
{
    status = system("/home/pi/Desktop/one");
}
else
{
    status = system("/home/pi/Desktop/zero");
}
if(heatingResPtr->m_auto_mode)
{
    // automatic mode - should hold temperature in limits
    if(sensorResPtr->m_temperature <= heatingResPtr->m_min_temp)
    {
        heatingResPtr->m_state_on = true;
    }
    if(sensorResPtr->m_temperature >= heatingResPtr->m_max_temp)
    {
        heatingResPtr->m_state_on = false;
    }
}
```

B PŘÍLOHY CD

Na CD je uložena složka *iotivity-1.2.1* z Raspberry Pi včetně kódů jak v binárním stavu tak před kompilací a Python skriptů. Dále se na CD nachází aplikace pro Android včetně celé složky pro Android studio a bakalářská práce ve formátu *.pdf*.