
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION


# INTERNET VĚCÍ PRO INTELIGENTNÍ DOMÁCNOST
INTERNET OF THINGS FOR SMART HOME

**HABILITAČNÍ PRÁCE**
HABILITATION THESIS


**AUTOR PRÁCE** **Ing. PETR ČÍKA, Ph.D.**
AUTHOR


**BRNO 2017**

## ABSTRAKT

Habilitační práce se věnuje problematice informačních systémů z oblasti tzv. Internetu věcí (Internet of Things, IoT) pro elektrická zařízení a spotřebiče používané v domácnostech. Konkrétně je zaměřena na komunikaci chytrých domácích serverů na aplikační vrstvě síťového modelu TCP/IP. V práci jsou podrobně popsány známé aplikační protokoly Constrained Application Protocol (CoAP) a Message Queue Telemetry Transport (MQTT) využívané v sítích IoT. Dále je popsán signalizační protokol Session Initiation Protocol (SIP) původně určený k signalizaci v oblasti internetové telefonie. Protokol SIP je podrobně analyzován z hlediska využití v prostředí IoT. Z analýzy protokolu SIP jsou vyvozeny výsledky v podobě doporučení pro návrh modelu komunikace a způsobu komunikace IoT zařízení. Dále jsou v práci zkoumány možnosti využití softwarových struktur pro vývoj softwarové části chytrého domácího serveru. Výkonnost vybraných softwarových struktur byla ověřena na vybraném hardwaru za použití původních experimentálních testů, výsledky analýz jsou uvedeny v přehledných grafech. Pro vybranou softwarovou strukturu byl navržen hardware chytrého domácího serveru tak, aby splňoval minimální požadavky zvolené implementace softwarové struktury. Cílem byl návrh koncepce chytrého domácího serveru s využitím protokolu SIP, jeho realizace a experimentální ověření teoretického návrhu. Problémy, které vznikly během experimentálního ověření, byly analyzovány a odstraněny. Návrhy na zdokonalení jsou v práci podrobně zdokumentovány. Přínosem systematické části práce je úvod do problematiky komunikace v sítích IoT, zvláště pak do komunikace na aplikační vrstvě TCP/IP modelu. Vědecká část práce obsahuje teoretické poznatky z oblasti Internetu věcí a jejich experimentální ověření, zejména využití protokolu SIP pro komunikaci zařízení v prostředí IoT.



## KLÍČOVÁ SLOVA

chytrý domácí server, Internet věcí, OSGi, SIP

## ABSTRACT

This habilitation thesis deals with problematics of information systems, so called Internet of Things (IoT) in electronic devices used at homes. Particularly, the thesis focuses on smart home server communications on application layer of TCP/IP. Next, Constrained Application Protocol (CoAP) and Message Queue Telemetry Transport (MQTT) used in IoT networks are described in detail. Session Initiation Protocol (SIP) designed to internet telephony is analyzed for use in IoT networks. The results from SIP analysis are used for recommendation of design of communication model for IoT devices. Furthermore, the work examines software frameworks for the development of the smart home server. Performance of the selected frameworks is verified using the selected hardware in the original experimental tests, the results of analyses are presented in synoptic charts. Smart home server hardware is designed for the selected framework. The designed hardware meets minimum requirements of the selected framework implementation. Proposal of a concept of a smart home server using the SIP protocol, its implementation and experimental verification of theoretical design are the main aims of this thesis. All problems that arose during the experimental verification are analyzed and removed. Suggestions for improvement are described in detail. Introduction to the problems of IoT communication networks, especially in communications at the application layer of TCP / IP model is the benefit of systematic part of this work. The scientific part of the work contains theoretical knowledge of the IoT and their experimental verification, in particular the use of the SIP protocol for communication of devices in the IoT.

## KEYWORDS

smart home server, Internet of things, OSGi, SIP

ČÍKA, Petr *Internet věcí pro inteligentní domácnost*: habilitační práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2017. 112 s.

# PROHLÁŠENÍ

Prohlašuji, že svou habilitační práci na téma „Internet věcí pro inteligentní domácnost“ jsem vypracoval samostatně s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené habilitační práce dále prohlašuji, že v souvislosti s vytvořením této habilitační práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom 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

# PODĚKOVÁNÍ

Rád bych poděkoval svým kolegům z výzkumné skupiny zabývající se Internetem věcí (IoT) na Ústavu telekomunikací, FEKT, Vysokého učení technického v Brně, zejména doc. Ing. Jiřímu Hoškovi, Ph.D. a Ing. Pavlovi Maškovi, za týmovou spolupráci při výzkumu a vývoji v oblasti Internetu věcí. Dále děkuji všem ostatním kolegům z mého pracoviště za morální podporu při zpracování práce.
Velké díky patří také celé mé rodině, hlavně manželce Nikole, dcerám Adélce a Anetce. Bez jejich pochopení a tolerance ve vypjatých chvílích, zvláště při dokončování práce, bych práci nemohl dokončit.

Brno ..... .....

podpis autora

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# ÚVOD

Komunikace mezi stroji M2M (Machine-to-Machine) se v posledních letech velmi rychle rozvíjí a zasahuje do mnoha průmyslových oblastí [1]. Je velmi často využívána ke kontrole, monitorování či optimalizaci určitého procesu a k okamžitému řešení možného havarijního stavu, který během procesu nastane. Vznikají komplexní systémy spojující různé měřicí senzory a zařízení, které jsou jako celek zpravidla dále propojeny pomocí sítě Internet. Tyto systémy mají schopnost měřit fyzikální veličiny (např. intenzitu osvětlení, spotřebu elektrické energie, teplotu, vlhkost apod.), získávat informace o stavu zařízení a na základě těchto dat provádět různé úkoly či inteligentně rozhodovat o svém dalším fungování. Propojení M2M zařízení s Internetem se nazývá Internet věcí (IoT – Internet of Things) nebo také Internet všeho (IoE – Internet of Everything) [2], [3]. Vzhledem k velmi široké oblasti využití se IoT dále dělí na cIoT (Consumer IoT) spadající do oblasti spotřební elektroniky a iIoT (Industrial IoT) spadající do oblasti průmyslu. IoT je stejně jako M2M rychle se rozvíjející oblast, což dokládá mimo jiné studie společnosti Cisco Systems, Inc., jedné z významných společností působících v oblasti IoT, která predikuje 3,1 bilionů funkčních zařízení M2M a IoT do roku 2021 [4].

Společně s rozvojem technologií používaných ke komunikaci M2M a IoT vznikají nová doporučení a standardy definující způsoby komunikace mezi systémy, způsoby sběru dat, jejich analýzy a vyhodnocení. Systémy IoT tedy definují hardware, software a prostředky vhodné ke komunikaci mezi jednotlivými zařízeními. Jedním z problémů rychlého rozvoje je vzájemná kompatibilita zařízení. Na trhu je řada otevřených i uzavřených řešení, které jsou často, pokud se týká komunikace, vzájemně nekompatibilní. Je tak zaznamenán obrovský tlak na vývoj otevřených řešení a standardů vedoucí ke vzájemné interoperabilitě zařízení.

# 1 PŘEHLED PRÁCE

## 1.1 Motivace

Technologie IoT definuje síť vzájemně propojených zařízení skrze síť Internet a je velmi často spojována se čtvrtou průmyslovou revolucí. Kromě průmyslových odvětví (automobilový, strojírenský apod.) se IoT velmi rychle prosadilo v oblasti spotřební elektroniky. Zařízení splňující charakteristiku IoT se často nazývají jako „chytrá zařízení“. Prakticky se jedná o fyzická zařízení (rádio, hodinky, mobilní telefon, automobil, budova apod.) osazené elektronikou (senzory, měřicími systémy), která umožňují shromažďovat a vzájemně si vyměňovat informace [5]. Vzhledem k širokému spektru využití se IoT dělí dle oblasti využití na cIoT (Consumer IoT), pokrývající oblast spotřební elektroniky, a iIoT (Industrial IoT), spadající do oblasti průmyslu.

**Oblast iIoT** je reprezentována aplikacemi zaměřenými na průmyslové prostředí, kde:

- zařízení a stroje pracují například v průmyslu, dopravě, energetice,
- objemy přenášených dat mezi zařízeními jsou relativně velké,
- funkce aplikací jsou důležité a jejich selhání může mít významný vliv na život lidí, ekonomiku, v případě dopravy může ohrozit lidské životy apod.

**Oblast cIoT** je reprezentována aplikacemi orientovanými na koncové uživatele, kde:

- zařízení jsou z oblasti spotřební elektroniky, mezi něž patří inteligentní spotřebiče - chytrá lednice, chytrá pračka, chytrá sušička, chytré hodinky, chytré brýle apod.,
- objemy přenášených dat mezi zařízeními jsou relativně malé,
- funkce aplikací nejsou tak důležité jako u iIoT, jejich selhání nezpůsobí fatální následky v žádné oblasti.

Jedna z podoblastí cIoT je domácnost. Domácnosti můžou být v současné době vybaveny celou řadou IoT zařízení/senzorů, mezi něž patří například chytré měřicí systémy (vodoměry, elektroměry, teploměry, vlhkoměry apod.), chytrá stínící technika, chytré ledničky, pračky, sušičky, chytré osvětlení, chytré zabezpečovací systémy apod. Společnou vlastností všech zmíněných zařízení/systémů je, že mohou být řízeny/sledovány vzdáleně přes Internet, automaticky rozhodovat o svém fungování a poskytovat různé informace [6].

Automatizace domácností spojená s IoT se neustále a vysokým tempem vyvíjí a vzniká celá řada různých pohledů na uvedenou problematiku. Existují a vznikají unikátní řešení chytrých domácností, ve většině případů nestandardní a uzavřená [6], [7], [8], [9]. Jedním ze zásadních problémů existujících a nově vyvíjených systémů je jejich vzájemná interoperabilita.

V běžné síti Internet je interoperabilita základní vlastností. Každé připojené zařízení do sítě Internet by mělo být schopné porozumět ostatním zařízením. K tomu existují standardizované protokoly definované prostřednictvím standardizační skupiny IETF (Internet Engineering Task Force).

V ideálním interoperabilním prostředí v sítích IoT by tedy všechna zařízení byla schopná vzájemné komunikace a výměny informací [10]. V reálném prostředí se interoperabilita mezi zařízeními IoT vyskytuje pouze v určité míře v jednotlivých vrstvách síťového modelu TCP/IP. Úplná interoperabilita napříč všemi zařízeními IoT není vždy možná [10].

K návrhu koncepce interoperabilního prostředí v IoT směřují významné organizace tvořící standardy a doporučení pro komunikaci v IoT a M2M. Jedná se zejména o společnosti oneM2M [11], Allseen Alliance [12] a IoTivity [13]. Zmíněné organizace provádějí vývoj v oblasti IoT a M2M za podpory svých členů, mezi něž patří světoví lídři na trhu elektroniky, strojírenství, senzorů ale i mobilních operátorů a společností poskytujících datová připojení (Microsoft, Qualcomm, Ericsson, Huawei a další). Seznam členů společnosti oneM2M je uveden v [14], Allseen Alliance v [15]. Hlavním cílem výzkumů je vyvinutí jednotné komunikační platformy pro IoT schopné vzájemně propojit současné i nově vyvinuté systémy.

Vedle technických aspektů je vhodné zmínit, že schopnost systémů vzájemně si poskytovat služby a efektivně spolupracovat má velký vliv na ekonomiku. Dobře fungující a dobře definovaná interoperabilita mezi zařízeními může podpořit inovace a poskytnout efektivitu pro výrobce zařízení IoT, což zvyšuje celkovou ekonomickou hodnotu na trhu [16].

## 1.2 Cíle

Habilitační práce se věnuje interoperabilitě v oblasti cIoT na aplikační vrstvě modelu TCP/IP a vzájemné komunikaci chytrých domácích serverů v rámci síťových infrastruktur využívaných telekomunikačními operátory. Hlavní cíle práce lze shrnout do následujících bodů:

- **Analýza modelů komunikace v IoT**

  Při vývoji systémů IoT je velmi důležité dopředu stanovit a navrhnout postup při komunikaci jednotlivých zařízení. Architektura sítě IoT je základem dobrého fungování a snadné rozšiřitelnosti systémů. Jedním z cílů práce je představení dostupných modelů komunikace a jejich zhodnocení.
- **Popis aplikačních protokolů vhodných pro IoT**

  V současné době existuje celá řada aplikačních protokolů, které jsou primárně určeny ke komunikaci v sítích M2M či IoT. Výběr vhodného aplikačního protokolu ke komunikaci v sítích IoT může zásadně ovlivnit interoperabilitu IoT systémů. Dílčím cílem práce je popsat nejvýznamnější aplikační protokoly určené pro IoT, analyzovat je a navzájem je porovnat či provázat.
- **Návrh komunikace v IoT vhodné pro využití v sítích telekomunikačních operátorů**

  Kromě standardních aplikačních protokolů existují i takové, které nejsou přímo navržené ke komunikaci IoT zařízení. Mezi ty patří například signalizační protokoly používané pro internetové telefonování, posílání textových / multimediálních zpráv apod. Mezi významný protokol na aplikační vrstvě síťového modelu TCP/IP, který je využívaný zejména v internetové telefonii, patří signalizační protokol SIP. Je otázkou, zda-li se tento protokol dá použít pro sítě IoT a za jakých podmínek. K hlavním cílům práce proto patří návrh komunikace chytrého domácího serveru s využitím protokolu SIP tak, aby byla splněna podmínka interoperability se síťovou infrastrukturou telekomunikačních operátorů. Chytrý domácí server propojuje a řídí domácí automatizované prvky (rolety, žaluzie, markýzy,...), spotřebiče či měřicí systémy. Získaná data systém zpracovává, vyhodnocuje a může je odesílat k dalšímu zpracování poskytovateli služby. Chytrý domácí server je ve své podstatě centrálním prvkem chytré domácnosti, který komunikuje jak s chytrými zařízeními domácnosti, tak i s prvky umístěnými kdekoli v síti Internet.
- **Analýza volně dostupných softwarových struktur pro vývoj IoT systémů, experimentální ověření jejich výkonnosti**

  Při návrhu systému IoT lze vycházet z dostupných komerčních řešení, které jsou ve většině případů proprietární, nebo je možné vytvořit zcela unikátní a otevřený systém s možností jakéhokoli rozšíření. K tomu je vhodné použít volně dostupné softwarové struktury (frameworky) pro vývoj systémů IoT. Dílčím cílem práce je popis významných softwarových struktur využívaných pro vývoj IoT systémů a experimentální ověření jejich výkonnosti.

- **Experimentální ověření navržené komunikace pro chytrý domácí server**
  Hlavním cílem habilitační práce je získání teoretických poznatků v oblasti Internetu věcí a dále experimentální ověření interoperability komunikace navrženého řešení chytrého domácího serveru s infrastrukturou telekomunikačních operátorů.

## 1.3 Přínos práce

Přínos práce lze rozdělit na systematický a vědecký. Systematická část práce si klade za cíl vysvětlit možnosti komunikace chytrých domácích serverů v rámci sítí IoT na aplikační vrstvě TCP/IP modelu. Vědecká část práce se věnuje problematice implementace chytrého domácího serveru do infrastruktur sítí telekomunikačních operátorů. Přínosy práce lze shrnout následovně:

1. **Systémové začlenění Internetu věcí**
   Systematická část habilitační práce je obsažena zejména v kapitolách 2 a 3, které rozebírají možnosti komunikace v sítích IoT zejména na aplikační vrstvě síťového modelu TCP/IP. V kapitole 2 jsou popsány a zhodnoceny používané modely komunikace v sítích IoT, kdy u každého z nich je uveden praktický příklad jeho použití. Třetí kapitola se zabývá významnými protokoly IoT realizovanými na aplikační vrstvě, konkrétně protokoly MQTT a CoAP. Oba protokoly včetně potřebných prvků síťové infrastruktury jsou v práci velmi podrobně analyzovány. Vedle standardních protokolů se kapitola věnuje také popisu signalizačního protokolu SIP, primárně určeného pro služby internetové telefonie. Je podrobně rozebrán komunikační model protokolu SIP. Navazuje vlastní návrh způsobu komunikace IoT zařízení prostřednictvím protokolu SIP s cílem dosažení interoperability IoT zařízení se sítěmi telekomunikačních operátorů.
   Po prostudování uvedených kapitol by zájemce měl získat přehled o možnostech komunikace zejména na aplikační vrstvě modelu TCP/IP v rámci IoT. Dále by měl získat nové informace o možnostech využití protokolu SIP ke komunikaci v rámci sítí IoT. Uvedená teorie by měla sloužit jako dobrý základ pro další, podrobnější studium konkrétní komunikační technologie či zvoleného aplikačního protokolu.

2. **Vědecký přínos**

   Kapitoly 4 a 5 přináší původní výsledky výzkumu a vývoje v oblasti IoT a experimentální potvrzení navržené komunikace IoT zařízení prostřednictvím protokolu SIP. V práci je na konkrétním příkladu ukázán návrh koncepce a postup vývoje chytrého domácího serveru interoperabilního se sítěmi telekomunikačních operátorů. Vedle doporučení pro výběr vhodného hardware a software jsou uvedeny výsledky experimentálních výkonnostních testů softwarových struktur určených pro vývoj softwaru IoT zařízení, možnosti komunikace chytrého domácího serveru s chytrými domácími prvky a problémy při komunikaci chytrého domácího serveru se síťovými prvky telekomunikačních operátorů prostřednictvím protokolu SIP.

## 1.4 Struktura práce

Habilitační práce je rozdělena do šesti kapitol. Úvodní kapitola popisuje základní koncepci IoT, jeho dělení a cíle práce. Druhá kapitola se věnuje komunikačním modelům navrženým ke komunikaci v sítích IoT. Následuje kapitola popisující protokoly vhodné ke komunikaci v IoT na aplikační vrstvě TCP/IP. Jsou zde popsány a porovnány jak standardně, tak i nestandardně používané protokoly. Pozornost je věnována zejména protokolu SIP. V rámci třetí kapitoly je uveden vlastní koncepční návrh komunikace zařízení IoT se sítěmi telekomunikačních operátorů právě prostřednictvím protokolu SIP včetně návrhu struktury přenášených dat.

Čtvrtá a pátá kapitola se věnuje experimentálnímu ověření výsledků výzkumu z cílem představit ucelené řešení chytrého domácího serveru interoperabilního se sítěmi telekomunikačních operátorů. Čtvrtá kapitola se věnuje komunikaci chytrého domácího serveru se sítěmi IP a IoT, popisu softwarových struktur využitelných k vývoji interní logiky chytrého domácího serveru a požadavkům na hardware. Jednotlivé softwarové struktury jsou vzájemně porovnány a je zhodnocena jejich použitelnost. Pátá kapitola se věnuje implementaci chytrého domácího serveru s využitím protokolu SIP za účelem experimentálního ověření navrženého řešení. Vedle popisu hardwarové i softwarové náročnosti chytrého domácího serveru jsou v páté kapitole uvedena úskalí, ke kterým docházelo při experimentálním nasazení protokolu SIP v chytrém domácím serveru. Závěr habilitační práce shrnuje její systematické i vědecké přínosy.

# 2 MODELY KOMUNIKACE V SÍTÍCH IOT

Vzájemnou komunikaci zařízení M2M v sítích IoT definuje mimo jiné architektura sítě. Doporučení RFC 7452 [17] definuje celkem čtyři modely komunikace:

- komunikace mezi zařízeními – D2D (Device-To-Device),
- komunikace mezi zařízením a datovým centrem – D2C (Device-To-Cloud),
- komunikace mezi zařízením a bránou – D2G (Device-To-Gateway),
- komunikace se sdílením dat,

z nichž jedno chytré zařízení může komunikovat prostřednictvím více modelů v jednom čase [17].

## 2.1 Komunikace mezi dvěma zařízeními

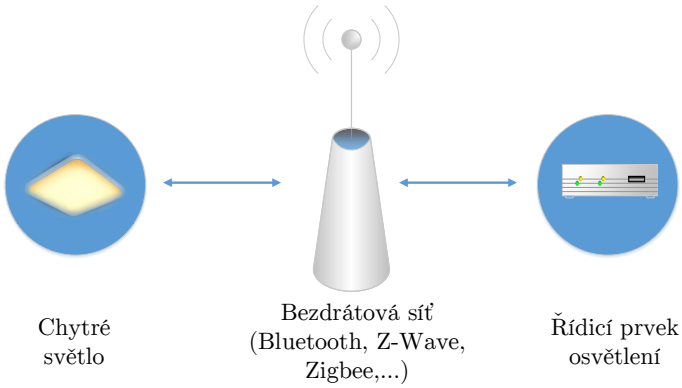
Model komunikace mezi dvěma zařízeními Device-to-Device, označovaný jako komunikace D2D, popisuje vzájemnou komunikaci dvou či více chytrých zařízení od stejných či různých výrobců. Chytrá zařízení jsou mezi sebou přímo propojena, neexistuje žádný mezilehlý uzel (server, brána apod.). Zařízení mohou komunikovat prostřednictvím různých sítí, zpravidla se používá síť Internet. Ke komunikaci jsou ale používány často i síťové technologie Bluetooth [18], BLE (Bluetooth Low Energy) [18], Z-Wave [19] nebo ZigBee [20] (viz obrázek 2.1). Komunikační model D2D využívající zmíněné přenosové technologie se často používá v aplikacích, ve kterých si zařízení mezi sebou vyměňují velmi malé množství dat. K typickým systémům patří domácí automatika (stínící technika, chytré osvětlení, řízení vzduchotechniky prostřednictvím termostatu apod.), ale také nositelná elektronika v podobě náramků fitness, měřičů tepové frekvence atd.

Nevýhodou komunikace D2D je fakt, že každé zařízení má většinou unikátní komunikační model definovaný výrobcem zařízení, což znamená, že každý výrobce vyvíjí proprietární komunikaci místo toho, aby použil standardní řešení. Z pohledu uživatele to znamená, že zařízení od různých výrobců nejsou vzájemně kompatibilní. Není tedy možné kombinovat zařízení od různých výrobců, což je mnohdy limitující.

Aby byla umožněna vzájemná komunikace různých zařízení v rámci modelu D2D, je zapotřebí striktně definovat požadavky, mezi něž patří zejména:

1. Definice fyzické vrstvy - například Bluetooth, Z-Wave, ...
2. Určení podporovaných síťových protokolů - IPv4, IPv6, DLMS/COSEM, HDLC.
3. Stanovení konfiguračního mechanismu přidělování IP adres.
4. Definice podporované síťové architektury - klient-server, peer-to-peer,...

5. Definice potřeb pro mechanizmus vyhledávání chytrých zařízení v síti.
6. Stanovení transportního protokolu.
7. Stanovení aplikačního protokolu.
8. Definice datového modelu.
9. Určení mechanismu pro zabezpečení komunikace a ochranu soukromí uživatele.




Obr. 2.1: Komunikace mezi dvěma zařízeními

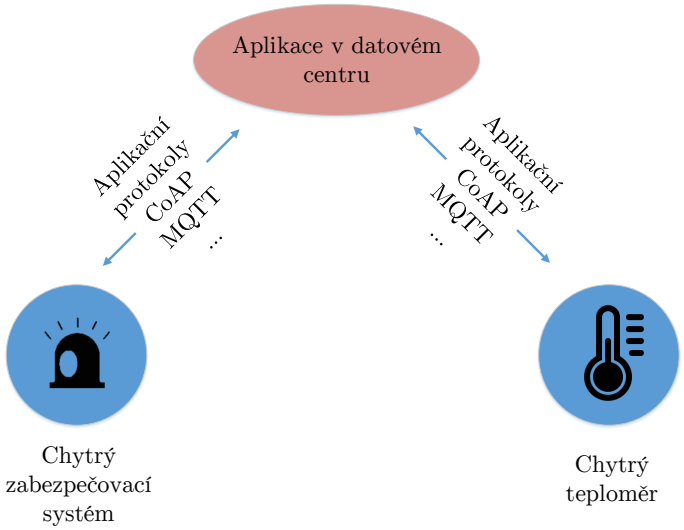
## 2.2 Komunikace mezi zařízením a datovým centrem

U modelu komunikace mezi zařízením a datovým centrem Device-To-Cloud, tzv. komunikace D2C, je každé zařízení připojeno přímo k aplikaci poskytovatele služby (obrázek 2.2). Aplikace si se zařízeními vyměňuje data a řídicí zprávy. Tento typ komunikace často využívá běžné komunikační mechanizmy, mezi něž patří Ethernetové připojení, Wi-Fi připojení či připojení prostřednictvím mobilních sítí 2G/3G/4G. Komunikace D2C je využitelná v případech, kdy chytré zařízení potřebuje získat informace ze vzdáleného serveru, nebo naopak odeslat informace k serveru pro další zpracování. V mnoha případech komunikace D2C je poskytovatel služby zároveň dodavatel chytrých senzorů a zařízení. Stejně jako v komunikaci D2D se často vyskytují chytrá zařízení, která komunikují proprietárními protokoly, čímž znemožňují připojení chytrých zařízení třetích stran.

Jako příklad komunikačního modelu D2C lze uvést chytrý termostat či chytrou televizi připojenou ke vzdálené aplikaci poskytovatele. V případě chytrého termostatu

připojeného pomocí D2C se uživatel může s termostatem spojit například prostřednictvím poskytované služby přes mobilní aplikaci či webový portál, čímž jej může plně kontrolovat na dálku. Chytrá televize připojená k poskytovateli může například odesílat informace o chování uživatele. Poskytovatel získané informace vyhodnocuje a nabízí zpět do televize například obsah šitý na míru danému uživateli.

Jak již bylo zmíněno, komunikace D2C je stejně jako D2D často svázaná s výrobcem konkrétního zařízení. Ten zpravidla provozuje taktéž aplikaci a ke komunikaci používá proprietární protokol, čímž znemožňuje použití služby třetí straně. Provázanost zařízení a služby poskytovatele však uživateli dává určitou jistotu a důvěryhodnost.

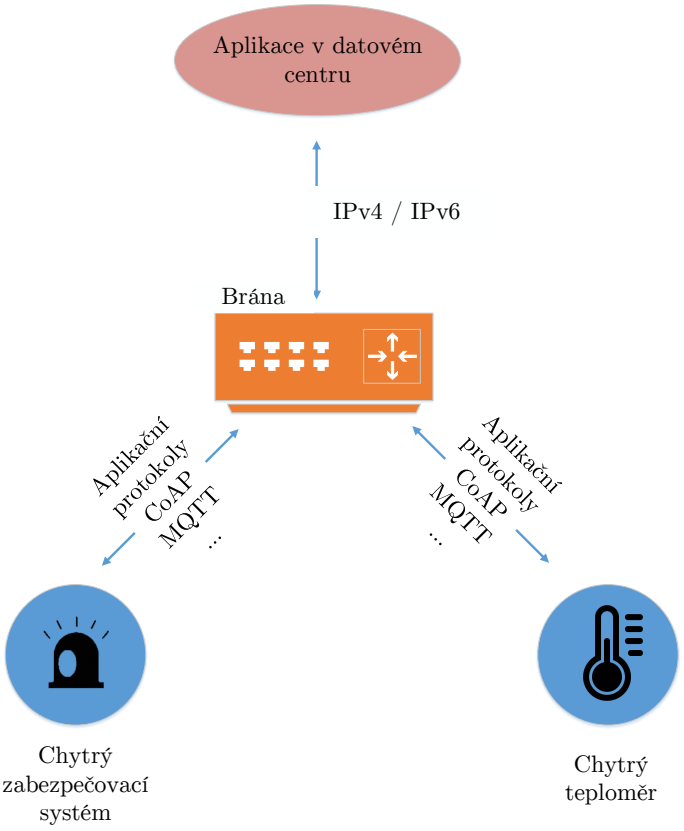



Obr. 2.2: Komunikace mezi zařízením a datovým centrem

## 2.3 Komunikace mezi zařízením a bránou

Komunikace D2C popsaná v předchozích odstavcích funguje spolehlivě v případech, kdy všechna chytrá zařízení a senzory používají běžné komunikační prostředky a široce rozšířené bezdrátové technologie, jako je například Wi-Fi (standard IEEE 802.11). Mnohdy nastávají případy, kdy jsou použity méně rozšířené bezdrátové technologie, které nejsou nijak spřažené s IP komunikací. V těchto případech je vhodné využít specializované chytré brány (Smart Gateway) umožňující přístup do jiných datových sítí. Systém komunikace mezi zařízeními a bránou (Device-To-Gateway) je označován jako komunikace D2G. V komunikačním modelu D2G komunikují všechna

zařízení v rámci jedné oblasti s chytrou bránou, která je propojena sítí Internet s aplikací poskytovatele služby (obrázek 2.3). Chytrá brána je prostředníkem zajišťujícím zabezpečení komunikace, převod datových jednotek, interních protokolů apod. V mnoha případech je chytrá brána mobilní telefon s nainstalovanou aplikací komunikující s aplikací poskytovatele služby (například fitness aplikace apod.). Velmi významnou oblastí komunikace D2G jsou chytré brány pro domácnosti. Ty mohou komunikovat prostřednictvím různých technologií s domácími senzory či jinými zařízeními IoT, sbírat informace, zpracovávat je a posílat dále poskytovateli služby. Nevýhodou modelu D2G je nutnost vývoje specializovaného software pro chytrou bránu.

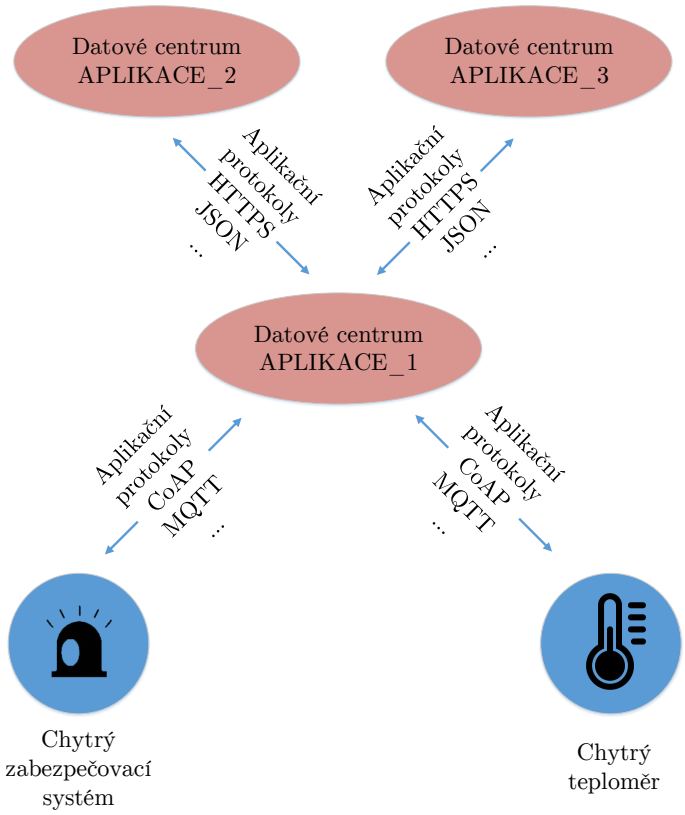



Obr. 2.3: Komunikace mezi zařízením a bránou

## 2.4 Komunikace se sdílením dat

Komunikace se sdílením dat rozšiřuje možnosti komunikace D2C, která je většinou jednoúčelová. Při komunikaci D2C jsou data ze zařízení odesílána pouze jedné apli-

kaci poskytovatele. Uživatelé ovšem mnohdy požadují analýzu dat a jejich porovnání s ostatními, což jim komunikační model D2C neumožňuje. Komunikační model se sdílením dat popisuje obrázek 2.4. Jeho základní myšlenkou je poskytnutí přístupu k informacím třetí straně. Jako příklad můžeme uvést zaměstnance velké korporace, který chce zjistit a analyzovat spotřebu elektrické energie všech budov. Jednotlivé budovy mají vlastní měřicí systémy a každá budova shromažďuje svá data na vlastním úložišti. Model komunikace se sdílením dat umožní získat data ze všech úložišť a tím zjednodušit jejich analýzu, popřípadě jejich další použití.




Obr. 2.4: Komunikace se sdílením dat

# 3 SIGNALIZAČNÍ PROTOKOLY APLIKAČNÍ VRSTVY POUŽÍVANÉ V INTERNETU VĚCÍ

Hlavní myšlenkou IoT v domácím prostředí je vzájemná komunikace všech zařízení. V ideálním případě, jak bylo zmíněno v předchozí kapitole, jsou data ze všech zařízení shromažďována v centrální jednotce, dále bude využíván pojem „chytrý domácí server“. Ten data vyhodnocuje, zpracovává a odesílá do datového centra poskytovatele. Vzájemná komunikace zařízení a chytrého domácího serveru závisí na použité, nejčastěji bezdrátové, síťové technologii (Wireless M-BUS [21], ZigBee, Bluetooth, Z-Wave,...) a je zpravidla řešená proprietárními protokoly výrobců zařízení. Na aplikační vrstvě poté komunikuje chytrý domácí server s aplikací spuštěnou v datovém centru poskytovatele. Mezi známé aplikační protokoly používané ke komunikaci v sítích IoT se řadí protokoly MQTT (Message Queue Telemetry Transport) a CoAP (Constrained Application Protocol). Méně používané protokoly jsou AMQP (Advanced Message Queuing Protocol) a DDS (Data Distribution Service).

Vedle zmíněných aplikačních protokolů přímo určených pro komunikaci v rámci sítí IoT se jeví jako možné použít již zavedené protokoly aplikační vrstvy využívané pro signalizaci, mezi něž patří například protokoly SIP (Session Initiation Protocol) či XMPP (Extensible Messaging and Presence Protocol).

Dále budou podrobně rozebrány protokoly MQTT, CoAP a SIP a bude zhodnocena možnost jejich využití pro chytré domácí servery. Velká pozornost bude věnována protokolu SIP vzhledem k jeho obrovskému rozšíření v sítích telekomunikačních operátorů a jeho využití pro Internet věcí. V případě jeho nasazení by totiž při komunikaci v rámci IoT odpadla telekomunikačním operátorům nutnost zcela nově definovat prvky sítě pro zavedení služeb IoT do domácností.

## 3.1 Protokol MQTT

Protokol MQTT (Message Queue Telemetry Transport) je aplikační protokol určený ke komunikaci zařízení na základě modelu PUBLIKACE/ODBĚR (PUBLISH/SUBSCRIBE), ve kterém jsou informace poskytované publikujícími aplikacemi doručovány infrastrukturou všem odebírajícím aplikacím, které se zaregistrovaly k odběru tohoto typu informací. Mezi velké přednosti protokolu MQTT patří jednoduchost a snadná implementace. V roce 2016 byl standardizován pod ISO/IEC 20922:1016 [22], [23]. Je nezávislý na nižších vrstvách síťového modelu, avšak při jeho nasazení

musí být nižšími vrstvami zajištěny základní dvě kritéria:

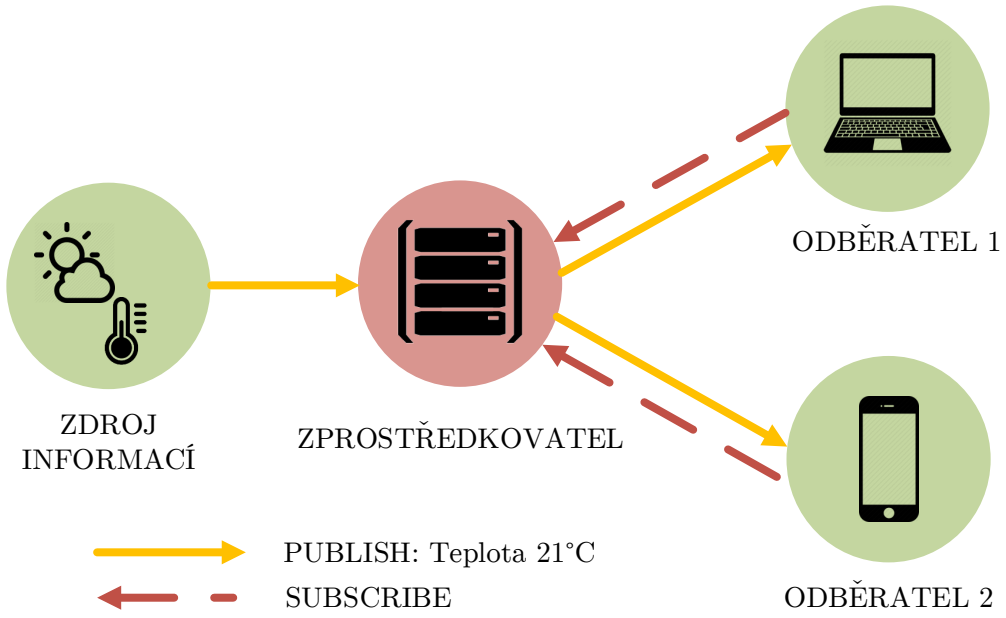
- doručení dat ve stejném pořadí, v jakém byla vyslána,
- bezeztrátový obousměrný přenos.

Obě zmíněná kritéria jsou běžně zajištěna využitím transportního protokolu TCP (Transmission Control Protocol) modelu TCP/IP, který je v současné době ke komunikaci běžně používán. Vlastní komunikace prostřednictvím protokolu MQTT probíhá na základě výměny zpráv, které jsou vždy zařazeny k určitému tématu. Každá zpráva spadá právě pod jedno téma. Témata jsou hierarchická, v hierarchii je dělicí znak /. Hierarchie témat je volitelná, pro každý systém je definována dle potřeb [24].

### 3.1.1 Komponenty sítě

V síti využívající ke komunikaci protokol MQTT jsou definovány tři komponenty viz obrázek 3.1. [24]:

- ZDROJ INFORMACÍ (PUBLISHER) – generuje zprávy PUBLISH,
- ODBĚRATEL (SUBSCRIBER) – generuje zprávy SUBSCRIBE,
- ZPROSTŘEDKOVATEL (BROKER) – zajišťuje rozesílání zpráv.




Obr. 3.1: Architektura protokolu MQTT

#### ZDROJ INFORMACÍ

ZDROJ INFORMACÍ je zařízení, které generuje a vysílá zprávy PUBLISH spadající do určitého tématu. Téma, ke kterému ZDROJ INFORMACÍ přispívá, je již vytvořené ZPROSTŘEDKOVATELEM. V případě že téma, do kterého je poslána zpráva neexistuje, ZPROSTŘEDKOVATEL téma vytvoří.

#### ODBĚRATEL

ODBĚRATEL je zařízení, které žádá zprávou SUBSCRIBE o příjem zpráv určitého tématu a tyto zprávy následně přijímá.

#### ZPROSTŘEDKOVATEL

ZPROSTŘEDKOVATEL je „centrální jednotkou“ sítě, udržuje informace o všech ODBĚRATELÍCH i ZDROJÍCH INFORMACÍ, filtruje příchozí zprávy, přiřazuje je do témat, vytváří témata a distribuuje zprávy na základě požadavků ODBĚRATELŮ. Mimo to se ZPROSTŘEDKOVATEL stará o autorizaci a autentizaci zařízení.

Je dáno, že:

- ZDROJ INFORMACÍ a ODBĚRATEL o sobě navzájem neví,
- ZDROJ INFORMACÍ a ODBĚRATEL nemusí být spuštěni ve stejný čas,
- funkce zařízení není nijak omezena v době publikování či přijímaní zpráv.

### 3.1.2 Komunikace

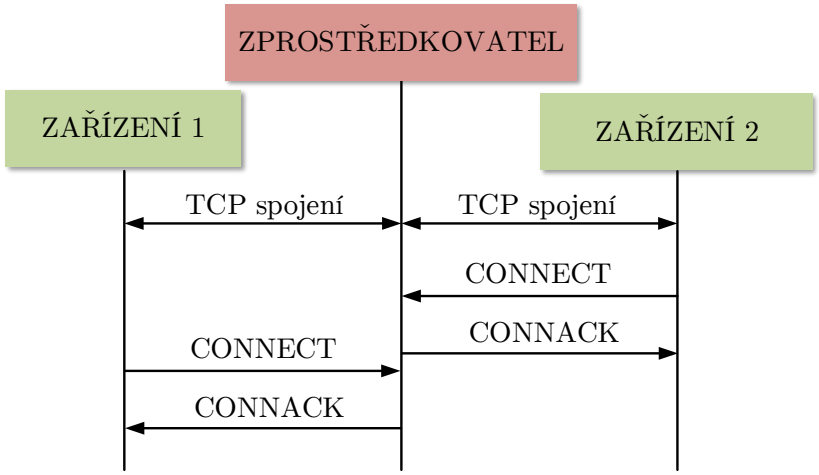
Protokol MQTT je možné využít ke komunikaci jakékoliv zařízení od mikrokontroléru až po plnohodnotný server. Výměna zpráv v síti prostřednictvím protokolu MQTT probíhá vždy mezi zařízením a ZPROSTŘEDKOVATELEM, nikdy mezi dvěma zařízeními navzájem.

#### Zahájení komunikace

Komunikace je vždy zahájena navázáním TCP spojení mezi zařízením (klientem) a ZPROSTŘEDKOVATELEM. Ke spojení se zpravidla používá [23]:

- port 1883 pro nezabezpečenou komunikaci,
- port 8883 pro zabezpečenou komunikaci prostřednictvím TLS (Transport Layer Security) [25],
- porty 8080/8081 pro komunikaci prostřednictvím WebSocketu [26].

Po navázání spojení TCP začne zařízení komunikovat se ZPROSTŘEDKOVATELEM prostřednictvím aplikačního protokolu MQTT. Komunikace je zahájena vysláním zprávy pro připojení zařízení - MQTT CONNECT, na kterou ZPROSTŘEDKOVATEL odpovídá zprávou s kódem indikujícím jeho stav - CONNACK (viz obrázek 3.2). Tím se spojení mezi zařízením a ZPROSTŘEDKOVATELEM považuje za sestavené a je otevřené do té doby, dokud zařízení nepošle příkaz pro odpojení nebo neztratí síťové spojení.

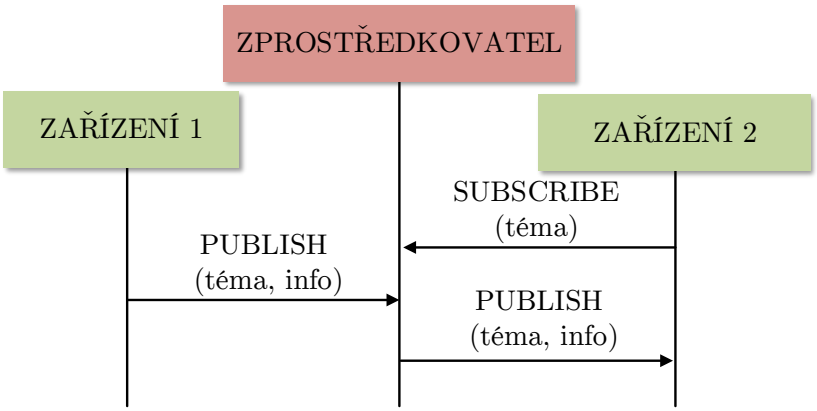



Obr. 3.2: MQTT – Navázání spojení mezi zařízením a ZPROSTŘEDKOVATELEM

### Průběh komunikace

Po úspěšném navázání spojení zařízení se ZPROSTŘEDKOVATELEM je možné komunikovat následovně (viz. obrázek 3.3):

- Zařízení se pomocí jedné či více zpráv SUBSCRIBE poslaných ke ZPROSTŘEDKOVATELI přihlásí k odběru informací o požadovaném tématu. Konkrétní téma je specifikováno ve zprávě SUBSCRIBE. Zpráva SUBSCRIBE může být posílána kdykoliv během spojení. Přijetí zprávy SUBSCRIBE ZPROSTŘEDKOVATEL indikuje odesláním potvrzení SUBACK.
- Zařízení i ZPROSTŘEDKOVATEL mohou vysílat zprávy PUBLISH, kterými přispívají k určitému tématu.
- Zařízení se odesláním zprávy UNSUBSCRIBE ke ZPROSTŘEDKOVATELI odhlásí z odběru informací o tématu popsaného ve zprávě. Odhlášení odběru ZPROSTŘEDKOVATEL potvrdí zprávou UNSUBACK.




Obr. 3.3: MQTT – obousměrná komunikace

#### Ukončení komunikace

Ukončení spojení je indikováno zprávou DISCONNECT vyslanou zařízením ke ZPROSTŘEDKOVATELI. Opětovné připojení zařízení ke ZPROSTŘEDKOVATELI lze provést odesláním zprávy CONNECT a to buď s příznakem `clean seassion` nebo bez něj. V případě, že příznak `clean seassion` není nastaven, zařízení bude po připojení přihlášeno k odběru stejných témat, jako před ukončením spojení.

Při komunikaci prostřednictvím protokolu MQTT je pro zařízení připojená ke ZPROSTŘEDKOVATELI nutné, aby v určitém intervalu hlásila svůj stav. V případě, že zařízení delší dobu neodesílá informace o svém stavu, považuje se za násilně odpojené. V případě, že zařízení nemá v daném intervalu co poslat, posílá zprávu PINGREQ, na kterou ZPROSTŘEDKOVATEL odpovídá zprávou PINGACK.

Vzhledem k tomu, že v případě použití aplikačního protokolu MQTT nemusí být zařízení publikující svůj stav neustále připojené k síti, šetří tím spotřebu elektrické energie. Zabezpečení komunikace je dosaženo autentizací ODBĚRATELŮ a ZDROJŮ INFORMACÍ na straně ZPROSTŘEDKOVATELE [27].

### 3.1.3 Zprávy

#### Zpráva CONNECT

Zpráva CONNECT slouží k připojení zařízení k ZPROSTŘEDKOVATELI a obsahuje zpravidla následující:

- `ClientId` – Jednoznačný identifikátor každého zařízení připojeného ke ZPROSTŘEDKOVATELI.

- `Clean Session` – Příznak indikující požadavek zařízení navázat trvalé spojení. V případě, že je příznak nastaven na hodnotu `false`, ZPROSTŘEDKOVATEL bude ukládat veškerá odebíraná témata a zprávy, které nebyly doručeny. V případě, že je příznak nastaven na hodnotu `true`, ZPROSTŘEDKOVATEL nebude ukládat žádné informace a vymaže vše, co pro dané zařízení bylo doposud uloženo.
- `Username/Password` – Uživatelské jméno a heslo se využívá při autentizaci zařízení. Standardně je heslo posíláno jako prostý text. Doporučuje se heslo šifrovat či posílat jeho hash vytvořený za pomoci hashovací funkce.
- `Will Message` – Zpráva s „poslední vůlí“ je využitelná v případě, že dojde k neočekávanému odpojení zařízení. V takovém případě ZPROSTŘEDKOVATEL rozešle ostatním zařízením informaci obsaženou v „poslední vůli“.
- `KeepAlive` – Časová informace indikující časový interval, ve kterém se zařízení ozve ZPROSTŘEDKOVATELI například zprávou PING. V případě, že se zařízení během stanoveného intervalu neozve, je prohlášeno za nedosažitelné.

### Zpráva CONACK

Zprávu CONNACK posílá ZPROSTŘEDKOVATEL k zařízení jako odpověď na zprávu CONNECT. CONNACK obsahuje pouze dvě informace:

- `Session Present flag` – Příznak indikující zda měl ZPROSTŘEDKOVATEL s připojujícím se zařízením v minulosti navázané trvalé spojení či nikoliv. V případě, že připojující se zařízení má nastaven příznak `Clean Session` na hodnotu `true`, `Session Present flag` je vždy nastaven na hodnotu `true`. V případě, že příznak `Clean Session` je nastaven na hodnotu `false`, tak:
  - pokud jsou pro zařízení s daným `ClientID` u ZPROSTŘEDKOVATELE dostupné nějaké informace, příznak je nastaven na `true`,
  - v ostatních případech je příznak nastaven na `false`.
- `Connect Acknowledge flag` – Kód signalizující úspěšné navázání spojení, v případě neúspěchu vrací kód neúspěchu:
  - 0 – spojení bylo akceptováno,
  - 1 – spojení bylo zamítnuto z důvodu neakceptovaného protokolu,
  - 2 – spojení bylo zamítnuto z důvodu chybného identifikátoru,
  - 3 – spojení bylo zamítnuto z důvodu nedostupnosti serveru,
  - 4 – spojení bylo zamítnuto z důvodu chybného jména nebo hesla,
  - 5 – spojení bylo zamítnuto z důvodu chybné autentizace.

### Zpráva PUBLISH

Zpráva PUBLISH musí vždy nést informace k určitému tématu. ZPROSTŘEDKOVATEL v závislosti na tématu přeposílá zprávu k ostatním zařízením, která mají o dané téma zájem. Data obsažená ve zprávě můžou být poslána binárně, v textové podobě, pomocí XML (Extensible Markup Language) nebo JSON (JavaScript Object Notation) [28]. Důležitý obsah ve zprávě PUBLISH je:

- `Topic Name` – Hierarchicky strukturovaný textový řetězec popisující dané téma. Jednotlivé položky ve struktuře jsou zpravidla odděleny znakem `/`, například `dům/první_patro/kuchyně/lednice/teplota`.
- `QoS` – Kvalita služby pro přenášenou zprávu. Protokol MQTT definuje tři stupně kvality: 0, 1, 2.
- `Retain-Flag` – Příznak stanovující, zda zpráva bude uložena u ZPROSTŘEDKOVATELE pro dané téma jako poslední známá. Nově připojená zařízení dostanou tuto zprávu ihned po požadavku odběru daného tématu.
- `Payload` – Obsah zprávy.
- `Packet Identifier` – Unikátní identifikátor vytvořený mezi zařízením a ZPROSTŘEDKOVATELEM sloužící k identifikaci zprávy. Identifikátor je relevantní pouze v případě nastavení kvality služby na hodnotu 1 nebo 2.
- `DUP flag` – Příznak označující duplicitně poslanou zprávu. Příznak je relevantní pro kvalitu služby s hodnotou 1 nebo 2.

ZPROSTŘEDKOVATEL zpracovává každou příchozí zprávu, v určitých stupních QoS potvrzuje přijetí zprávy zařízení, které ji odeslalo. Následně ZPROSTŘEDKOVATEL zprávu odesílá ostatním zařízením, které mají o téma zájem. ZPROSTŘEDKOVATEL však nemá žádnou zodpovědnost za doručení zprávy všem ODBĚRATELŮM. Zařízení, které zprávu PUBLISH odeslalo, nemá žádné informace o doručení zprávy ODBĚRATELŮM.

### Zpráva SUBSCRIBE

Zprávy PUBLISH s daným tématem jsou vždy vysílány pouze k těm zařízením, které si vyžádala jejich odběr u ZPROSTŘEDKOVATELE. Vyžádání příjmu zpráv určitého tématu probíhá posláním zprávy SUBSCRIBE ke ZPROSTŘEDKOVATELI. Zpráva SUBSCRIBE obsahuje unikátní identifikátor paketu a seznam témat, o které má zařízení zájem:

- *Identifikátor paketu.* Jedinečné číslo identifikující zprávu vyslanou mezi zařízením a ZPROSTŘEDKOVATELEM. Identifikátor je relevantní pouze v případě, že kvalita služeb je vyšší než 0.
- *Seznam témat.* Jedna zpráva SUBSCRIBE může popisovat libovolné množství témat. Každé téma je svázané se stupněm kvality služeb pro dané téma.

#### Zpráva SUBACK

Každou příchozí zprávu SUBSCRIBE potvrdí ZPROSTŘEDKOVATEL zprávou SUBACK poslanou zpět k zařízení. Tato zpráva obsahuje:

- *Identifikátor paketu.* Jedinečné číslo identifikující zprávu vyslanou mezi zařízením a ZPROSTŘEDKOVATELEM. Identifikátor je vždy shodný s identifikátorem ve zprávě SUBSCRIBE.
- *Návratový kód.* ZPROSTŘEDKOVATEL posílá ve zprávě SUBACK ke každému vyžádanému tématu návratový kód. V případě 8 témat je posíláno v rámci jednoho potvrzení 8 kódů:
  - návratový kód 0 – úspěšné zpracování, kvalita služeb nastavena na hodnotu 0,
  - návratový kód 1 – úspěšné zpracování, kvalita služeb nastavena na hodnotu 1,
  - návratový kód 2 – úspěšné zpracování, kvalita služeb nastavena na hodnotu 2,
  - návratový kód 128 – chyba zpracování, zařízení nemá oprávnění k příjmu daného tématu.

#### Zpráva UNSUBSCRIBE

Zpráva je posílána zařízeními k ZPROSTŘEDKOVATELI, jejím úkolem je odhlásit zařízení z odběru zpráv definovaných témat. Její formát je podobný formátu zprávy SUBSCRIBE, obsahuje:

- *Identifikátor paketu.* Jedinečné číslo identifikující zprávu vyslanou mezi zařízením a ZPROSTŘEDKOVATELEM.
- *Seznam témat k odhlášení.* Jedna zpráva UNSUBSCRIBE může odhlásit libovolné množství témat. U odhlášení témat se nepoužívá nastavení kvality služeb, jak tomu bylo u zprávy SUBSCRIBE.

#### Zpráva UNSUBACK

Příchod a zpracování zprávy UNSUBSCRIBE ZPROSTŘEDKOVATEL potvrzuje odesláním zprávy UNSUBACK k zařízení. Zpráva obsahuje pouze identifikátor paketu, který je vždy shodný s identifikátorem ve zprávě UNSUBSCRIBE.

### 3.1.4 Témata

Jak bylo popsáno výše, při odeslání zprávy SUBSCRIBE zařízení definuje, jaké téma jej zajímá a o jakém tématu chce být informováno zprávami PUBLISH. Téma je použito ZPROSTŘEDKOVATELEM k filtrování zpráv. Struktura témat je tvořena hierarchicky. Hierarchie může mít jeden či více stupňů. Při vícestupňové hierarchii je každý stupeň oddělen znakem „/“. Příklady témat:

- `dům/první_patro/kuchyně/lednice/teplota`,
- `dům/první_patro/obývací pokoj/televize/program`,
- `dům/první_patro/kuchyně/voda/tepla/teplota`,
- `dům/první_patro/kuchyně/voda/studena/teplota`,
- `dům/první_patro/koupelna/voda/studena/teplota`,
- `dům/druhé_patro/pokoj/teplota`.

Témata nemusí být vytvořena dopředu. V případě, že zařízení pošle ZPROSTŘEDKOVATELI zprávu s novým tématem ve validním formátu, ZPROSTŘEDKOVATEL téma vytvoří. V případě přihlašování se k tématům zprávou SUBSCRIBE se využívá znaků „+“ a „#“. Znak „+“ se používá v případě požadavku přeskočit jeden stupeň v hierarchii. V případě výše uvedených příkladů a vyžádání témat dle hierarchie:

- `dům/první_patro/koupelna/voda/+/teplota`
  se zařízení upíše pro odběr teplot jak studené, tak i teplé vody.

Znak „#“ se používá v případě požadavku přeskočit více stupňů v hierarchii. V případě výše uvedených příkladů a vyžádání témat dle hierarchie

- `dům/první_patro/#`
  se zařízení upíše pro odběr všech informací z prvního patra. Bude přijímat všechny informace, jejichž téma spadá do hierarchie začínající na:
- `dům/první_patro/`

### 3.1.5 Kvalita služeb v sítích MQTT

Kvalita služeb (Quality of Service – QoS) představuje dohodu mezi vysílačem a příjemcem zprávy. Protokol MQTT definuje tři stupně QoS:

- *Nejvýše jednou* (hodnota 0). Zpráva je doručena bez zajištění kvality (best effort), doručení zprávy není nijak potvrzeno. Může nastat ztráta vysílané zprávy.
- *Alespoň jednou* (hodnota 1). Je garantováno, že zpráva bude doručena alespoň jednou, může však nastat situace vícenásobného doručení stejné zprávy. Vysílač udržuje zprávu v paměti do té doby, dokud nepřijme potvrzení doručení SUBACK. V případě, že potvrzení přijetí SUBACK nepřijde k vysílači v definovaném čase, zpráva je opětovně odeslána.
- *Právě jednou* (hodnota 2). Doručení zprávy je garantováno právě jednou, situace vícenásobného doručení je nepřípustná. Jedná se o nejbezpečnější zajištění kvality služby, doba doručení je však často nejdelší. Garance je zajištěna odesláním dvou zpráv k ZPROSTŘEDKOVATELI a dvou zpráv k zařízení. Po přijetí zprávy PUBLISH s kvalitou služby nastavenou na hodnotu 2 ZPROSTŘEDKOVATEL potvrdí přijetí zprávou PUBREC. ZPROSTŘEDKOVATEL stále drží referenci na identifikátor přijaté zprávy až do té doby, dokud neodešle zprávu PUBCOMP. Jakmile zařízení přijme potvrzení PUBREC, odstraní ze své paměti vyslanou zprávu PUBLISH, protože ví, že ZPROSTŘEDKOVATEL zprávu úspěšně přijal. Zařízení si uloží odpověď PUBREC a k ZPROSTŘEDKOVATELOVI odešle zprávu PUBREL. Jakmile ZPROSTŘEDKOVATEL obdrží zprávu PUBREL, může odstranit veškeré stavy a odpovědi a odeslat zpět k zařízení potvrzení PUBCOMP. Zařízení po přijetí zprávy PUBCOMP může taktéž smazat veškeré stavy a zprávy, které se daného přenosu týkaly. Jakmile během přenosu není některá ze zpráv doručena, je opětovně poslána.

### 3.1.6 Formát protokolu

Formát zprávy MQTT znázorňuje obrázek 3.4 [24]. První dva byty tvoří fixní hlavičku, druhé dva jsou volitelné. Položky hlavičky jsou:

- Typ zprávy (`Message Type`) – definuje typ přenášené zprávy (například Connect, Publish, Subscribe).
- Příznak `DUP` – podává informaci, zda se daná zpráva duplikuje a příjemce ji tedy má čekat podruhé.

- Stupeň kvality služeb (`QoS level`) – umožňuje nastavení tří úrovní kvality doručení zpráv.
- Příznak udržení (`Retain`) – informuje server o tom, že informace poslána zprávou PUBLISH nesmí být vymazána do té doby, dokud nebude doručena všem současným ODBĚRATELŮM.
- Zbývající délka (`Remaining Length`) – definuje počet bytů zprávy zahrnující volitelnou hlavičku a přenášená data.

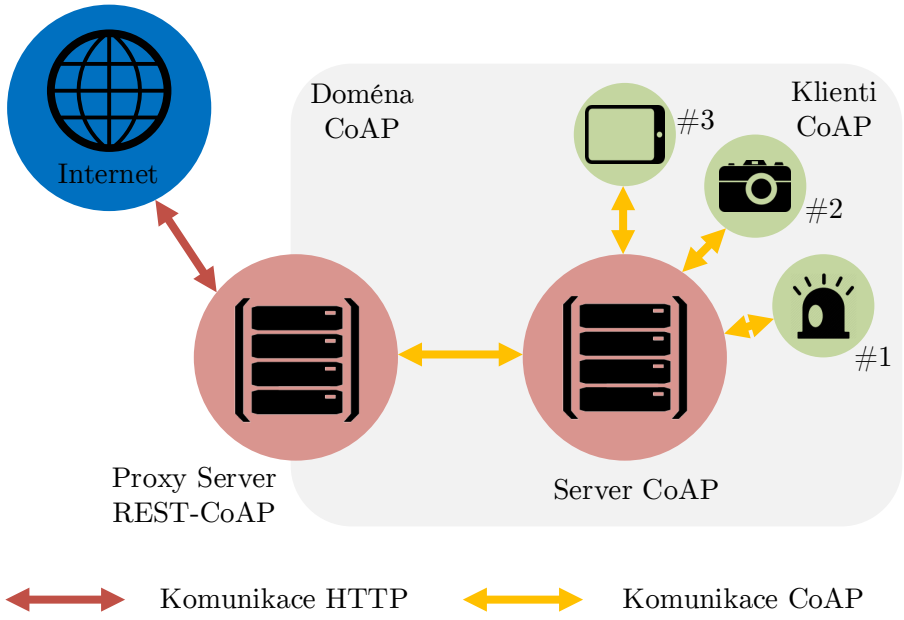
| 0 1 2 3 | 4 | 5 6 | 7 |
|---|---|---|---|
| Typ zprávy | DUP | Stupeň QoS | Udržení |
| Zbývající délka (1 – 4 B) | | | |
| Volitelná délka hlavičky (volitelné) | | | |
| Volitelná data (volitelné) | | | |

Obr. 3.4: Formát zprávy MQTT

## 3.2 Protokol CoAP

Aplikační protokol Constrained Application Protocol - CoAP je určený ke komunikaci přes Internet se zařízeními s nízkým výpočetním výkonem a nízkou spotřebou. Komunikaci prostřednictvím protokolu CoAP lze velmi snadno přeložit do HTTP (Hypertext Transfer Protocol) a následně jednoduše integrovat na webové stránky. Protokol je popsán v doporučení RFC 7252 [29]. Ke komunikaci využívá protokol transportní vrstvy UDP (User Datagram Protocol) síťového modelu TCP/IP. Z důvodu ochrany přenášené zprávy proti chybám, které mohou během přenosu nastat, je v protokolu CoAP integrována kontrola chyb. Protokol CoAP vychází z protokolu REST (REpresentational State Transfer) používaným pro distribuované prostředí, který umožňuje snadným způsobem vytvořit, číst, editovat a smazat informace ze serveru pomocí jednoduchých HTTP příkazů. Komunikaci prostřednictvím protokolu CoAP naznačuje obrázek 3.5 [29].

Komunikační model protokolu CoAP je velmi podobný modelu klient / server protokolu HTTP. U komunikace M2M a IoT jsou však obě komunikující strany v




Obr. 3.5: Architektura protokolu CoAP

roli klienta i serveru. Dále, na rozdíl od HTTP, je ke komunikaci použit nespolehlivý protokol UDP transportní vrstvy. Protokol CoAP definuje dvě podvrsty na aplikační vrstvě [24], [30]:

- vrstva ŽÁDOSTI/ODPOVĚDI (Requests/Responses),
- vrstva ZPRÁVY (Messages).

### 3.2.1 Vrstvy protokolu CoAP

#### Vrstva zprávy

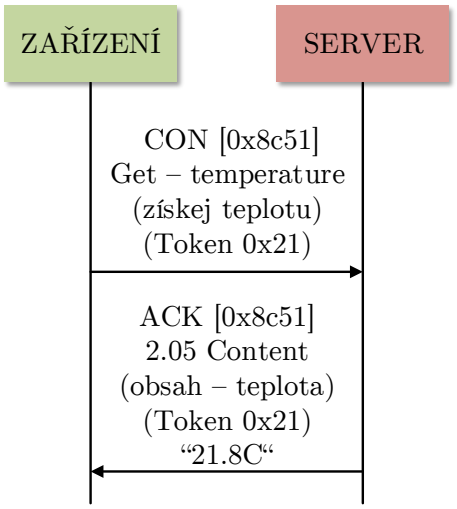
Vrstva zprávy zajišťuje komunikaci s protokolem UDP a zabezpečení zpráv proti chybám. Definuje čtyři typy zpráv:

- CON (Confirmable) – zpráva vyžadující potvrzení přijetí.
- NON (Non-confirmable) – zpráva nevyžadující potvrzení přijetí.
- ACK (Acknowledgement) – zpráva potvrzující přijetí konkrétní jedné zprávy CON. ACK potvrzuje pouze přijetí zprávy, nepotvrzuje její úspěšné zpracování či vyřízení.
- RST (Reset) – zpráva indikující přijetí konkrétní zprávy CON nebo NON, kterou nelze zpracovat z důvodu chybějících informací.

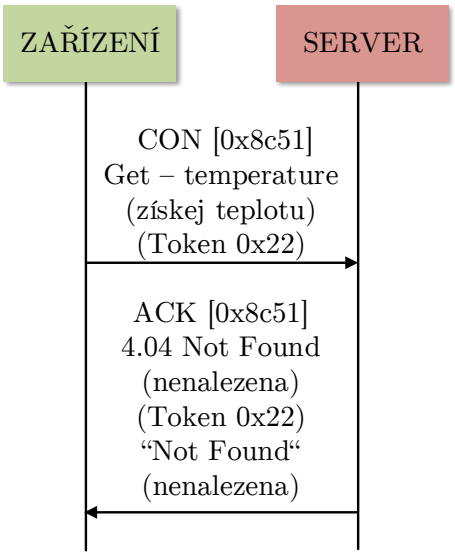
### Vrstva ŽÁDOSTI/ODPOVĚDI

Vrstva ŽÁDOSTI/ODPOVĚDI spravuje komunikaci mezi jednotlivými uzly. Komunikace ve vrstvě ŽÁDOSTI/ODPOVĚDI může být řízena následujícími způsoby:

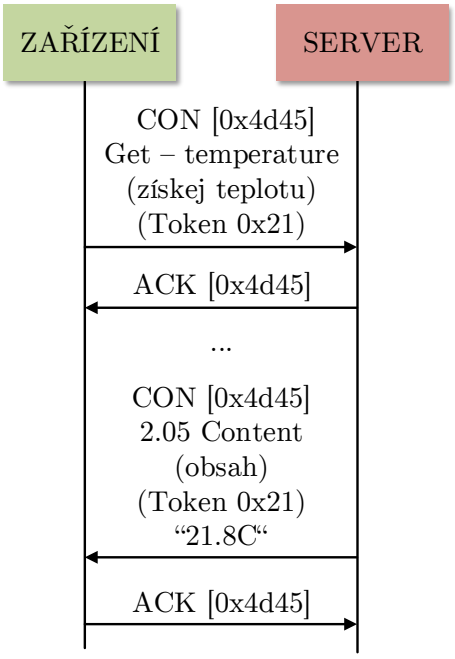
1. PiggyBacked – Zařízení posílá zprávy typu CON nebo NON, v případě spolehlivého přenosu přijímá odpověď připojenou ke zprávě ACK. Odpověď je poslána v rámci ACK nezávisle na tom, zda zpráva CON / NON byla či nebyla úspěšně zpracována. Při úspěšném přenosu zpráva ACK obsahuje oznámení – token (obrázek 3.6) značící úspěšné zpracování, v opačném případě obsahuje chybový kód (obrázek 3.7).
2. Separátní odpověď – V případě, kdy server přijme žádost CON a není schopen ji v daný okamžik zpracovat, odpovídá zařízení prázdnou zprávou ACK. Jakmile je server připraven odpovědět na přijatou žádost, vyšle novou zprávu CON k zařízení, které následně odpoví potvrzením ACK viz obrázek 3.8.
3. Nepotvrzená žádost a odpověď – Zařízení posílá k serveru zprávu NON indikující, že server nemusí odpovídat. Server přeposílá zprávu NON s odpovědí viz obrázek 3.9.



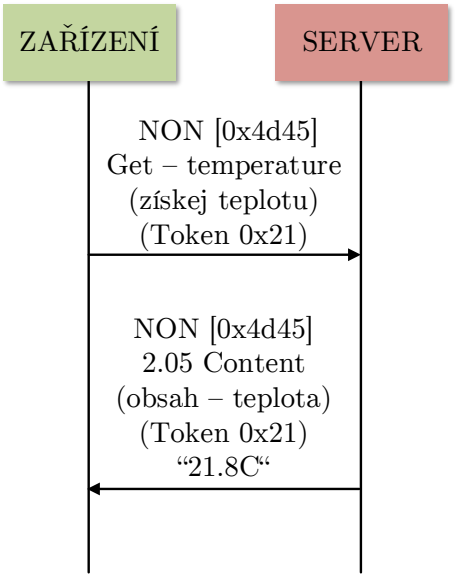

Obr. 3.6: CoAP – odpověď na úspěšné zpracování zprávy serverem




Obr. 3.7: CoAP – odpověď na neúspěšné zpracování zprávy serverem




Obr. 3.8: CoAP – žádost se separátní odpovědí




Obr. 3.9: CoAP – žádost a odpověď se zprávou typu NON

### 3.2.2 Typy komunikace

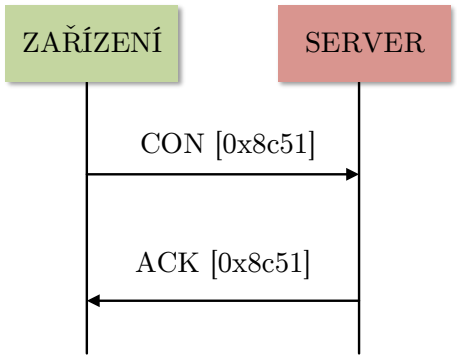
Aplikační protokol CoAP podporuje spolehlivou i nespolehlivou formu komunikace.

#### Spolehlivá komunikace

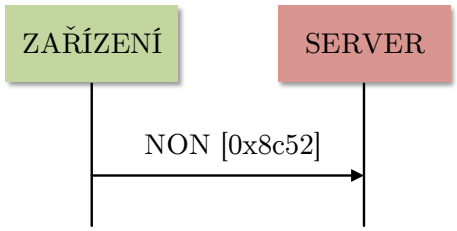
Pro zajištění spolehlivé komunikace se používají zprávy CON, které jsou posílány tak dlouho, dokud vysílač nepřijme zprávu ACK potvrzující příjem dané zprávy. Zpráva ACK obsahuje stejný identifikátor, jako měla zpráva CON, viz obrázek 3.10. Na zprávu ACK se čeká určitý, předem definovaný, časový interval. Pokud potvrzení ACK nedorazí, vysílá se zpráva RST.

#### Nespolehlivá komunikace

Pro zajištění nespolehlivé komunikace se využívají zprávy NON. Tyto zprávy nevyžadují potvrzení ACK viz obrázek 3.11. Zprávy NON však obsahují ID z důvodu možného opětovného odeslání. V případě, že server zprávu chybně zpracuje, odešle odpověď v podobě zprávy RST.




Obr. 3.10: CoAP – spolehlivá komunikace




Obr. 3.11: CoAP – nespolehlivá komunikace

### 3.2.3 Formát protokolu

Formát zprávy protokolu CoAP je znázorněn na obrázku 3.12. Hlavička zprávy má pevnou délku čtyři byty, dále může být připojena volitelná hlavička a tělo zprávy. Její typická délka je mezi deseti a dvaceti byty [31]. Verze protokolu je v současné době jedna. Pole `T` (`TYP`) definuje typ zprávy (CON, NON, ACK, RESET), pole `DT` (`DÉLKA TOKENU`) specifikuje délku volitelného pole `Token`, pole `Kód` indikuje kódy přenášených zpráv a `ID zprávy` obsahuje jedinečný identifikátor přenášené zprávy. Používá se k určení duplicit při přenosu [29].

## 3.3 Protokol SIP

Protokol SIP (Session Initiation Protocol) je signalizační protokol, který je původně určen pro inicializaci, modifikaci a ukončení multimediální relace. Mimo výše zmíněné je možné protokol SIP použít pro posílání krátkých zpráv IM (Instant Messaging), zjišťování stavu (presence) jednotlivých zařízení v síti, publikování různých

| 0 1 | 2 3 | 5 6 7 | 8 15 | 16 31 |
|---|---|---|---|---|
| Ver | T | DT | Kód | ID zprávy |
| Token (volitelné) | | | | |
| Možnosti (volitelné) | | | | |
| Data (volitelné) | | | | |

Obr. 3.12: Formát zprávy CoAP

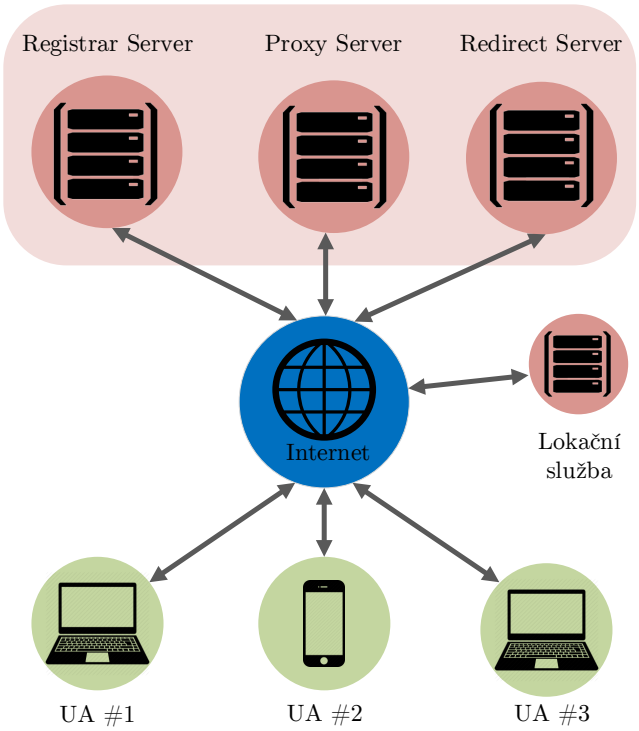
zpráv apod. Aktuální verze protokolu je definována v doporučení RFC 3261 (červen 2002) [32]. K základním funkcím protokolu SIP patří vyhledání uživatele, určení schopností uživatele, zjištění dostupnosti uživatele, nastavení multimediální relace, změna relace. V současné době je protokol SIP často implementován ke komunikaci v sítích telekomunikačních operátorů, kde hraje primární roli zejména při poskytování služby internetové telefonie VoIP (Voice over Internet Protocol). Níže budou popsány základní prvky sítě SIP, možnosti komunikace v rámci sítě a signalizační protokol SIP. V rámci kapitoly bude také uveden vlastní návrh komunikace určené pro IoT zařízení zajišťující možnost výměny informací mezi IoT zařízeními a sítěmi telekomunikačních operátorů. V [32] je zmíněno, že není žádný důvod k rozšiřování protokolu SIP pro prostředí IoT, jeho použití je však možné bez jakýchkoli úprav.

### 3.3.1 Komponenty a komunikace

Komunikace prostřednictvím protokolu SIP je založena na komunikačním modelu modelu ŽÁDOST/ODPOVĚĎ, standardní prvky a síťová architektura SIP jsou naznačeny na obrázku 3.13.

**SIP definuje čtyři základní síťové komponenty:**

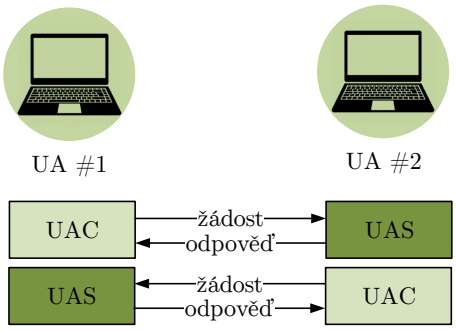
1. uživatelský agent (User Agent),
2. server určený k registraci (Registrar Server),
3. server určený ke směrování (Proxy Server),
4. server určený k přesměrování (Redirect Server).




Obr. 3.13: Architektura sítě SIP

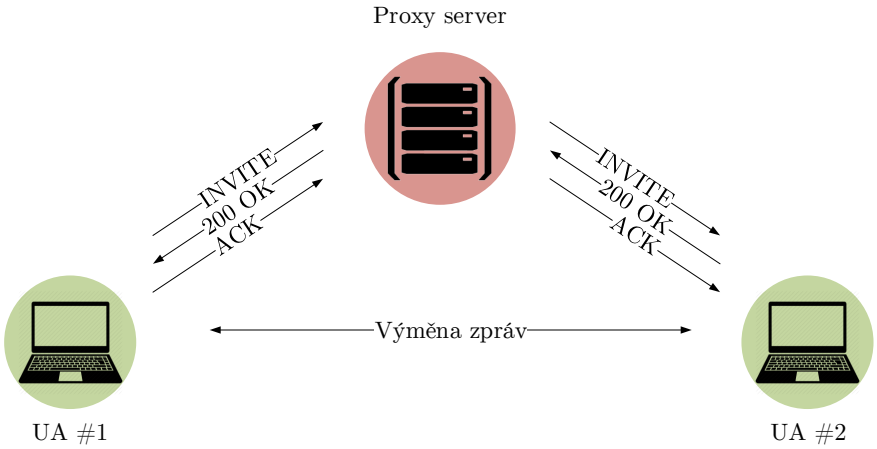
**K jednotlivým síťovým komponentům podrobněji:**

1. *Uživatelský agent.* Uživatelský agent UA je definován jako koncové zařízení, které má dvě funkční jednotky (obr. 3.14):
   - uživatelský agent typu klient UAC (User Agent Client) – generuje a posílá žádosti, přijímá a zpracovává odpovědi,
   - uživatelský agent typu server UAS (User Agent Server) – přijímá a zpracovává žádosti, generuje odpovědi.
2. *Server určený ke směrování.* Server určený ke směrování, Proxy server, se v sítích SIP stará o směrování a přeposílání žádostí na cílový UAS nebo odpovědí na UAC – zajišťuje směrování paketů, volitelně autentizaci uživatelů (obrázek 3.15).
3. *Server určený k přesměrování.* Server určený k přesměrování, Redirect server. Jeho hlavním úkolem je odpovědět na žádost UAC. V odpovědi je uvedena adresa, na kterou má volající svoji žádost přesměrovat tak, aby byla úspěšně doručena k požadovanému UAS. Jinými slovy se jedná o entitu, která zajišťuje přesměrování žádostí (obrázek 3.16).

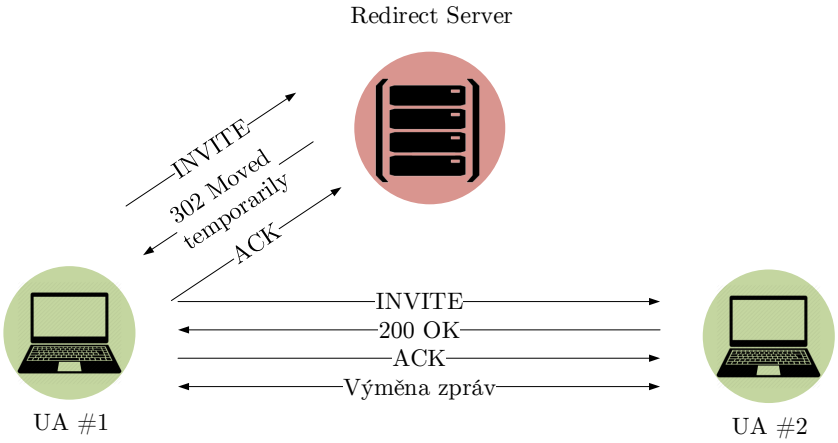



Obr. 3.14: SIP – uživatelský agent

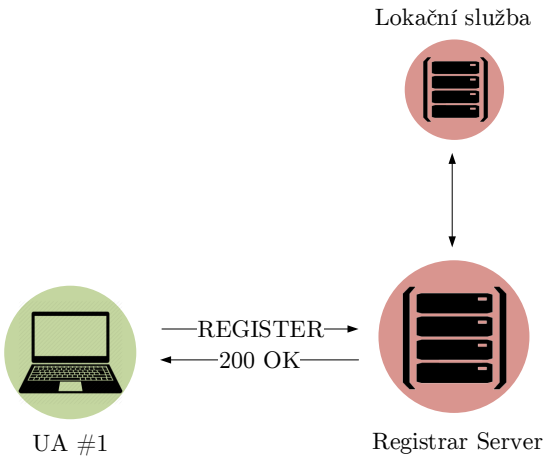
4. *Server určený k registraci.* Server určený k registraci, Registrar server, je speciální typ UAS, který přijímá pouze žádosti o registraci, konkrétně zprávy SIP REGISTER. Registrar server je napojen na lokační službu, která spravuje databázi veškerých registrovaných SIP zařízení jedné domény SIP – udržuje záznamy SIP URI (Uniform Resource Identifier) a IP adresy UA. Lokační služba není součástí doporučení RFC 3261, odkazuje na ni tabulka s veřejnou identitou uživatele AoR (Address of Record). Lokační služba obsahuje informace o konkrétní poloze SIP UA v síti.




Obr. 3.15: SIP – server určený ke směrování (Proxy server)




Obr. 3.16: SIP – server určený k přesměrování (Redirect server)




Obr. 3.17: SIP – server určený k registraci (Registrar server)

### Adresace v sítích SIP

Adresa SIP identifikuje uživatele v síti SIP. Je definována pomocí SIP URI ve tvaru `sip:uzivatel@domena(host):port`. Pole `uživatel` může být například jméno nebo telefonní číslo, `port` je implicitně nastaven na číslo 5060, lze však použít libovolné číslo. Praktické příklady SIP URI jsou následující:

- `sip:petr.cika@vutbr.cz`,
- `sip:541126666@vutbr.cz`.

#### Transakce

Výměna zpráv mezi dvěma UA se v SIP terminologii nazývá transakce. Každá transakce začíná vysláním žádosti od UAC k UAS a končí finální odpovědí přijatou od UAS. Transakce je jednoznačně identifikována v hlavičce zpráv SIP položkami:

- `Call-ID`,
- `via-branch`,
- `tag volajícího`,
- `tag volaného`,
- `CSeq`.

Výsledkem transakce je sestavení, modifikace nebo ukončení dialogu.

#### Dialog

Dialog je v SIP terminologii definován jako vztah mezi dvěma či více UA setrvávající po celou dobu relace. Dialog je jednoznačně identifikován v hlavičce SIP položkami:

- `Call-ID`,
- `tag volajícího`,
- `tag volaného`.

Během jednoho dialogu může být vyřízeno několik transakcí. Každé transakci v rámci jednoho dialogu je inkrementována hodnota CSeq o jedna. [33]

### 3.3.2 Signalizační protokol SIP

Komunikace v sítích SIP až na výjimky pracuje v režimu ŽÁDOST / ODPOVĚĎ. Obecná struktura signalizačního protokolu SIP je naznačena na obrázku 3.18. První řádek hlavičky (startovací řádek) stanovuje, zda se jedná o zprávu typu žádost nebo odpověď. Následující řádky hlavičky definují parametry přenosu, kontakty, různé identifikátory a jiné [32]. Tělo zprávy může nést v podstatě jakékoli informace. Startovací řádek má následující tvary:

- pro žádosti: `METODA_Požadované URI_verze SIP`
- pro odpovědi: `verze SIP_trojciferný kód odpovědi_textová fráze`

| Startovací řádek (ŽÁDOST/ODPOVĚĎ) |
| --- |
| Jeden nebo více řádků hlavičky |
| Prázdný řádek |
| Tělo zprávy (volitelné) |

Obr. 3.18: Struktura protokolu SIP

### Metody definované protokolem SIP

Metody definované protokolem SIP jsou generovány UAC a posílány v rámci žádostí na UAS. V doporučení RFC 3261 [32] je definováno 6 základních metod:

- **INVITE** – slouží k zahájení nebo modifikaci spojení. Žádost INVITE vždy obsahuje veřejnou identitu volaného, volitelně požadované parametry SIP spojení (použité kodeky (audio/video), IP adresy a porty stanovené pro příjem RTP paketů apod.). Pro popis parametrů spojení se většinou používá protokol SDP (Session Description Protocol) [34].
- **ACK** – potvrzuje, že UAC přijal finální odpověď na zprávu INVITE. ACK se používá vždy a pouze s žádostí INVITE. Žádost ACK může nést popis parametrů SIP spojení a to v případě, že tyto informace nebyly přeneseny žádostí INVITE.
- **OPTIONS** – je používána pro dotázání se UAS na jeho schopnosti (podporované metody, kodeky apod.).
- **BYE** – slouží k ukončení dialogu.
- **CANCEL** – slouží ke zrušení nevyřízené žádosti, zrušení transakce. Žádost CANCEL neovlivní předchozí žádosti, které již byly zpracovány a potvrzeny finální odpovědí.
- **REGISTER** – využívá se k registraci UAC k SIP Registrar serveru. UAC generuje žádost REGISTER, která obsahuje následující informace:
  - Veřejnou identitu vyjádřenou jako SIP URI. Ta je obsažena v poli `To` hlavičky.
  - Pozici uživatele v SIP síti vyjádřenou pomocí SIP URI. Ta je obsažena v poli `Contact` hlavičky.

Potvrzení registrace je provedeno odpovědí 200 OK. Po potvrzení je daný uživatel svázán se svou pozicí v Lokační službě. Registrace je pouze dočasná a musí být pravidelně obnovována. Mimo doporučení RFC 3261 existují další metody rozšiřující možnosti komunikace prostřednictvím protokolu SIP:

- INFO (RFC 6086) [35],
- PRACK (RFC 3262) [36],
- SUBSCRIBE (RFC 3265) [37],
- NOTIFY (RFC 3265) [37],
- UPDATE (RFC 3311) [38],
- MESSAGE (RFC 3428) [39],
- REFER (RFC 3515) [40],
- PUBLISH (RFC 3903) [41].

### Odpovědi SIP

Odpovědi SIP indikují stav žádosti, kterou UAC poslal k UAS. Odpovědi SIP jsou číslovány v rozsahu 100 až 699, jsou rozděleny do 6 kategorií 1xx, 2xx, ..., 6xx a klasifikovány jako dočasné nebo finální. Dočasná odpověď indikuje průběh (zpracovávání) žádosti, finální odpověď indikuje vyřízení žádosti.

### Kategorie odpovědí

- **1xx – Dočasná (informativní) odpověď** – indikuje zpracovávání žádosti.
- **2xx – Finální odpověď ÚSPĚCH** – indikuje úspěšné zpracování žádosti.
- **3xx – Finální odpověď PŘESMĚROVÁNÍ** – indikuje úspěšné zpracování žádosti, nejčastěji ji posílá SIP Redirect server. Informuje UAC o adrese, na které se nachází požadovaný UAS.
- **4xx – Finální odpověď CHYBA NA STRANĚ KLIENTA** – indikuje, že žádost nemůže být splněna z důvodu chyby na straně klienta. Důvod nezpracování žádosti je popsán v odeslané odpovědi. To, že se jedná o chybu na straně klienta ve většině případů znamená, že žádost měla špatný formát (např. chybělo některé povinné pole v hlavičce apod.).
- **5xx – Finální odpověď CHYBA NA STRANĚ SERVERU** – indikuje, že žádost nemůže být splněna z důvodu chyby na straně klienta. Odpověď většinou obsahuje pole `Retry-After`, ve kterém je zapsán časový interval udávající dobu, za kterou je možné se opět pokusit o poslání žádosti.

- **6xx – Finální odpověď GLOBÁLNÍ CHYBA** – indikuje případ, kdy žádost nemůže být zpracována žádným UAS v síti. Pokud UAC dostane odpověď 6xx, znamená to, že by neměl opakovat vysílání žádosti do té doby, než uplyne doba obsažená v poli `Retry-After`.

### 3.3.3 Návrh komunikace v IoT s využitím protokolu SIP

Při návrhu komunikace v IoT s využitím protokolu SIP bude vycházeno z předpokladu využití současné síťové infrastruktury telekomunikačních operátorů ke komunikaci s prvky cIoT, zejména pak s chytrými domácími servery. Před samotným návrhem byl analyzován běžný provoz chytré domácnosti za účelem získání představy o objemu přenášených dat v rámci domácích sítí cIoT. Bylo zjištěno, že objem přenášených dat je relativně malý (do 100 kbit/s), ve srovnání s přenosem dat od ostatních prvků domácnosti (počítače, mobilní telefony apod.) je téměř zanedbatelný. V sítích cIoT standardně probíhá výměna krátkých zpráv o stavu různých senzorů, měřidel apod. mezi chytrým domácím serverem a prvky chytré domácnosti. Navrhované řešení bude cílit na komunikaci chytrého domácího serveru s ostatními prvky sítě telekomunikačního operátora nebo s prvky třetích stran připojených do sítě telekomunikačního operátora. I když protokol SIP není primárně určen ke komunikaci v rámci IoT, jeho aplikace není nijak omezena a tudíž by jej mělo být možné využít i pro tuto oblast [24].

Ze základních metod definovaných v doporučení RFC 3261 je využitelná pouze metoda REGISTER určená k registraci UAC (IoT zařízení) k Registrar serveru. Ostatní metody definované v RFC 3261, mezi něž patří INVITE, BYE, CANCEL, ACK a OPTIONS, jsou primárně určeny pro signalizaci audio / video hovoru, což do problematiky IoT nepatří. Z výčtu metod dostupných mimo doporučení RFC 3261 uvedených v předchozím textu lze pro komunikaci uvažovat o využití následujících doporučení:

1. RFC 3265 - Session Initiation Protocol (SIP)-Specific Event Notification popisující komunikaci prostřednictvím nově definovaných metod SUBSCRIBE a NOTIFY,
2. RFC 3903 - Session Initiation Protocol (SIP) Extension for Event State Publication popisující komunikaci prostřednictvím nově definované metody PUBLISH,
3. RFC 3428 - Session Initiation Protocol (SIP) Extension for Instant Messaging popisující komunikaci prostřednictvím nově definované metody MESSAGE.

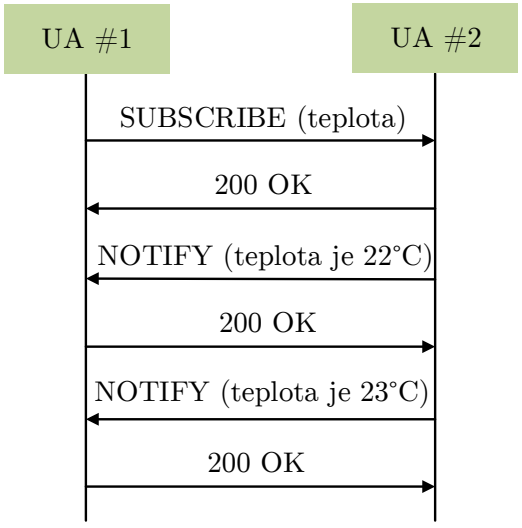
**Jednotlivé metody z výše uvedených doporučení a jejich možné využití jsou dále podrobněji rozebrány.**

### Metoda SUBSCRIBE

Metoda SUBSCRIBE slouží k vytvoření „spojení“ mezi klientskou aplikací poskytující informace o určité službě a službou, která tyto informace poskytuje. U IoT se jedná například o centrální měřicí systém, který uživateli poskytuje informace o teplotě jednotlivých místností v domě. Systém je přihlášen k odběru teplot z jednotlivých čidel, které informují o teplotě, například při každé její změně.

### Metoda NOTIFY

Metoda NOTIFY slouží k oznámení stavu, který je sledován na základě žádosti SUBSCRIBE. V případě změny stavu je následně všem, kteří daný stav sledují, odeslána zpráva NOTIFY. V případě teplotního čidla je aktivátorem například změna teploty (obrázek 3.19).




Obr. 3.19: SIP – SUBSCRIBE, NOTIFY

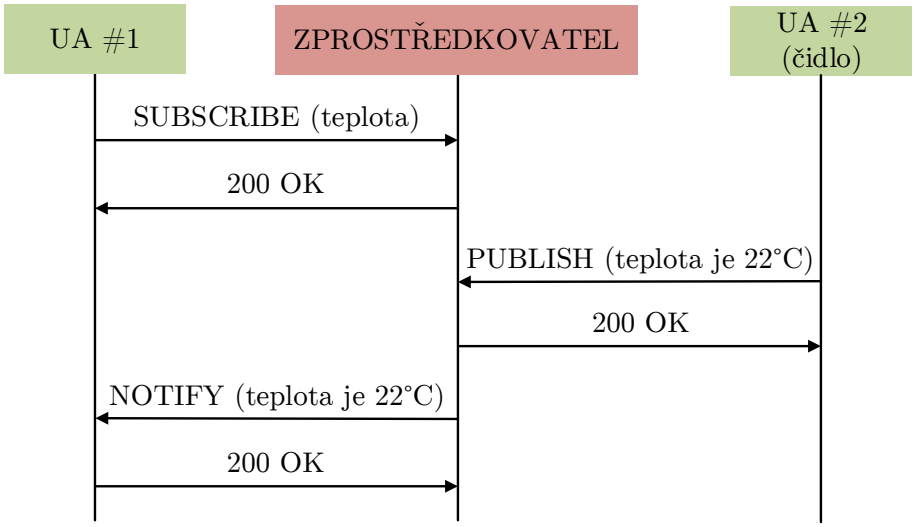
### Metoda PUBLISH

Pomocí metody PUBLISH zařízení zveřejní, obdobně jako u metody NOTIFY, určitý stav či změnu stavu. Metoda PUBLISH byla navržena pro redukci potřebné šířky pásma. Ta je zabezpečena tak, že zpráva PUBLISH je odeslána pouze jednou na

ZPROSTŘEDKOVATELE (BROKER), který následně rozešle tuto informaci všem uživatelům, kteří jsou přihlášeni u ZPROSTŘEDKOVATELE k odběru informací. V případě teplotních čidel z předchozího příkladu je stav následující (viz obrázek 3.20):

1. všechna čidla posílají ZPROSTŘEDKOVATELI pomocí zpráv PUBLISH informaci při změně jejich stavu,
2. ZPROSTŘEDKOVATEL posílá zprávy NOTIFY všem uživatelům, kteří se přihlásili k odběru informací o stavu čidel.

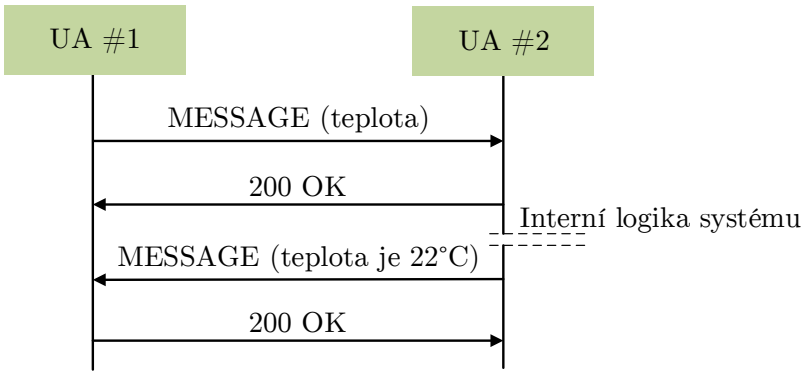
Použití metod SUBCRIBE, NOTIFY a PUBLISH se dá přirovnat k principu komunikace prostřednictvím protokolu MQTT.




Obr. 3.20: SIP – PUBLISH

### Metoda MESSAGE

Metoda MESSAGE byla navržena zejména pro výměnu krátkých textových zpráv mezi uživatelskými agenty – Instant Messaging, IM. Tělo žádosti může nést libovolnou textovou zprávu, která může být odeslána kdykoli uživatel požaduje. Její nespornou výhodou oproti žádostem SUBSCRIBE, NOTIFY a PUBLISH je, že je téměř vždy podporována v rámci sítí SIP. Při oznamování stavu různých zařízení je v podstatě na stejné úrovni se zprávami PUBLISH a NOTIFY. Při její implementaci se musí věnovat pozornost interní logice systému, která není pro IoT specifikována.




Obr. 3.21: SIP – komunikace zprávou MESSAGE

### Registrace uživatelského agenta k registračnímu serveru

V případě řízené komunikace prostřednictvím protokolu SIP se musí každý UA registrovat u registračního serveru (Registrar server). Poté může vysílat žádosti a přijímat odpovědi z dané sítě SIP (provádět transakce). Registrace je taktéž důležitá z toho důvodu, že poloha i IP adresa jednotlivých UA se může měnit společně s jejich identifikací. Při odesílání žádosti je tedy nutné zjistit, kam se má daná žádost odeslat. Informace o aktivitě a pozici klienta udržuje Lokační služba.
Žádost REGISTER musí obsahovat následující pole:

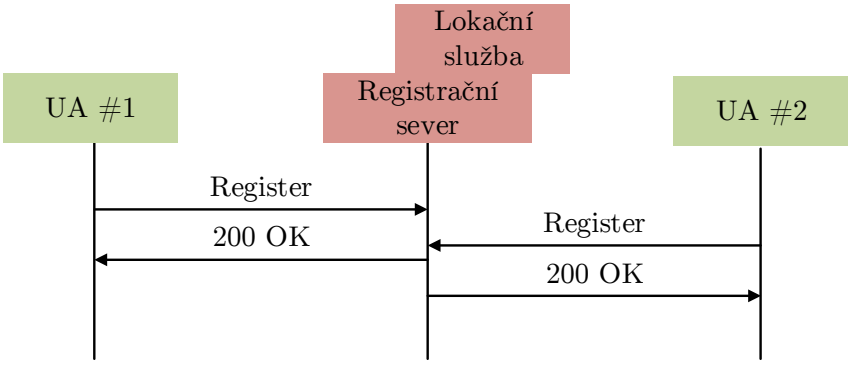
- `Request-URI`: Obsahuje doménu lokační služby, ke které se registrace váže.
- To: Obsahuje SIP URI zápisu v lokační službě, na který se UA registruje.
- `From`: Obsahuje adresu UA provádějícího registraci. Adresa může být shodná s adresou v poli `To`.
- `Call-ID`: Náhodné číslo určené k identifikaci. Číslo je stejné pro všechny registrace daného UA.
- `CSeq`: Sekvenční číslo inkrementující se s každou novou zprávou REGISTER se stejným `Call-ID`. Slouží ke zpracování příchozích zpráv ve správném pořadí.
- `Contact`: Volitelné pole, které může obsahovat kontaktní údaje adres svázaných s daným SIP URI.
- `expires`: Určuje čas v sekundách, po jaký je registrace platná. Běžně se parametr nastavuje na hodnotu 3600. Registrace musí být v pravidelných intervalech obnovována.

Registrace UA k registračnímu serveru může probíhat principiálně dvěma způsoby:

- bez autentizace,
- s autentizací.

### Registrace uživatelského agenta k registračnímu serveru bez autentizace

Registrace uživatelského agenta k registračnímu serveru bez autentizace je popsána na obrázku 3.22. Registrující se UA potřebuje ke své registraci pouze SIP URI, heslo



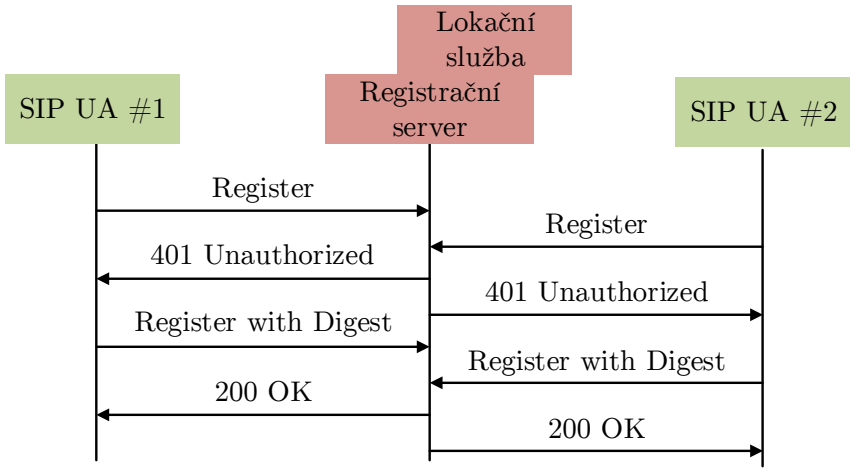

Obr. 3.22: SIP – registrace uživatelského agenta k registračnímu serveru bez autentizace

není vyžadováno. V případě úspěšné registrace je vrácena odpověď 200 OK

### Registrace uživatelského agenta k registračnímu serveru s autentizací

Mnohem častěji je využívána registrace uživatelského agenta k registračnímu serveru s autentizací, kterou lze z důvodu zabezpečení autentičnosti zařízení v rámci IoT použít. Výměna zpráv při registraci s autentizací je naznačena na obrázku 3.23. Pro ověření registrujícího se UA je doporučené využít autentizační proces popsaný v RFC2617 [42]. Do hlavičky registrační zprávy se přidává speciální pole `WWW-Authenticate`, do odpovědi pak pole `Authorization`. Registraci uživatelského agenta k registračnímu serveru s autentizací lze rozdělit do následujících kroků:

1. UA odesílá na registrační server standardní žádost REGISTER bez pole `Authorization`.
2. Registrační server reaguje na příchozí žádost odpovědí 401 Unauthorized, do které přidává pole `WWW-Authenticate`. V poli `WWW-Authenticate` jsou obsažené položky, které jsou nutné pro správný výpočet odpovědi potřebné pro au-




Obr. 3.23: SIP – registrace uživatelského agenta k registračnímu serveru s autentizací

tentizaci UA. Jedná se o položky `realm, qop, nonce, opaque, algorithm`. `Realm` nese název serveru, ke kterému se UA autentizuje, `qop` (Quality of Protection) definuje kvalitu zabezpečení a nabývá hodnot `auth` nebo `auth-int`, `nonce` (Number of once) a `opaque` jsou náhodně vygenerované řetězce, `algorithm` definuje algoritmus pro výpočet odpovědi (standardně MD5) [42]. Příklad pole `WWW-Authenticate` je uveden níže:

```
WWW-Authenticate: Digest realm="test.cz", qop="auth",
nonce="OTc3ODk4MDI5O4QT9tFRfY2wUh0gO5d+HqU=",
opaque="ab3686061475a405d901b4aafee51b68", algorithm="MD5"
```

3. Po zpracování odpovědi 401 Unauthorized UA vygeneruje novou žádost s autentizačními údaji vloženými do pole `Authorization`. Pole `Authorization` obsahuje položky `username`, `realm`, `opaque`, `nonce`, `digest-uri`, `response`. Položky `realm`, `nonce` a `opaque` obsahují stejné informace, jako byly v poli `WWW-Authenticate`, `username` obsahuje uživatelské jméno, `uri` je duplikované URI žádosti z hlavičky protokolu SIP a `response` obsahuje vypočtenou odpověď, díky níž je ověřena autenticita UA. Příklad pole `Authorization` je uveden níže:

```
Authorization: Digest response="1f185cec85e3b325c453af28467e52b3",
username="test",nonce="OTc3ODk4MDI5O4QT9tFRfY2wUh0gO5d+HqU=",
realm="test.cz",opaque="ab3686061475a405d901b4aafee51b68",
uri="sip:test.cz"
```

4. Potvrzení úspěšné autentizace je provedeno odeslání odpovědi 200 OK registračním serverem k UA. V případě negace je proces autentizace opakován.

**Výpočet odpovědi `response`** Odpověď se počítá v šestnáctkové soustavě dle rovnice:

$$response = KD(H(A1), nonce + " : " + H(A2)), \tag{3.1}$$

kde

- `KD` je funkce pro zřetězení v kombinaci s hashovací funkcí:
  $KD(x, y) = H(x + " : " + y)$,
- `H` je hashovací funkce MD5,
- `A1` je řetězec spojený z:
  $A1 = username + " : " + realm + " : " + password$,
- `A2` je řetěze spojený z
  $A2 = method + " : " + digest - uri + "$.

Registrace se musí pravidelně opakovat. Nejzazší termín pro opakování je 3600 s.

Po úspěšné registraci UA může být zahájen přenos zpráv. Pokud dojde k odpojení, UA je potřeba odregistrovat. V případě, že dojde ke korektnímu ukončení, UA odesílá zprávu REGISTER s hodnotou 0 v poli `expires`, v případě násilného odpojení UA (výpadek proudu apod.) se zařízení odregistruje po uplynutí doby stanovené v poli `expires` poslední odeslané zprávě REGISTER.

### Komunikace

Přenos zpráv v rámci sítě IoT mezi chytrými domácími servery či jinými entitami síťové infrastruktury telekomunikačních operátorů by mohl být řešen výměnou zpráv popsanou na obrázku 3.21. Zpráva SIP MESSAGE je primárně určená k posílání krátkých textových zpráv. K jejich využití v rámci IoT je však nutné navrhnout strukturu přenášených dat, která z důvodu interoperability musí být u všech komunikujících stran dodržena.

### Formát přenášených dat ve zprávě SIP MESSAGE

Formát zprávy posílané mezi zařízením IoT a serverem není v současných standardech nikde definován. Terminologie pro vyjádření různých naměřených hodnot z chytrých zařízení není známa a je definována pro každý systém unikátně. Tento fakt představuje problém v interoperabilitě mezi systémy, neboť při propojení nejsou systémy od různých výrobců vzájemně kompatibilní ani v případě využití stejné přenosové technologie a stejného aplikačního protokolu. Jako vhodné se ke komunikaci prostřednictvím protokolu SIP a zpráv SIP MESSAGE jeví využití datové struktury dle JSON [28]. JSON je nezávislý na programovacím jazyku a je hojně využíván právě k výměně dat mezi různými entitami. Jeho struktura je definována v doporučení IETF RFC 7159 [43]. Vedle protokolu JSON v současné době existuje protokol Protocol Buffers, který vyvíjí společností Google Inc. Struktura protokolu je velmi podobná struktuře JSON. Nejedná se však o standardizované řešení a jeho další vývoj je přímo závislý na soukromém sektoru, společnosti Google Inc. Z uvedeného důvodu není toto řešení vhodné použít, neboť při ukončení vývoje společností Google Inc. by systémy s daným protokolem mohly mít potíže při jejich dalším rozvoji.

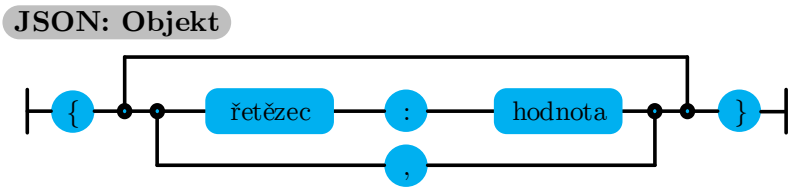
Strukturu JSON lze využít vedle protokolu SIP také s ostatními aplikačními protokoly v IoT, mezi něž patří CoAP, MQTT a další. V současné době se u formátu JSON běžně používají dvě datové struktury [43]:

1. Kolekce párů název-hodnota – lze implementovat jako objekt.
2. Seřazený seznam hodnot – lze implementovat jako pole.

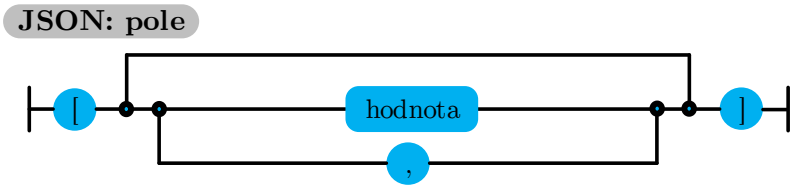
Obě struktury lze realizovat podle následujícího systému:

- *Objekt.* Objekt je neuspořádaná množina párů (řetězec-hodnota) (obrázek 3.24). Každý řetězec je od hodnoty oddělen znakem `:`. Jednotlivé páry jsou odděleny znakem `,`. Objekt je uvozen závorkami `{}`
- *Pole.* Pole představuje seřazenou kolekci dat (obrázek 3.25). Pole je vždy uvozeno závorkami `[ ]`. Hodnoty v poli jsou odděleny čárkou `,`.
- *Hodnota.* Hodnota je reprezentována řetězcem, číslem, objektem, hodnotou logickou hodnotou pravda (boolean true), logickou hodnotou nepravda (boolean false), prázdnou hodnotou označovanou jako null, uvozeným uvozovkami `“ “`. (obrázek 3.26)

- *Řetězec.* Řetězec lze přirovnat k řetězcům jazyků C nebo Java. Řetězec představuje nula nebo více znaků ve znakové sadě Unicode využívajících únikových sekvencí. Sekvence znaků Unicode je uvozená uvozovkami " ". Hodnoty jsou odděleny zpětným lomítkem \. (obrázek 3.27)
- *Číslo.* Číslo je podobné číslům v jazycích C a Java s tím, že není používán ani oktalový ani hexadecimální zápis. (obrázek 3.28)



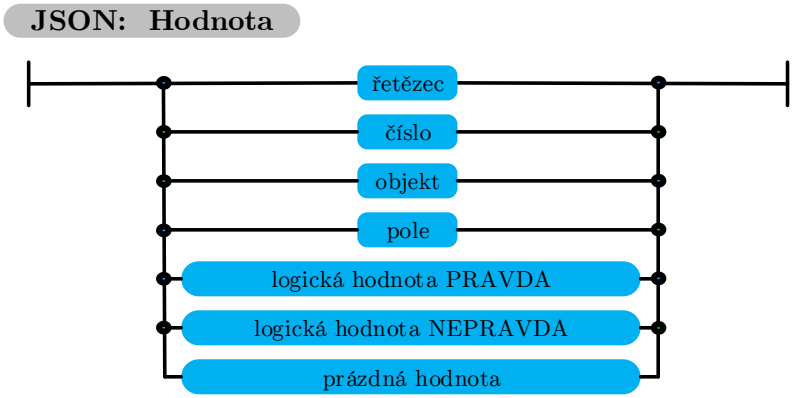

Obr. 3.24: Objekt ve struktuře JSON



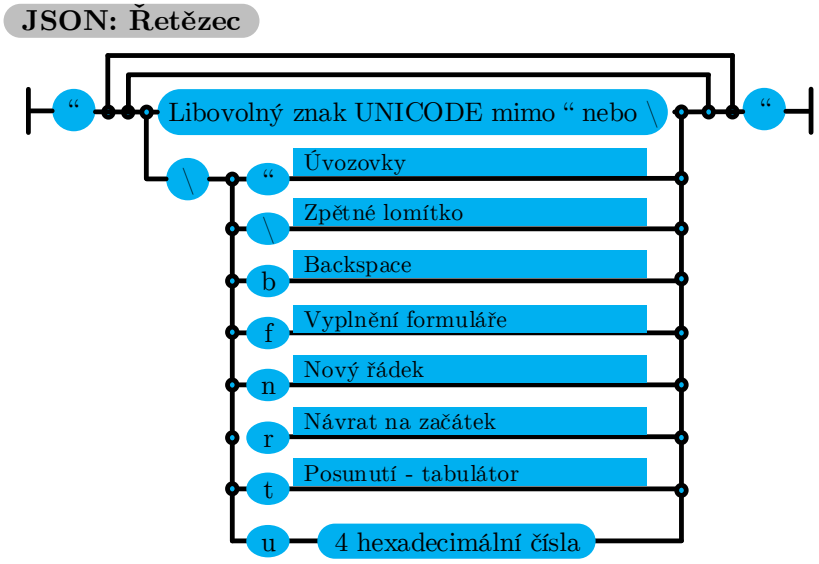

Obr. 3.25: Pole ve struktuře JSON

### Navržená struktura pro přenos dat mezi chytrými domácími servery a sítěmi telekomunikačních operátorů

Pro komunikaci chytrého domácího serveru s vnější sítí prostřednictvím protokolu SIP byla navržena struktura JSON tak, aby bylo možné jednoznačně identifikovat posílané informace. Při návrhu bylo vycházeno ze současných chytrých měřicích prvků v domácnostech, které posílají informace o svém stavu v systémových kódech. Nejčastěji používaný standard definující systémový kód je OBIS (Object Identification System) [44].




Obr. 3.26: Hodnota ve struktuře JSON



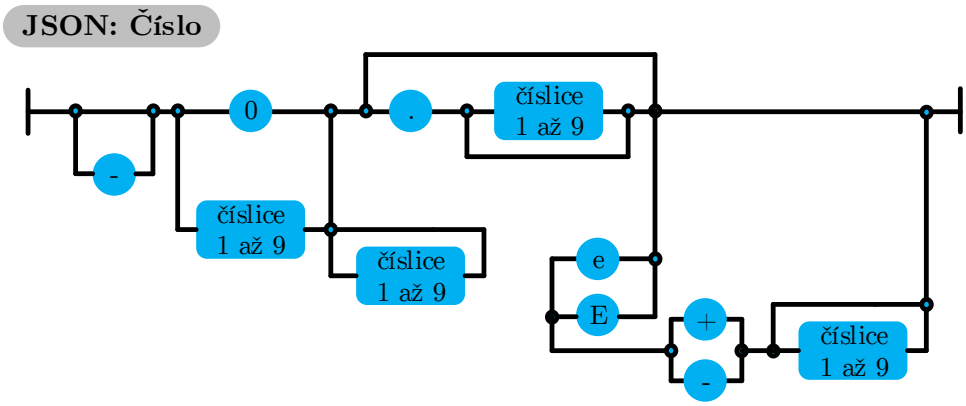

Obr. 3.27: Řetězec ve struktuře JSON

Kód OBIS definuje celkem šest skupin označených písmeny A až F, z nichž nemusí být všechny v sekvenci použity:

- Skupina A definuje médium (0 = libovolný objekt, 1 = elektřina, 6 = teplo, 7 = plyn, 8 = voda atd.)
- Skupina B definuje kanál. Zařízení s více kanály musí definovat přesný kanál, kterým jsou data vysílána.

- Skupina C definuje fyzickou veličinu (teplota, proud, napětí apod.).
- Skupina D definuje výsledek výpočtu specifického algoritmu.
- Skupina E určuje typ měření definovaném skupinami A-D - například přepínání rozsahů.
- Skupina F odděluje výsledky z části definované skupinami A až F. Typicky se jedná o specifikace časových pásem.

Jednotlivé skupiny jsou odděleny dle následujících separátorů: A-B:C.D.E*F.



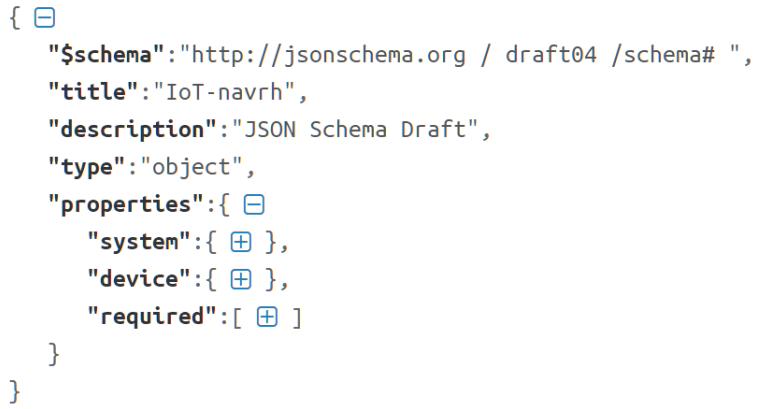

Obr. 3.28: Číslo ve struktuře JSON

V rámci návrhu komunikace mezi domácími chytrými servery byla na základě standardu OBIS definována struktura JSON vhodná pro použití v IoT v chytrých domácích serverech tak, aby byla dosažena vzájemná interoperabilita komunikujících zařízení. Navržená struktura na obrázku 3.29 obsahuje objekt `properties`, který zahrnuje objekty `system`, `device` a `required`. Objekt `required` definuje, jaké ostatní objekty jsou vyžadované při odesílání informací. V současném návrhu je povinný pouze objekt `device`.

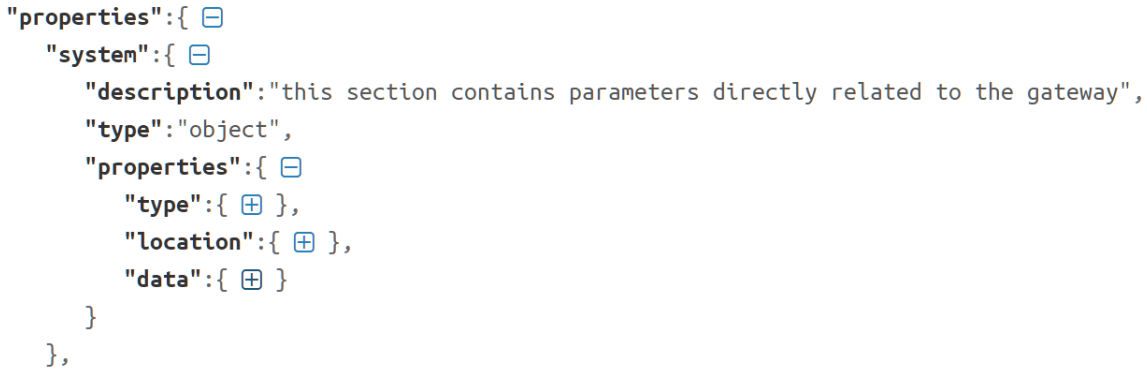
Objekt `system` slouží k popisu parametrů konkrétního chytrého domácího serveru. Obsahuje jeho vlastnosti, informace o jeho geografickém umístění a přenášená data.

Objekt `device` obsahuje parametry konkrétního chytrého senzoru či zařízení včetně jeho identifikace. Udržuje informace o typu zařízení, verzi kódu OBIS, jednoznačném identifikátoru zařízení, čase generování dat a samotná data.

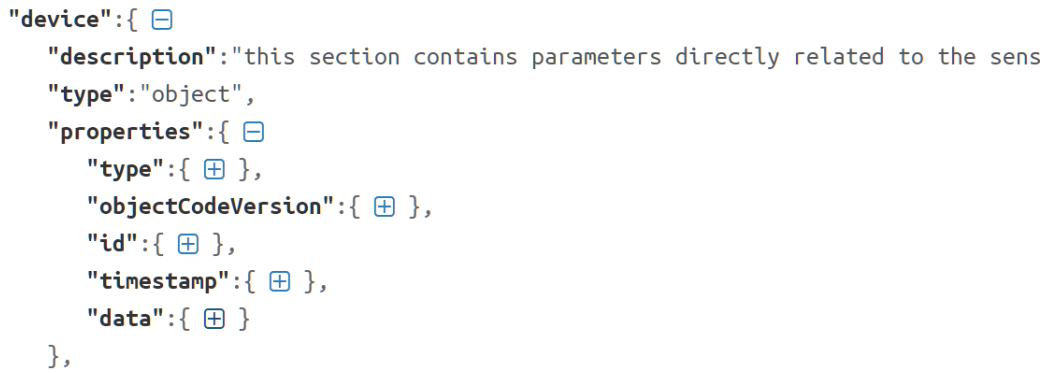
Struktura dodržuje předepsaná pravidla formátu JSON, je snadno rozšiřitelná o další potřeby a byla prezentována v impaktovaném článku [24].




Obr. 3.29: Návrh struktury JSON k přenosu dat mezi chytrými domácími servery a sítěmi telekomunikačních operátorů




Obr. 3.30: Struktura JSON objektu s názvem `system`




Obr. 3.31: Struktura JSON objektu s názvem `device`

## 3.4 Ostatní protokoly

Mezi ostatní aplikační protokoly používané ke komunikaci v IoT patří například Advanced Message Queuing Protocol – AMQP, Data Distribution Service – DDS a Extensible Messaging and Presence Protocol – XMPP. Vzhledem k charakteristice práce není podstatné tyto protokoly dále rozebírat, neboť jejich zastoupení v praxi je menší než u dříve popsaných a ani jeden z nich se nehodí pro využití v sítích telekomunikačních operátorů. Jejich rozbor byl publikován v impaktovaném článku [24].

## 3.5 Porovnání protokolů MQTT, CoAP, SIP a ostatních protokolů

Aplikační protokoly lze porovnat na základě různých kritérií. Pro využití aplikačního protokolu ke komunikaci chytrého domácího serveru se sítěmi telekomunikačních operátorů v rámci IoT je vhodné se zaměřit na následující parametry [24]:

- standardní podpora protokolu v sítích telekomunikačních operátorů,
- podpora architektury REST,
- podpora přenosových protokolů na transportní vrstvě,
- možnost použít metody přenosu typu Publish/Subscribe,
- možnost použít přenos na principu Request/Response,
- podpora zabezpečení přenosu,
- možnost zabezpečit definovanou kvalitu služby (Quality of Service - (QoS),
- velikost hlavičky aplikačního protokolu.

Porovnání aplikačních protokolů na základě výše zmíněných parametrů je přehledně znázorněno v tabulce 3.1, kde sloupec TELCO popisuje nativní podporu protokolu v sítích telekomunikačních operátorů. Jako velmi universální protokoly se dle tabulky 3.1 jeví protokoly CoAP a SIP. Rozdíl je pouze v podpoře protokolu v sítích telekomunikačních operátorů, RESTful a ve velikosti hlavičky. Protokol CoAP má přesně definovanou velikost hlavičky, která je pouze 4B. U protokolu SIP je velikost hlavičky variabilní a nelze přesně stanovit. Nicméně se předpokládá, že bude vždy větší než jsou 4B u protokolu CoAP.

Tab. 3.1: Porovnání aplikačních protokolů

| Aplikační protokol | TELCO | RESTful | Transportní protokol | Publish / Subscribe | Žádost / Odpověď | Zabezpečení | QoS | Velikost hlavičky (B) |
|---|---|---|---|---|---|---|---|---|
| CoAP | N | A | UDP, SMS | A | A | DTLS | A | 4 |
| MQTT | N | N | TCP | A | N | SSL | A | 2 |
| MQTT-SN | N | N | TCP | A | N | SSL | A | 2 |
| XMPP | N | N | TCP | A | A | SSL | N | variabilní |
| AMQP | N | N | TCP | A | N | SSL | A | 8 |
| SIP | A | N | TCP, UDP, SMS | A | A | SSL, TLS | A | variabilní |
| DDS | N | N | TCP, UDP | A | N | SSL, DTLS | N | variabilní |
| HTTP | N | A | TCP | N | A | SSL | N | variabilní |

Při návrhu systémů IoT je nutné stanovit způsob komunikace s poskytovatelem služby. V případě, že je navrhován systém určený pro domácnosti, který má být pod správou telekomunikačních operátorů, jeví se jako vhodný kandidát protokol SIP, který je v současnosti většinou sítí telekomunikačních operátorů podporován a využíván. Zároveň je pevnou součástí mobilních sítí čtvrté generace, 4G jako část IMS (IP Multimedia Subsystem).

# 4 KONCEPCE CHYTRÉHO DOMÁCÍHO SERVERU

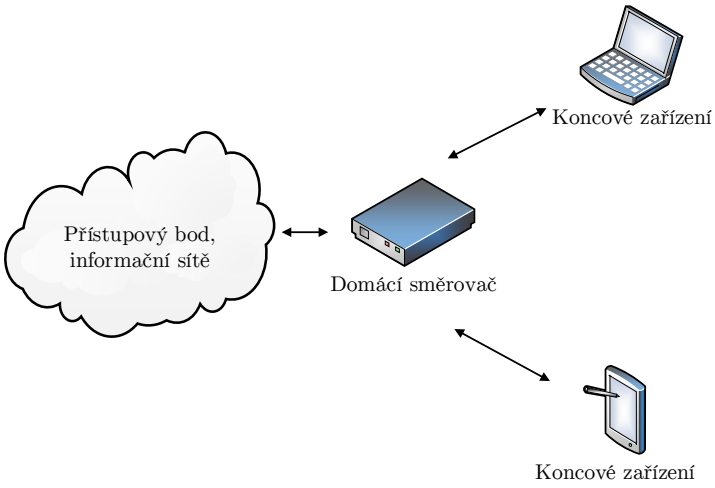
Cílem čtvrté a páté kapitoly je představení problematiky vývoje chytrého domácího serveru s implementovaným návrhem komunikace prostřednictvím protokolu SIP a experimentální ověření správnosti návrhu. Dle HGI (Home Gateway Initiative) [45], AllSeen Alliance [12] i oneM2M [11] je chytrý domácí server považován za klíčový prvek chytré domácnosti. Předpokládá se, že koncový uživatel může k chytrému domácímu serveru přistupovat přímo z domácí sítě nebo prostřednictvím poskytovatele služeb. Některé výstupy popsané ve čtvrté a páté kapitole jsou výsledky výzkumu a vývoje v oblasti IoT na Vysokém učení technickém v Brně realizovaného pro Telekom Austria Group (Rakousko) v letech 2013 – 2016, jehož se autor habilitační práce ve velké míře účastnil. Cílem výzkumu byl, mimo jiné, návrh komunikace chytré domácnosti se sítěmi telekomunikačních operátorů bez nutnosti zásahu do současné síťové infrastruktury a její experimentální ověření.

Před vlastním návrhem komunikace popsané v předchozí kapitole byly stanoveny požadavky na funkcionalitu chytrého domácího serveru a na jeho schopnosti komunikovat s okolím. Hlavní požadavky byly zvoleny následovně:

- Server bude možné integrovat do sítí telekomunikačních operátorů bez potřeby rozšiřování síťové infrastruktury.
- Server bude snadno rozšiřitelný a modifikovatelný pro potřeby konkrétního uživatele.
- Server bude umožňovat komunikaci prostřednictvím komunikačních standardů IoT využívaných v domácnostech.
- Server bude schopný v případě potřeb integrovat nové standardy.
- Hardware serveru bude založen na běžně dostupných HW komponentech.
- Software serveru bude vyvinut prostřednictvím otevřené, volně dostupné softwarové struktury, tzv. frameworku.
- Server bude mít za úkol sběr, agregaci, zpracování a vyhodnocení přijatých dat z chytrých zařízení.
- Server bude schopen distribuovat zpracovaná data z chytrých zařízení do mobilních aplikací, aplikací chytrých televizí apod.
- Server bude umožňovat kompresi multimediálních dat společně s konverzí formátů.

## 4.1 Komunikační model a přenosové technologie

Chytrá domácnost je považována za uzavřený systém, který pracuje pouze s daty získanými z domácích chytrých zařízení a měřicích systémů. Domácí síť je ve své podstatě heterogenní síť, která obsahuje jednu páteřní síť a jednu nebo více podsítí. Centrálním bodem sítě je ve velké většině případů směrovač, který řídí veškerý tok dat. Ke směrovači je následně možné připojit jakékoliv zařízení splňující komunikační standard směrovače (viz obrázek 4.1). Bohužel k němu nelze připojit tzv. chytrá zařízení, která většinou komunikují prostřednictvím jiných technologií, než se používají ve standardních síťových směrovačích.




Obr. 4.1: Síťová architektura běžné domácí sítě

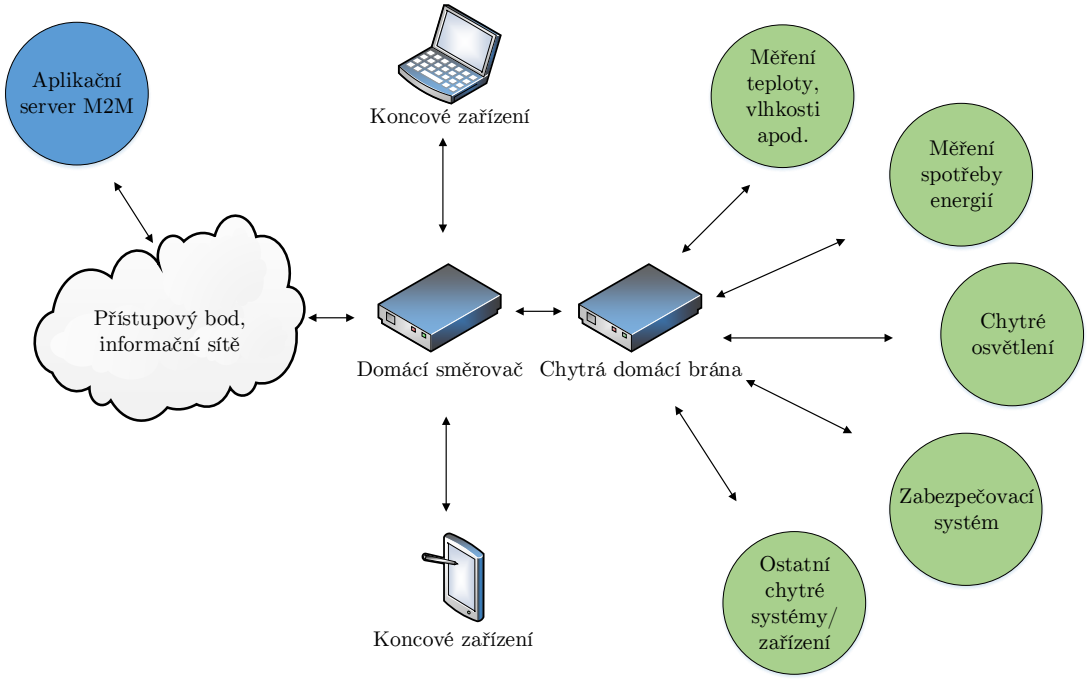
V případě integrace IoT do stávající domácí sítě se ke směrovači připojí chytrá domácí brána, která poskytuje konektivitu a softwarovou strukturu pro zpracování M2M dat – viz obr. 4.2. Brána se stane vstupním bodem pro veškerá chytrá zařízení a měřicí systémy v domácnosti. Každá sub-síť brány komunikuje na základě vlastních pravidel speciálně nastavených dle požadavku aplikace.

Druhou možností je vyvinutí jediného zařízení zajišťujícího nejen směrování v rámci domácí sítě, ale i funkci chytré domácí brány – viz obrázek 4.3. Popsaným spojením vznikne tzv. chytrý domácí server, který zajistí celkovou správu domácí sítě, komunikaci mezi zařízeními, agregaci dat, jejich analýzu a komunikaci s aplikací poskytovatele služby.

Záměrem práce bylo takový systém navrhnout a experimentálně prověřit použitelnost protokolu SIP pro komunikaci chytrého domácího serveru s aplikacemi v

sítích telekomunikačních operátorů.

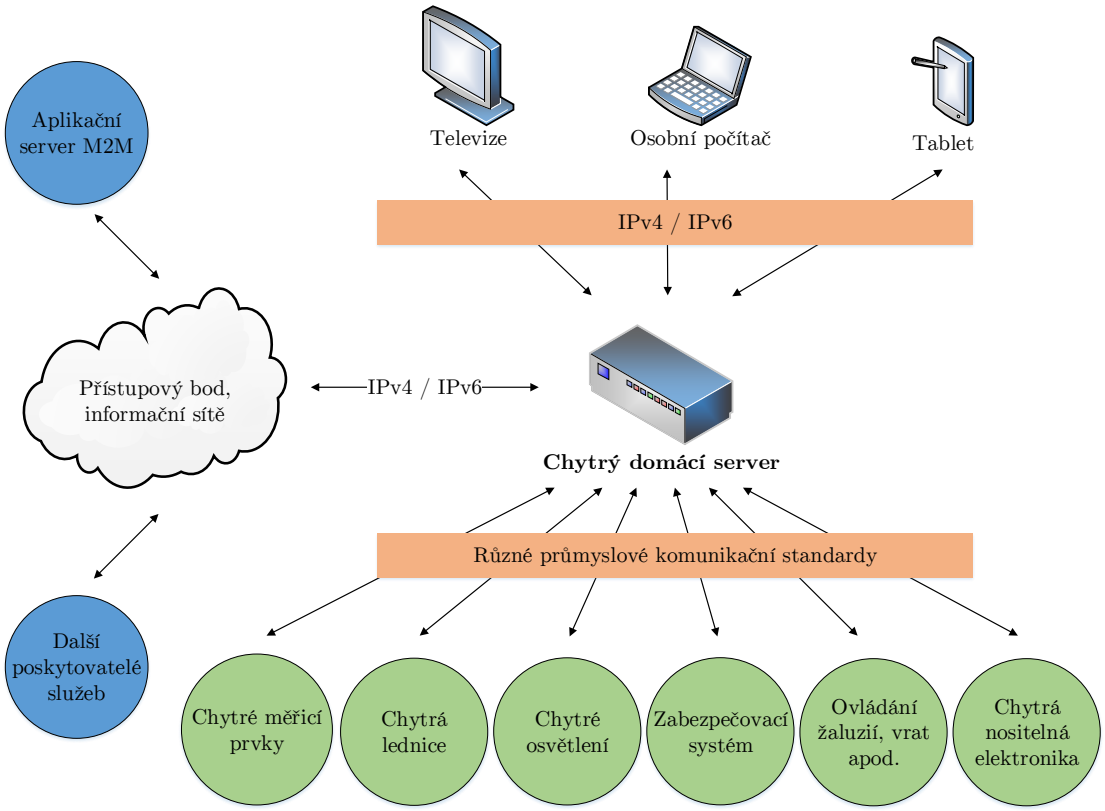
Z uvedeného vyplývá, že pro chytrý domácí server dle obrázku 4.3 je vhodné využít komunikační model D2G popsaný v kapitole 2.3. Chytrý domácí server v tomto případě řídí veškerou komunikaci v domácnosti, sbírá nejrůznější informace ze všech lokálních sítí a zpracovává je. V závislosti na funkcionalitě je může dále odesílat poskytovateli služby.




Obr. 4.2: Síťová architektura s odděleným domácím směrovačem a chytrou domácí bránou, mezistupeň vyvinutý pro Telekom Austria Group

Vedle síťové architektury je nutné stanovit přenosové technologie, které musí být v rámci chytrého domácího serveru dostupné [46]. Přenosová technologie slouží ke komunikaci chytrého domácího serveru s okolím a lze ji rozdělit na dvě oblasti:

1. komunikace chytrého domácího serveru s vnější a vnitřní sítí IP (poskytovatel služby, datové centrum, webový server pro zobrazení informací, domácí spotřebiče - PC, mobilní telefon apod.),
2. komunikace chytrého domácího serveru s vnitřní sítí IoT, prvky chytré domácnosti (senzory, měřicí zařízení, zabezpečovacími systém, stínící technika, chytré spotřebiče apod.).




Obr. 4.3: Síťová architektura s chytrým domácím serverem

### 4.1.1 Komunikace chytrého domácího serveru s vnější a vnitřní IP sítí

Ke komunikaci chytrého domácího serveru s vnější i vnitřní sítí IP lze uvažovat o přenosových technologiích:

- WAN Ethernet,
- LAN Ethernet,
- ADSL/VDSL,
- 2G/3G/4G sítě,
- WLAN 802.11 a/b/g/n/ac.

V závislosti na typu přípojného bodu musí být implementována konkrétní přenosová technologie. Pro běžné zabezpečení funkcionality z uvedeného seznamu by měly být integrovány alespoň WAN Ethernet, LAN Ethernet a WLAN 802.11 a/b/g/n/ac [46] .

### 4.1.2 Komunikace chytrého domácího serveru s vnitřní sítí Internetu věcí

Komunikace chytrého domácího serveru s vnitřní sítí IoT lze definovat jako komunikaci serveru se zařízeními chytré domácnosti, mezi které patří domácí automatizované prvky, měřicí systémy apod. V této oblasti se dnes běžně používají technologie:

- Wireless M-Bus,
- ZigBee,
- Z-Wave,
- Bluetooth, BLE.

Mezi další velmi perspektivní technologie se řadí LoRa a Sigfox. Ty se zaměřují na komunikaci na větší vzdálenosti a dle specifikací nepočítají s použitím domácího chytrého serveru [47]. Dále bude uveden stručný popis přenosových technologií používaný v současných domácích zařízeních a spotřebičích, se kterými musí chytrý domácí server komunikovat.

#### Bluetooth

Bluetooth dle specifikace standardu IEEE 802.15.1 do verze 4.0 je vhodný pro aplikace vyžadující vyšší přenosové rychlosti. Na rozdíl od jiných protokolů nezohledňuje spotřebu zařízení a není tedy vhodný pro zařízení s požadavkem nízké spotřeby energie. Bluetooth 4.0 definuje nový mechanizmus pro snížení spotřeby energie nazývaný BLE (Bluetooth Low Energy). BLE používá pro různá zařízení odlišné stupně spotřeby energie [48]. BLE není, oproti svému předchůdci, navržen pro přenos souborů, ale je vhodný pro přenos krátkých zpráv. Předpokládá se, že do roku 2018 bude BLE integrováno v devadesáti procentech chytrých mobilních telefonů se systémy iOS, Android a Windows [49]. Chytrý domácí server může tento protokol využít s chytrým telefonem, nicméně v současné době se jeví jako vhodnější použití vnitřní sítě IP a připojení prostřednictvím technologie Wi-Fi.

#### IEEE 802.11

IEEE 802.11 - Wi-Fi patří k nejrozšířenějším komunikačním standardům bezdrátové komunikace v domácnostech. Jeho nespornou výhodou je vysoká přenosová rychlost a nízké náklady na jeho nasazení. V současné době je nejčastěji využíván standard 802.11n díky jeho propustnosti (řádově stovky Mbit/s), který je vhodný pro přenos velkých souborů. Vysoká propustnost má za následek velkou spotřebu energie [50].

Wi-Fi komunikace je v současné době, vzhledem k jejímu rozšíření, vhodná využít pro komunikaci aplikace mobilního telefonu s chytrým domácím serverem za účelem získání naměřených dat či řízení chytrých domácích zařízení.

#### ZigBee

ZigBee dle specifikace standardu IEEE 802.15.4 je primárně navržena ke komunikaci zařízení s nízkou spotřebou elektrické energie [51]. Technologie je určena zejména pro průmyslové prostředí, v domácnostech je využívána prozatím pouze zřídka, většinou je aplikována v různých druzích senzorů. Fyzickou vrstvu pro standard ZigBee definuje standard IEEE 802.15.4 [52].

#### Wireless M-BUS

Wireless M-BUS slouží ke vzájemné komunikaci zařízení v sítích M2M a IoT. V porovnání s technologií ZigBee potřebuje pro provoz méně elektrické energie [53], [54]. Je implementován v celé řadě chytrých měřicích systémů v oblasti měření a regulace topných systémů, plynu, odběru vody a elektrické energie. Je téměř nezbytné, aby právě technologie Wireless M-BUS byla integrovaná do každého chytrého domácího serveru s ohledem na interoperabilitu s instalovanými měřicími systémy domácností.

### 4.1.3 Komunikační protokol aplikační vrstvy

Posledním bodem, který je nutné v oblasti komunikace v rámci IoT vyřešit, je komunikační (signalizační) protokol na aplikační vrstvě, který bude sloužit ke komunikaci chytrého domácího serveru jak ve vnitřní síti, tak i s vnější sítí. Možné komunikační protokoly byly podrobně rozebrány v kapitole 3. Při požadavku využití chytrého domácího serveru v sítích telekomunikačních operátorů se jeví jako výhodné použití protokolu SIP, který je široce zastoupen v oblasti internetové telefonie - VoIP [55].

Využitím protokolu SIP by měl být zcela splněn stěžejní požadavek provozu chytrého domácího serveru bez zásahu do síťové infrastruktury telekomunikačního operátora. Infrastruktura současných telekomunikačních sítí má ve velké většině případů integrovaný právě protokol SIP dle RFC3261[32]. Z logických entit je vždy integrovaný minimálně registrační server SIP registrar a jeden či více SIP Proxy serverů. V základním řešení jsou akceptované metody definované v RFC3261 (INVITE, ACK, BYE, CANCEL, REGISTER, OPTIONS) a v RFC3428 (MESSAGE). Některé sítě mají integrovány další metody, jako například SUBSCRIBE, NOTIFY, PUBLISH

apod. – není to však podmínkou. Návrhem komunikace na aplikační vrstvě chytrého domácího serveru se podrobně věnuje kapitola 3.3.3.

## 4.2 Softwarové požadavky

Chytrý domácí server řídí software, který má na starosti zajištění komplexní komunikace v rámci vnitřních i vnějších IoT sítí, práci se získanými daty a distribuci těchto dat. Software je společně s hardwarem velmi důležitý pro správnou funkčnost chytrého domácího serveru.

Pro vývoj systémů IoT a jejich funkcí se nejčastěji používají softwarové struktury, které slouží jako podpora při programování. Ty ve většině případů nabízí rozhraní pro programování aplikací - API (Application Programming Interface). Dostupných softwarových struktur je v současnosti celá řada, velmi se liší nabízenými službami a s tím spojenou použitelností. Nadnárodní technologické, softwarové a telekomunikační společnosti operující v oblasti IoT se spojují společně s vývojovými společnostmi do větších konsorcií, které mají za cíl sjednotit komunikaci v rámci IoT a vytvořit jednotnou, obecně použitelnou a volně dostupnou softwarovou strukturu k vývoji IoT systémů. K nejvýznamnějším konsorciím patří aliance OSGi (Open Service Gateway initiative) [56], Allseen Alliance [15] a IoTivity [13]. Tato konsorcia poskytují vlastní pohled na technologii IoT a M2M a nabízí unikátní řešení, jak implementovat IoT v praxi. Pro demonstraci možnosti vývoje byla vybrána softwarová struktura vyvíjená aliancí OSGi [56] zejména z důvodů:

- jedná se o velmi robustní softwarovou strukturu s širokým spektrem použitelnosti,
- je otevřená (open-source) => jsou dostupné veškeré zdrojové kódy a lze ji upravit k vlastní potřebě,
- je kompletně vyvinuta v programovacím jazyku JAVA.

### Aliance OSGi

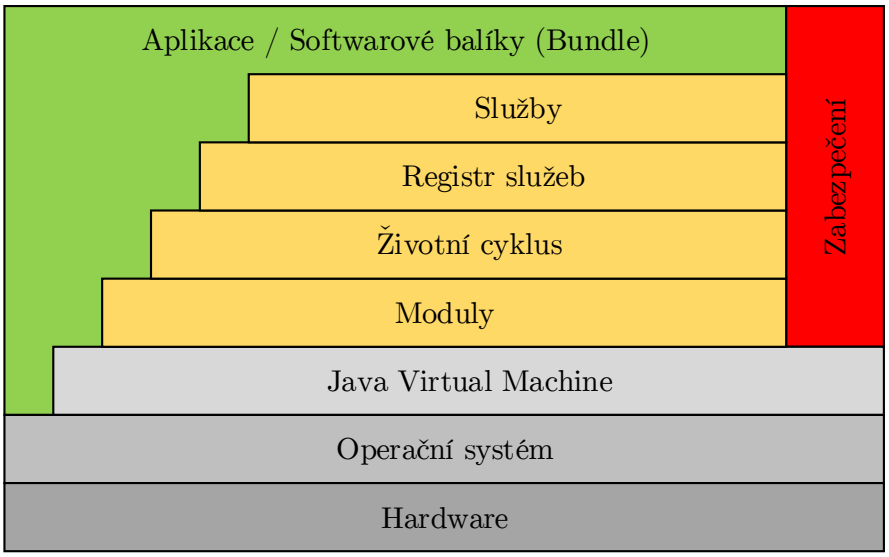
Aliance OSGi je konsorcium společností zabývajících se novými technologiemi v oblasti IoT a vývojem průmyslových standardů pro IoT. Přispívají do něj společnosti, mezi něž patří například Adobe, Oracle, Huawei Technologies, Inc., IBM apod. Hlavním produktem aliance OSGi je softwarová struktura OSGi specifikující dynamický modulární systém pro programovací jazyk JAVA.

## 4.3 Softwarová struktura OSGi

Softwarová struktura OSGi (framework OSGi) umožňuje instalaci a odebírání softwarových balíků (bundlů) za běhu, definuje životní cyklus modulu a nabízí infrastrukturu pro spolupráci modulů skrze služby. V současné době je framework OSGi považován za nejvyspělejší modulární systém pro jazyk JAVA. Poskytuje prostředí pro práci s libovolnou aplikací libovolného výrobce, poskytovatele služby či operátora a umožňuje vývoj nových aplikací. Každý softwarový balík poskytuje vždy jednu konkrétní funkci, např. FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), SIP apod. [57]

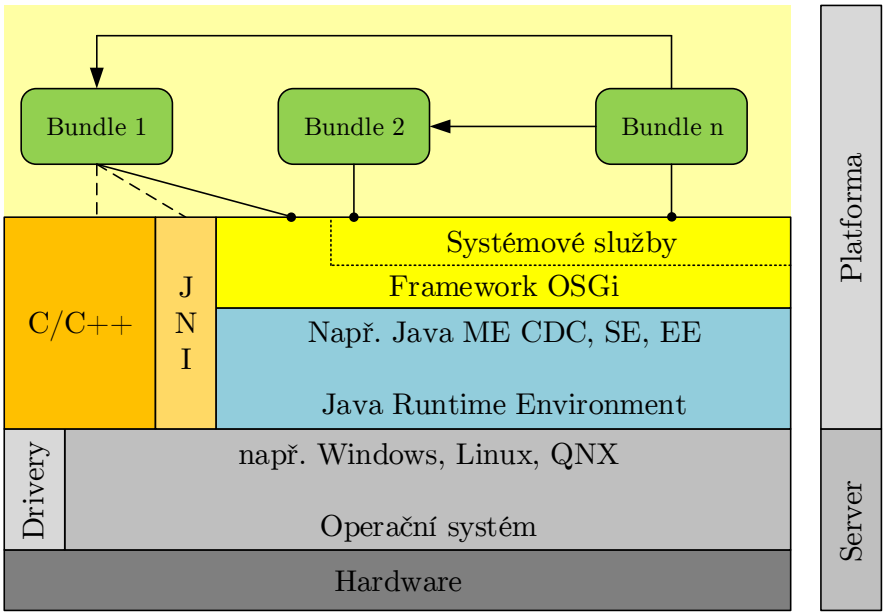
### 4.3.1 Obecná architektura softwarové struktury OSGi

Architektura softwarové struktury OSGi je znázorněna na obrázku 4.4 a zahrnuje:



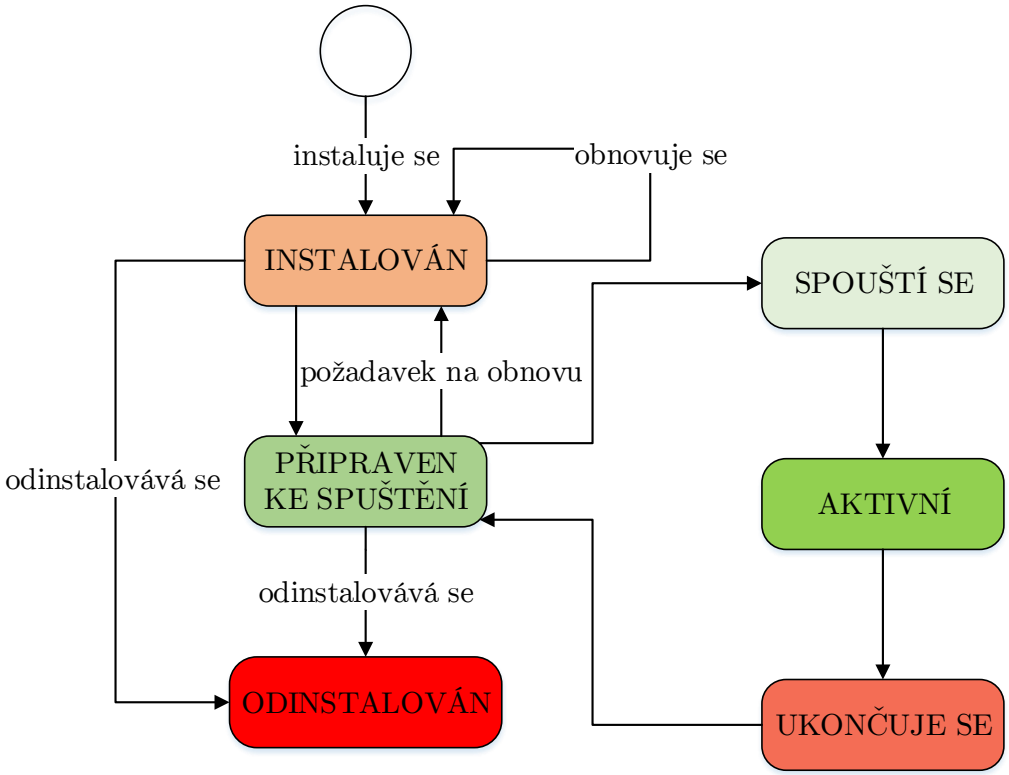

Obr. 4.4: Referenční model OSGi

- **Softwarový balík (Bundle)** – Softwarový balík je aplikace naprogramovaná v jazyce JAVA a zkompilovaná do souboru s příponou JAR. Oproti standardní aplikaci v jazyce JAVA má navíc speciální hlavičku. Každý softwarový balík představuje dynamicky spustitelnou aplikaci složenou ze tříd a konfiguračního souboru, který explicitně deklaruje závislost softwarového balíku na externích proměnných, jak je znázorněno na obrázku 4.5 [56], [58].




Obr. 4.5: Softwarový balík (Bundle) a jeho začlenění

- **Služby** – Vrstva služeb tvoří dynamické závislosti softwarových balíků. Služba je libovolný objekt POJO (Plain Old Java Objects) a je spuštěna současně se softwarovým balíkem.
- **Registry služeb** – Registr služeb poskytuje rozhraní aplikace pro řízení služeb.
- **Životní cyklus** – Každý softwarový balík prochází striktně definovaným životním cyklem obdobně, jako je tomu u appletů JAVA. Applet JAVA je program v jazyce JAVA, který je možné spustit ve virtuálním prostředí jazyku JAVA (Java Virtual Machine - JVM) při návštěvě webové stránky v internetovém prohlížeči. Pro softwarový balík jsou definovány stavy, jejichž vzájemnou provázanost popisuje obrázek 4.6. Jednotlivé stavy jsou definovány následovně:
  - INSTALOVÁN (INSTALLED) – Ve stavu INSTALOVÁN se softwarový balík nachází po instalaci softwarového balíku.
  - PŘIPRAVEN KE SPUŠTĚNÍ (RESOLVED) – Ve stavu PŘIPRAVEN KE SPUŠTĚNÍ se softwarový balík nachází tehdy, kdy již byl nainstalován a jeho všechny požadované prerekvizity jsou dostupné. Pokud je tento stav vykonán, softwarový balík může být spuštěn.
  - SPOUŠTÍ SE (STARTING) – Stav SPOUŠTÍ SE indikuje spouštění softwarového balíku. Pokud proběhne vše bez komplikací a při spouštění




Obr. 4.6: Stavový diagram bundlu OSGi

softwarového balíku nenastane chyba, softwarový balík plynule přejde do stavu AKTIVNÍ.

- AKTIVNÍ (ACTIVE) – Softwarový balík je spuštěn.
- UKONČUJE SE (STOPPING) – Softwarový balík se ukončuje. Po korektním ukončení softwarový balík přechází do stavu PŘIPRAVEN KE SPUŠTĚNÍ.
- ODINSTALOVÁN (UNINSTALLED) – Ve stavu UNINSTALLED se softwarový balík nachází v případě, kdy není daným softwarovým balíkem spravován.

- **Moduly** – V rámci OSGi je nemožné, aby spolu softwarové balíky navzájem přímo spolupracovaly. Sdílet spolu můžou pouze to, co předem definují a tuto funkci tzv. exportují ze softwarového modulu. Vrstva modulů definuje, jakým způsobem je možné importovat či exportovat služby softwarových modulů.
- **Zabezpečení** – Vrstva zabezpečení se stará o bezpečnostní aspekty celé aplikace.

- **Prostředí pro běh programu - Java Virtual Machine** – definuje způsoby, jakými jsou jednotlivé metody a třídy dostupné pro specifickou platformu.

### 4.3.2 Implementace softwarové struktury OSGi

Jednou z velkých výhod softwarové struktury OSGi (frameworku OSGi) je její kompletní implementace v jazyce JAVA. Výhoda jazyku JAVA je v možnosti jeho použití téměř kdekoli. V současné době existuje mnoho implementací frameworků OSGi a to jak komerčních, tak i volně dostupných (open-source). Dostupné frameworky včetně podpory různých modulů OSGi jsou uvedeny v tabulce 4.1 [59], [60]. K dalšímu porovnání byly z výčtu v tabulce 4.1 vybrány pouze volně dostupné frameworky s největší podporou modulů. Jednalo se o:

- Eclipse Equinox [61],
- Knopflerfish [62],
- Apache Felix [63].

Tyto frameworky mají velký potenciál budoucího využití [64], a proto byly v rámci řešení habilitační práce porovnány z hlediska jejich výkonnosti. Výsledky porovnání popisuje kapitola 4.3.3.

Tab. 4.1: Softwarové struktury OSGi, podporované moduly

| | Apache Felix (4.4)<br>Open source [63] | Eclipse Equinox (4.5)<br>Open source [61] | JBoss (1.0)<br>Open source [65] | Hitachi (SuperJ)<br>komerční | Knopflerfish (5.1)<br>Open source [62] | ProSyst (SDK 7.3)<br>Commercial |
|---|---|---|---|---|---|---|
| Security (latest v. 1.8) | 1.7 | 1.8 | x | x | 1.7 | 1.5 |
| Core Framework (latest v. 6.0) | 5.0 | 6.0 | 4.2 | 4.x | 5.1 | 4.2 |
| Package Admin Service (latest v. 1.2) | 1.2 | 1.2 | 1.2 | x | 1.2 | 1.2 |
| Start Level Service (latest v. 1.1) | 1.1 | 1.1 | 1.2 | x | 1.2 | 1.2 |
| Conditional Admin Service (latest v. 1.1) | 1.1 | 1.1 | x | x | 1.1 | 1.1 |
| Permission Admin Service (latest v. 1.2) | 1.2 | 1.2 | x | x | 1.2 | 1.2 |
| URL Handler Service (latest v. 1.0) | 1.0 | 1.0 | 1.0 | x | 1.0 | 1.0 |
| Log Service (latest v. 1.3) | 1.3 | 1.3 | x | x | 1.3 | 1.3 |
| HTTP Service (latest v. 1.2) | 1.2 | 1.2 | 1.2 | x | 1.2 | 1.2 |
| Device Access Service (latest v. 1.1) | x | 1.1 | x | x | 1.1 | 1.1 |
| Configuration Admin Service (latest v. 1.5) | 1.3 | 1.3 | 1.3 | x | 1.5 | 1.3 |
| Meta-type Service (latest v. 1.2) | 1.1 | 1.2 | x | x | 1.2 | 1.1 |
| Preference Service (latest v. 1.1) | 1.1 | 1.1 | x | x | 1.1 | 1.1 |
| User Admin Service (latest v. 1.1) | 1.1 | 1.1 | x | x | 1.1 | x |
| Wire Admin Service (latest v. 1.0) | 1.0 | x | x | x | 1.0 | 1.0 |
| IO Connector Service (latest v. 1.3) | 1.3 | x | x | x | 1.3 | 1.3 |
| Initial Provisioning Service (latest v. 1.2) | 1.2 | x | x | x | 1.2 | 1.2 |
| UPnP Device Service (latest v. 1.2) | 1.1 | x | x | x | 1.2 | 1.1 |
| Declarative Service (latest v. 1.2) | 1.2 | 1.2 | x | x | 1.2 | 1.1 |
| Event Admin Service (latest v. 1.3) | 1.3 | 1.3 | x | x | 1.3 | 1.2 |
| Deployment Admin Service (latest v. 1.1) | 1.1 | x | x | x | 1.1 | 1.1 |
| Auto Configuration Service (latest v. 1.0) | x | x | x | x | 1.0 | 1.0 |
| Application Admin Service (latest v. 1.1) | x | 1.1 | x | x | 1.1 | 1.1 |
| DMT Admin Service (latest v. 2.0) | x | x | x | x | 2.0 | 1.0 |
| Monitor Admin Service (latest v. 1.0) | x | x | x | x | 1.0 | 1.0 |
| Foreign Applications Access (latest v. 1.0) | x | x | x | x | 1.0 | 1.0 |
| Blueprint Container (latest v. 1.0) | x | 1.0 | x | x | x | x |
| JTA Service (latest v. 1.0) | 1.0 | x | 1.0 | x | x | x |
| Web Applications (latest v. 1.0) | x | x | 1.0 | x | x | x |
| Coordinator Service Spec. (latest v. 1.0) | x | 1.0 | x | x | x | x |
| Repository Service Spec. (latest v. 1.0) | x | x | 1.0 | x | 1.0 | x |
| Subsystem Service Spec. (latest v. 1.0) | 1.0 | x | x | x | x | x |
| Resolver Service Spec. (latest v. 1.0) | 1.0 | x | x | x | x | x |

### 4.3.3 Výkonnost implementací softwarové struktury OSGi

K výkonnostnímu testování implementací softwarové struktury OSGi (frameworků) byly vybrány tři zmíněné: Apache Felix, Eclipse Equinox a Knopflerfish.

#### Metodika testování

Výkonnostní charakteristika se ve velké řadě případů testuje na službách HTTP. HTTP server běžící na výkonnostně omezeném zařízení reprezentuje klíčovou funkcionalitu. Pro testování tedy byly vybrány jednoduché implementace serveru HTTP:

- Jetty Web Server,
- Jetty Embedded HTTP bundle,
- Knopflerfish HTTP bundle.

Zmíněné tři implementace jsou běžně používané a jsou standardní součástí většiny frameworků OSGi. Na zvolený hardware, mikropočítač Raspberry Pi [66], byl nainstalován server HTTP, který při testování hraje důležitou roli – přijímá žádosti klientů, vyhledává v lokální databázi a odesílá nalezené výsledky.

K testování byl použit hardwarový tester Spirent Avalanche 3100 B [67] určený k testování síťové infrastruktury, webových aplikací, kvality služeb QoS, služeb Triple play, bezpečnosti síťových prvků, ke generování reálného provozu a distribuovaným útokům odepření služby DDoS (Distributed Denial of Service).

#### Testovací Servlety

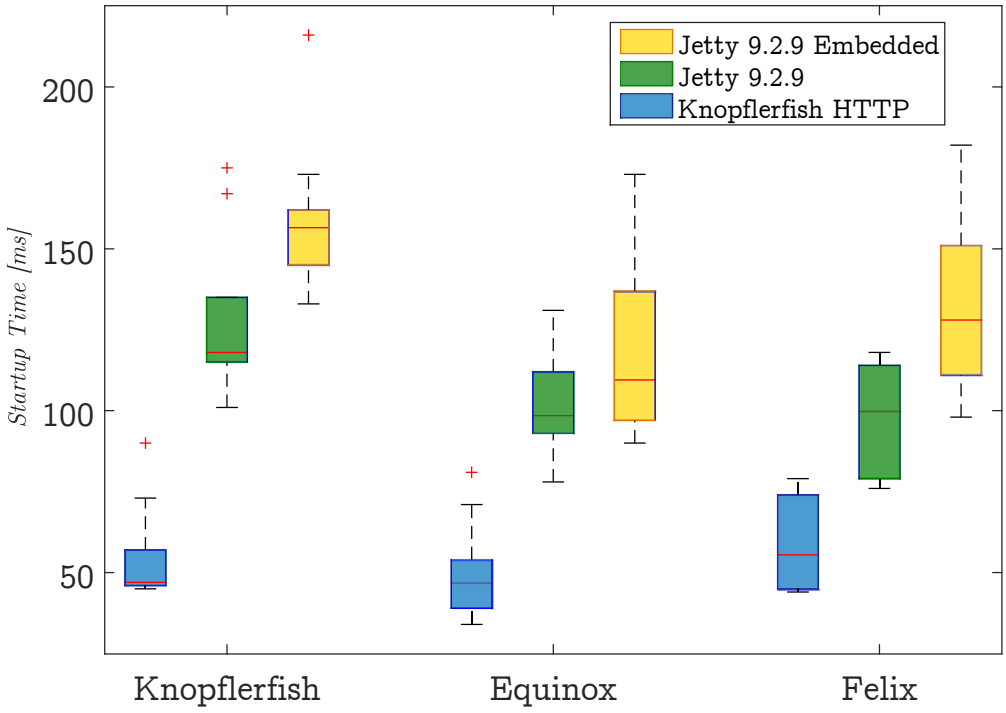
K testování byly vyvinuty dvě jednoduché webové aplikace v podobě servletů poskytující odpověď na metodu HTTP GET s textem OK. Všechny servery byly zpřístupněny přes port 8080, servlety byly registrovány na adrese `http://adresa_serveru:8080/test`. První servlet využíval standardní službu OSGi HTTP a standardní sledovací nástroj ze servisní platformy OSGi. Sledovací nástroj byl využit k manipulaci s životním cyklem servletu od jeho inicializace až k jeho ukončení. Druhý servlet byl využit s Jetty serverem.

#### Testování času potřebného ke spuštění služby HTTP

Jedním z důležitých parametrů potřebným k porovnání různých frameworků je doba potřebná ke spuštění softwarového balíku. K měření doby spuštění byl vyvinut speciální softwarový balík, který měřil čas potřebný k úplnému načtení softwarového balíku zajišťujícího funkcionalitu HTTP serveru. Vytvořený měřicí softwarový balík

využívá metodu pro zpracování událostí OSGi Bundle Listener k získání informací o změně stavu jiného softwarového balíku. Jeho funkce je taková, že nejprve zastaví veškeré služby serveru a následně spustí vybraný server a měří rozdíl časů mezi přechodem softwarového balíku ze stavu PŘIPRAVEN KE SPUŠTĚNÍ do stavu AKTIVNÍ.

Každý softwarový balík daného serveru HTTP byl testován desetkrát, každý test byl proveden pro analyzované frameworky OSGi. Výsledky měření jsou znázorněné v grafu na obrázku 4.7. Krabicový graf na obrázku 4.7 obsahuje v každém obdélníku




Obr. 4.7: Doba potřebná ke spuštění HTTP bundlu

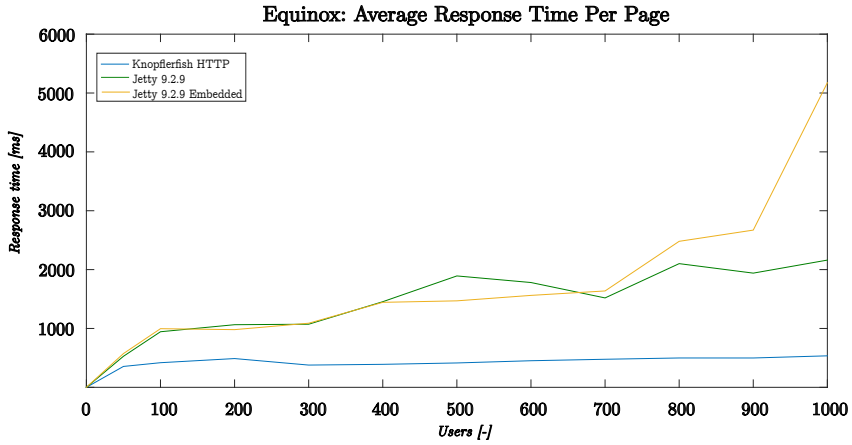
červenou linku, která označuje medián naměřené hodnoty, kraje obdélníků představují 25. a 75. percentil, hraniční čáry definují extrémní hodnoty minim a maxim, body označené červeným symbolem + jsou hodnoty změřené mimo standardní rozsah. Z grafu je patrné, že časy potřebné ke spuštění softwarového balíku jsou v jednotlivých frameworcích OSGi velmi podobné.

Z hlediska doby potřebné pro spuštění služby HTTP dosahuje nejlepších časů služba Knopflerfish HTTP. Spuštění služby Knopflerfish HTTP spotřebuje pouze polovinu času, než služba Jetty či Jetty Embedded. Průměrný čas potřebný pro spuštění služby Knopflerfish HTTP byl při měření $55,1\,ms$. U služeb Jetty a Jetty

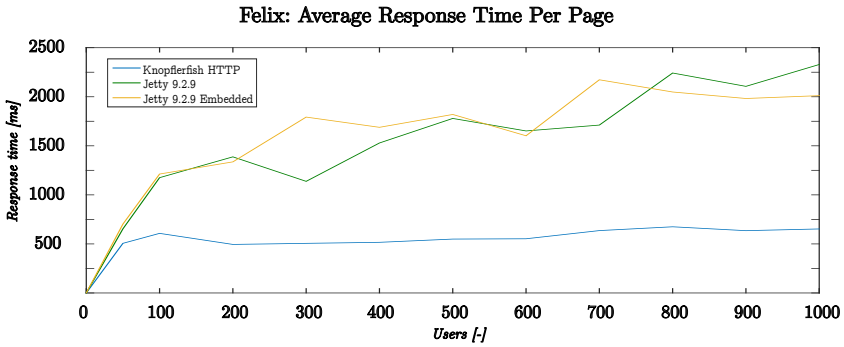
Embedded průměrný čas pro spuštění služby dosahoval $152,2\,ms$ a $158,6\,ms$. Časy pro spuštění softwarového balíku jsou velmi ovlivněny jejich celkovou velikostí [68].

### Testování doby odezvy na dotaz

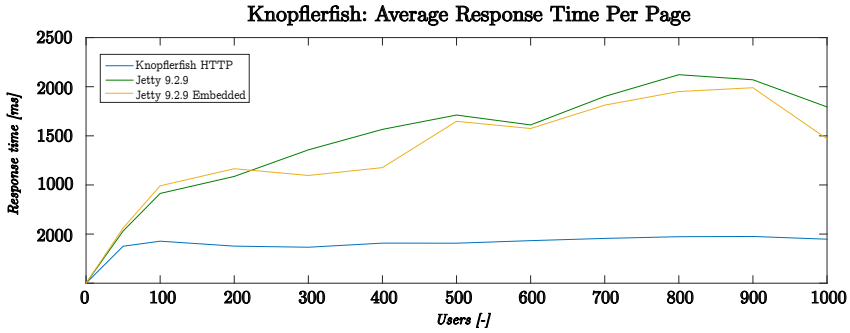
Klíčová hodnota, která má velký vliv na spokojenost uživatele - QoE (Quality of Experience), je reakční doba HTTP serveru – doba potřebná na zpracování dotazu a vygenerování relevantní odpovědi. Reakční doba byla měřena v závislosti na počtu uživatelů, výsledky jsou znázorněny v grafech na obrázcích 4.8, 4.9, 4.10.




Obr. 4.8: Průměrná doba odezvy HTTP serverů pro softwarový balík Equinox

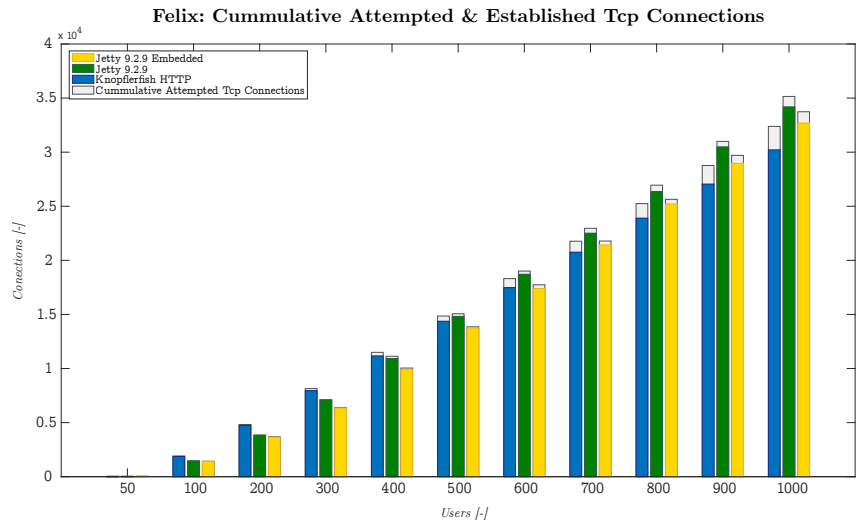



Obr. 4.9: Průměrná doba odezvy HTTP serverů pro softwarový balík Felix

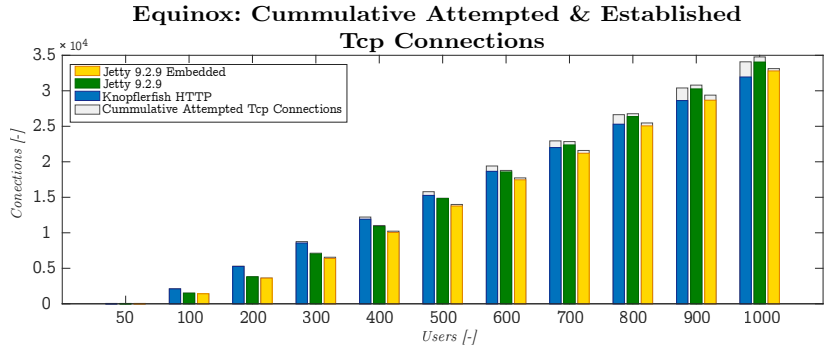



Obr. 4.10: Průměrná doba odezvy HTTP serverů pro softwarový balík Knopflerfish

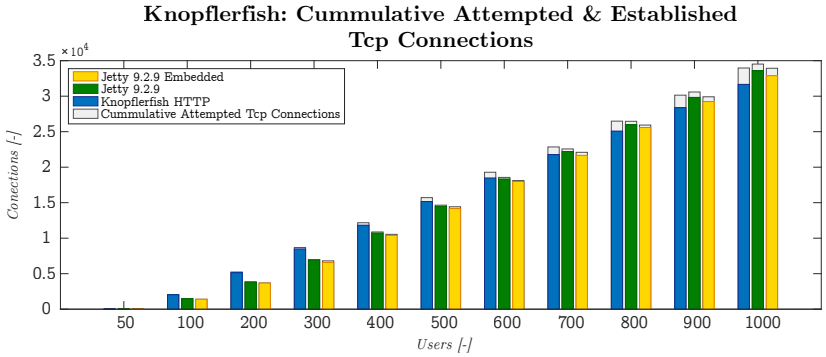
Z naměřených hodnot je patrné, že doba odezvy s počtem uživatelů roste, vyjma serveru HTTP Knopflerfish, kde doba odezvy zůstává téměř konstantní. Zmíněná diference je dána přístupem serverů k řízení zahlcení. Servery Jetty se pokouší všechny fronty obsloužit v jeden okamžik. Z toho důvodu narůstá jejich reakční doba s počtem uživatelů. Server Knopflerfish se chová rozdílně. V případě, že nemůže obsloužit všechny požadavky, automaticky resetuje TCP spojení. Následkem je největší počet resetovaných TCP spojení ze všech testovaných služeb – viz grafy na obrázcích 4.11, 4.12, 4.13.




Obr. 4.11: Počet kumulativních pokusů a sestavených spojení TCP pro softwarový balík Felix




Obr. 4.12: Počet kumulativních pokusů a sestavených spojení TCP pro softwarový balík Equinox



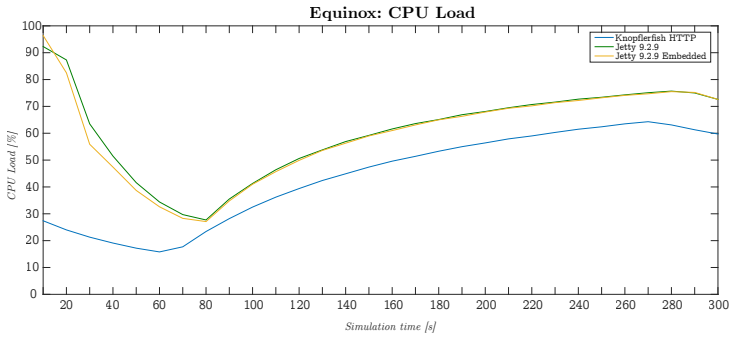

Obr. 4.13: Počet kumulativních pokusů a sestavených spojení TCP pro softwarový balík Knopflerfish

### Vytížení operační paměti a procesoru

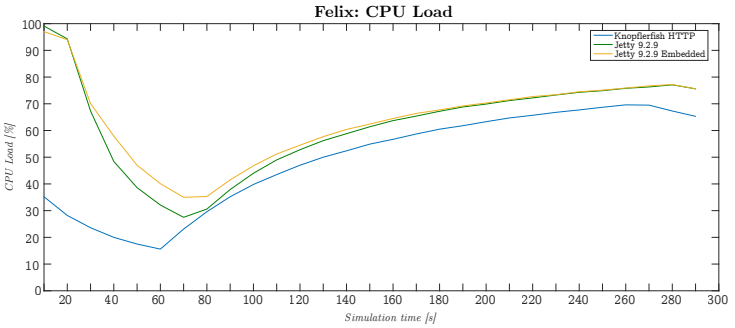
Velmi důležitým parametrem frameworku je alokace operační paměti a výpočetního výkonu procesoru. Grafy na obrázcích 4.14 a 4.17 zobrazují vytížení procesoru a operační paměti při použití frameworku Equinox, grafy na obrázcích 4.15 a 4.18 zobrazují vytížení procesoru a operační paměti při použití frameworku Apache Felix a grafy na obrázcích 4.16 a 4.19 zobrazují vytížení procesoru a operační paměti při použití frameworku Knopflerfish.

Z grafů je patrné, že klienti začali generovat požadavky HTTP šedesát sekund po zahájení měření. Výsledky testů jsou pro všechny tři frameworky velmi podobné. Nejnižší spotřeba paměti byla změřena u frameworku OSGi Knopflerfish se službou Knopflerfish HTTP. Při použití zbylých dvou serverových implementací Jetty byla spotřeba paměti téměř dvojnásobná.

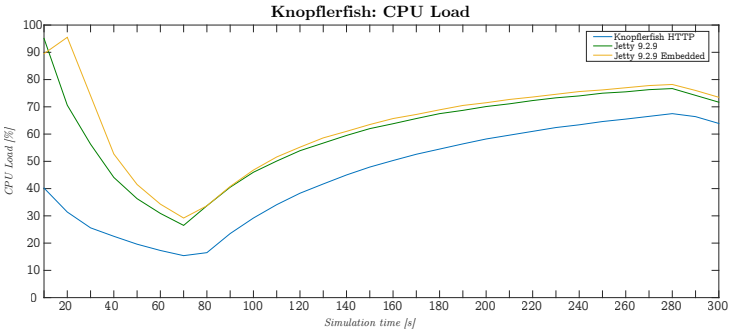
Vytížení procesoru vykazovalo při použití různých frameworků podobných výsledků. Nejvíce byl procesor využit frameworkem Knopflerfish. Obecně je procesor využit nejvíce při startu frameworků, což je zapříčiněno spouštěním všech jeho bundlů.



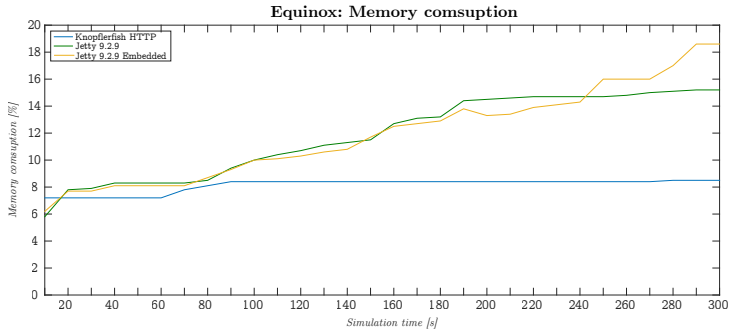

Obr. 4.14: Equinox – vytížení procesoru



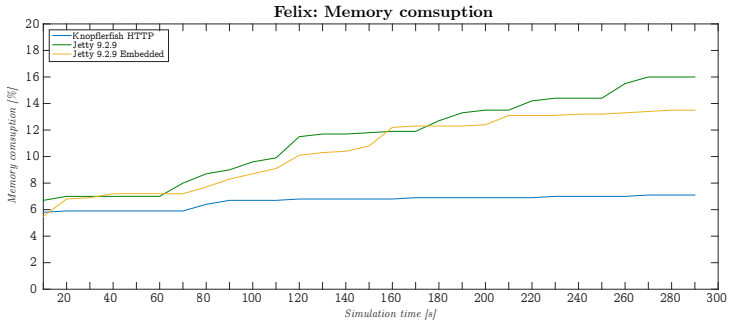

Obr. 4.15: Felix – vytížení procesoru



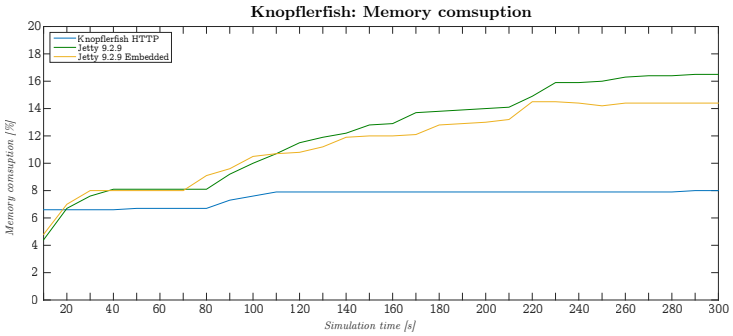

Obr. 4.16: Knopflerfish – vytížení procesoru




Obr. 4.17: Equinox – vytížení operační paměti




Obr. 4.18: Felix – vytížení operační paměti




Obr. 4.19: Knopflerfish – vytížení operační paměti

### 4.3.4 Výběr vhodné softwarové struktury k dalšímu použití

Z testování je patrné, že všechny tři testované frameworky mají velmi podobné časové i výkonnostní nároky. Pro použití lze tedy z hlediska výkon/čas doporučit kterýkoliv z vybraných. Z hlediska počtu podporovaných funkcí popsaných v tabulce 4.1 je vítězem framework Knopflerfish. Kromě největší podpory funkcí podporuje

knihovnu ASM byte-code [69]. Díky ní mohou být softwarové balíky modifikovány či automaticky obnovovány za chodu. Zmíněná funkce je vhodná zejména pro spouštění nové verze systému, kdy nemusí být opětovně spouštěna celá aplikace, ale pouze softwarové balíky, které byly upraveny.

## 4.4 Hardwarové požadavky

Hardwarové požadavky na chytrý domácí server se odvíjejí od aktuálně využívaných technologií v rámci domácností. Z předchozí části je zřejmé, že pro robustní řešení je vhodné použít softwarovou strukturu OSGi Knopflerfish, která běží na virtuálním stroji jazyku JAVA (JVM). Hardwarová struktura je tedy limitovaná a musí minimálně umožnit spuštění a běh javovského virtuálního stroje JVM, jehož minimální požadavky jsou dle [70] následující:

- architektura CPU - ARM,
- frekvence CPU - 1700 MHz (dual-core),
- paměť RAM: 512 MB (DDR3) / 128MB NAND FLASH,
- paměť pro SW: 256 MB (interní nebo externí v podobě SD karty, USB disku).

Pro jádro serveru je tedy možné použít libovolný mikropočítač splňující minimální požadavky uvedené výše. Dále je zapotřebí zajistit, aby server umožnil síťové připojení domácích zařízení. K tomu je nutná integrace přenosových technologií, které jsou v domácnostech využívány. Z důvodu možného budoucího rozšíření o jiné přenosové technologie je vhodné osadit server několika volnými USB porty, které lze v budoucnu použít pro připojení USB modulů s implementovanou technologií (anténa+vysílač+přijímač). Jako úsporné, finančně málo nákladné řešení chytrého domácího serveru, se jeví využití současných domácích směrovačů, do kterých se implementují funkce k obsluze chytré domácnosti. Bylo však zjištěno, že běžně dostupné domácí směrovače, brány či modemy, jsou založené na architektuře MIPS (Microprocessor without Interlocked Pipeline Stages), která hardwarově nedostačuje na spuštění JVM. Zařízení s architekturou ARM (Advanced RISC Machine) jsou zastoupena minimálně [71], [72]. Při použití frameworku OSGi je platforma ARM nutností pro bezproblémový chod virtuálního prostředí JVM [72]. Z uvedeného zjištění plynou dvě možnosti:

1. Využití současného hardware, mikropočítače, do kterého by se implementovaly pouze funkce chytré brány. Mikropočítač by se následně připojil do domácí sítě prostřednictvím technologie Ethernet. Toto zařízení jsme vyvinuli na zakázku jako mezistupeň cílového řešení.

2. Vývoj zcela nového hardwarového zařízení chytrého domácího serveru, který by integroval jak funkce domácího směrovače, tak i chytré brány. Protože cílem prací bylo ověřit možnosti protokolu SIP pro toto cílové řešení, nikoliv vývoj chytrého domácího serveru, byl pro ověření činnosti použit běžný domácí směrovač doplněný o rozšiřující modul. Spojení obou komponent simuluje funkci chytrého domácího serveru.

# 5 EXPERIMENTÁLNÍ OVĚŘENÍ NÁVRHU KOMUNIKACE CHYTRÉHO DOMÁCÍHO SERVERU

K experimentálnímu ověření navržené komunikace chytré domácnosti a poskytovatele služeb prostřednictvím protokolu SIP na aplikační vrstvě popsané v kapitole 3.3.3 byl vyvinut chytrý domácí server.

## 5.1 Hardwarové řešení chytrého domácího serveru

Hardware chytrého domácího serveru byl sestaven z běžně dostupného směrovače FRITZ!Box Fon WLAN 7390 [73], mikropočítače Raspberry Pi (verze B) [66] a obecného USB rozbočovače tak, aby splňoval požadavky definované v kapitole 4. Jednotlivé prvky byly navzájem propojeny prostřednictvím technologie Ethernet. Ke komunikaci byl tedy zvolen komunikační model s chytrým domácím serverem 4.3 popsaný v kapitole 4.1. Mikropočítač Raspberry Pi (verze B) (obrázek 5.1) využitý pro jádro serveru poskytuje:

- architekturu ARMv6,
- dvoujádrový mikroprocesor s frekvencí 700 MHz,
- paměť 512 MB RAM / 128 MB NAND FLASH,
- paměť pro data na kartě SD (do 128 GB),
- komunikační rozhraní Ethernet LAN (1x), JTAG, USB (2x),
- operační systém Raspbian (Linux).

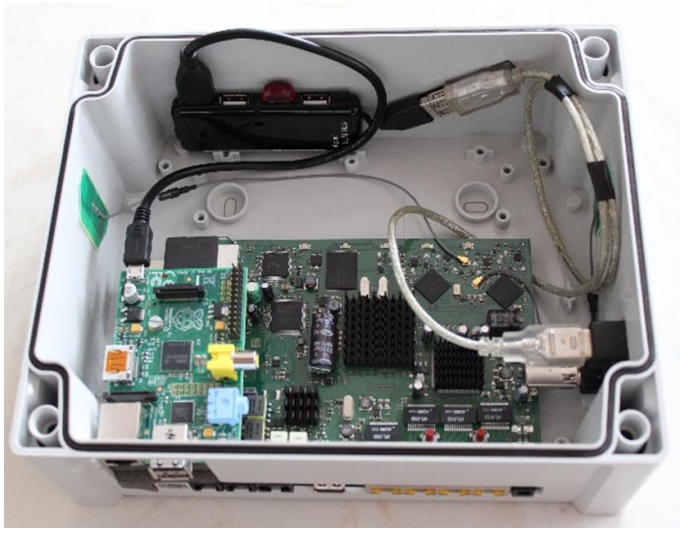
Směrovač FRITZ!Box Fon WLAN 7390 má nainstalovaný operační systém založený na linuxové distribuci OpenWRT [74]. Poskytuje přístup k síti Internet a lokální síť LAN. Mimo to má implementováno rozhraní pro vzdálenou konfiguraci směrovače prostřednictvím protokolu TR-069, který je v sítích telekomunikačních operátorů ve velké míře používán zejména pro vzdálenou aktualizaci zařízení připojených do sítě Internet [75]. Směrovač FRITZ!Box Fon WLAN 7390 poskytuje:

- architekturu MIPS,
- dvou jádrový mikroprocesor s frekvencí 500 MHz,
- paměť 128 MB RAM / 16 MB NAND FLASH,
- paměť pro data 512 MB,
- komunikační rozhraní Ethernet WAN (1x), LAN (4x), ADSL/VDSL (1x), 3G/4G slot (1x), WLAN 802.11 b/g/n, USB (2x),

- operační systém OpenWRT.

Navzájem propojené komponenty tvořily hardwarovou část chytrého domácího serveru – viz obrázek 5.2.

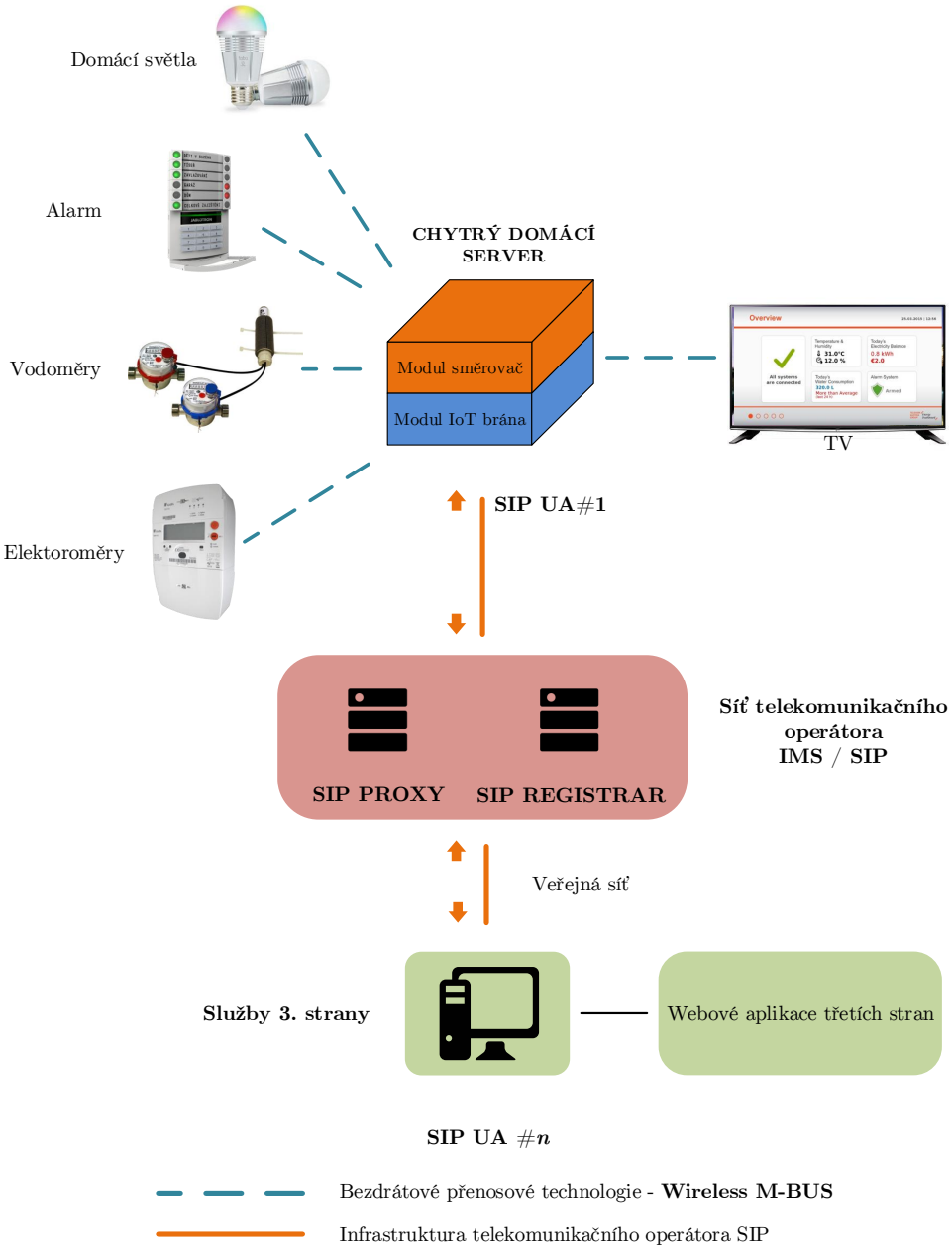
Obr. 5.1: Mikropočítač Raspberry Pi použitý pro jádro chytrého domácího serveru



Obr. 5.2: Hardwarová realizace testovacího vzorku chytrého domácího serveru

Po zvolení komunikačního modelu a hardwaru chytrého domácího serveru byla zvolena komunikační technologie Wireless M-BUS ke komunikace chytrého domácího serveru s měřicími senzory chytré domácnosti a technologie Wi-Fi ke komunikaci s ostatními prvky. Komunikační technologie Wi-Fi je součástí hardwarového řešení

směrovače, komunikační technologie Wireless M-BUS byla řešena připojením modulu USB Wireless M-BUS do jednoho z dostupných portů USB. Komunikace je popsána pomocí obrázku 5.3.

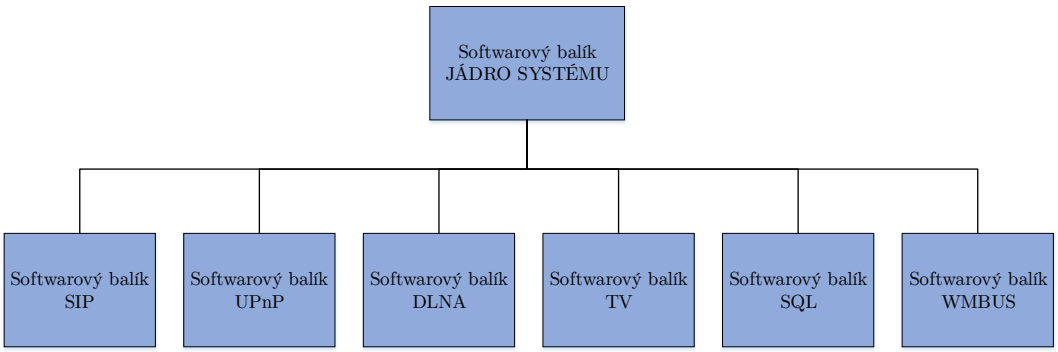



Obr. 5.3: Komunikační architektura s využitím chytrého domácího serveru pro Internet věcí

## 5.2 Softwarové řešení chytrého domácího serveru

Pro vývoj softwaru domácího chytrého serveru byla použita implementace softwarové struktury OSGi Knopflerfish z důvodu velké podpory modulů a dobrých výsledků výkonnostních testů dostupných v kapitole 4.3.3.

Softwarová architektura vyvinutého systému se skládá z jednotlivých softwarových balíků (bundlů). Každý softwarový balík řeší právě jednu problematiku, celkové řízení systému zaštiťuje jádro, tzv. core bundle. Přehled vyvinutých a testovaných softwarových balíků je na obrázku 5.4.

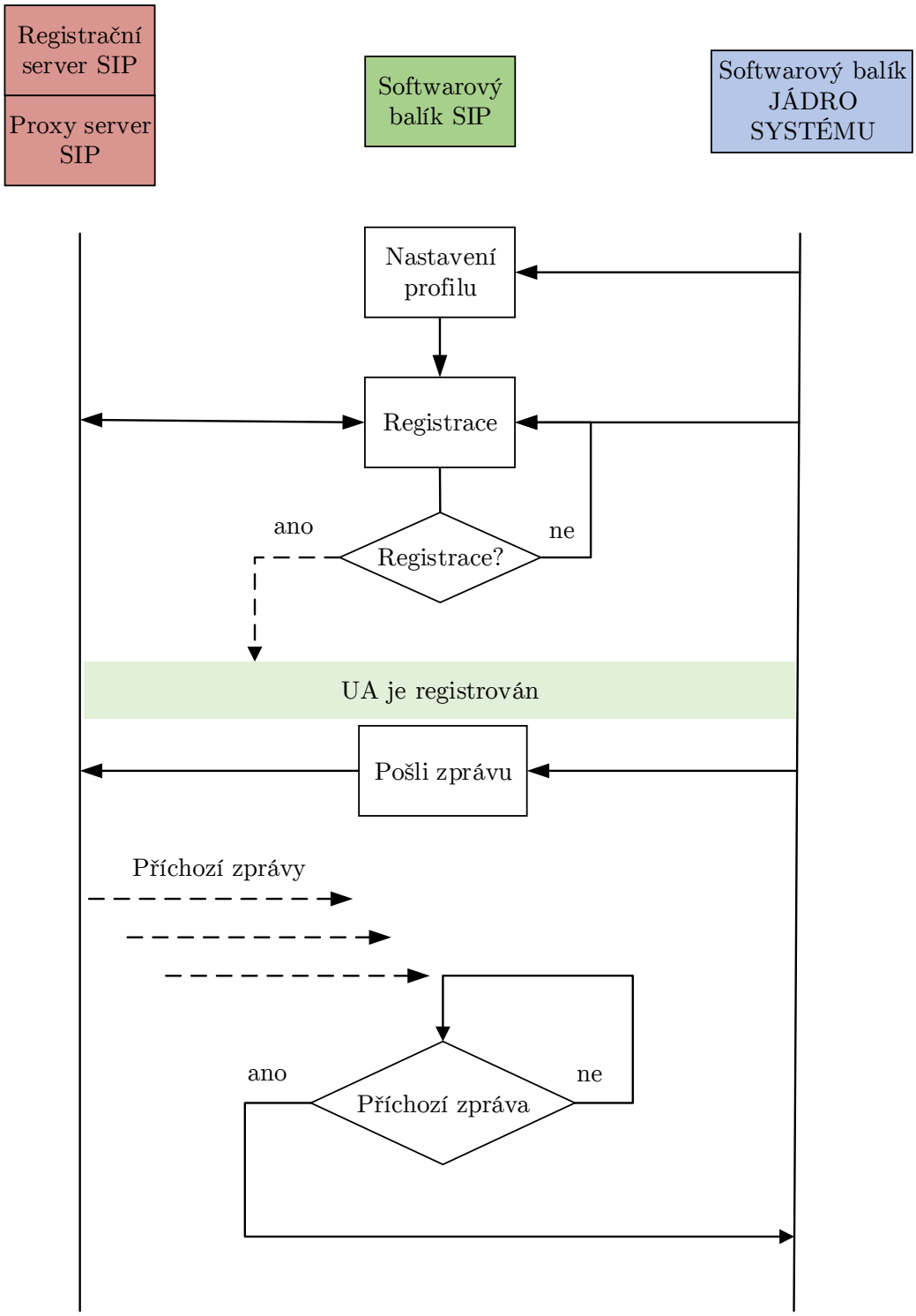



Obr. 5.4: Přehled vyvinutých softwarových balíků

### 5.2.1 Softwarový balík SIP

Softwarový balík SIP implementuje uživatelského agenta SIP, UA, který umožňuje komunikaci prostřednictvím zpráv SIP s jinými SIP UA. V softwarovém balíku jsou implementovány metody generující žádosti SIP REGISTER, SIP MESSAGE a metody generující odpovědi 200 OK, 401 UNAUTHORIZED a 407 PROXY AUTHENTICATION.

Softwarový balík SIP je složen ze čtyř hlavních metod a jedné události. Vývojový diagram na obrázku 5.5 zobrazuje procesy, které se v rámci softwarového balíku odehrávají. Po inicializaci softwarového balíku je volána metoda `setProfile` a `register`.




Obr. 5.5: Architektura softwarového balíku SIP

V nastavení profilu jsou zadány základní parametry pro registraci klienta SIP k registračnímu serveru SIP a následné komunikace přes proxy server SIP:

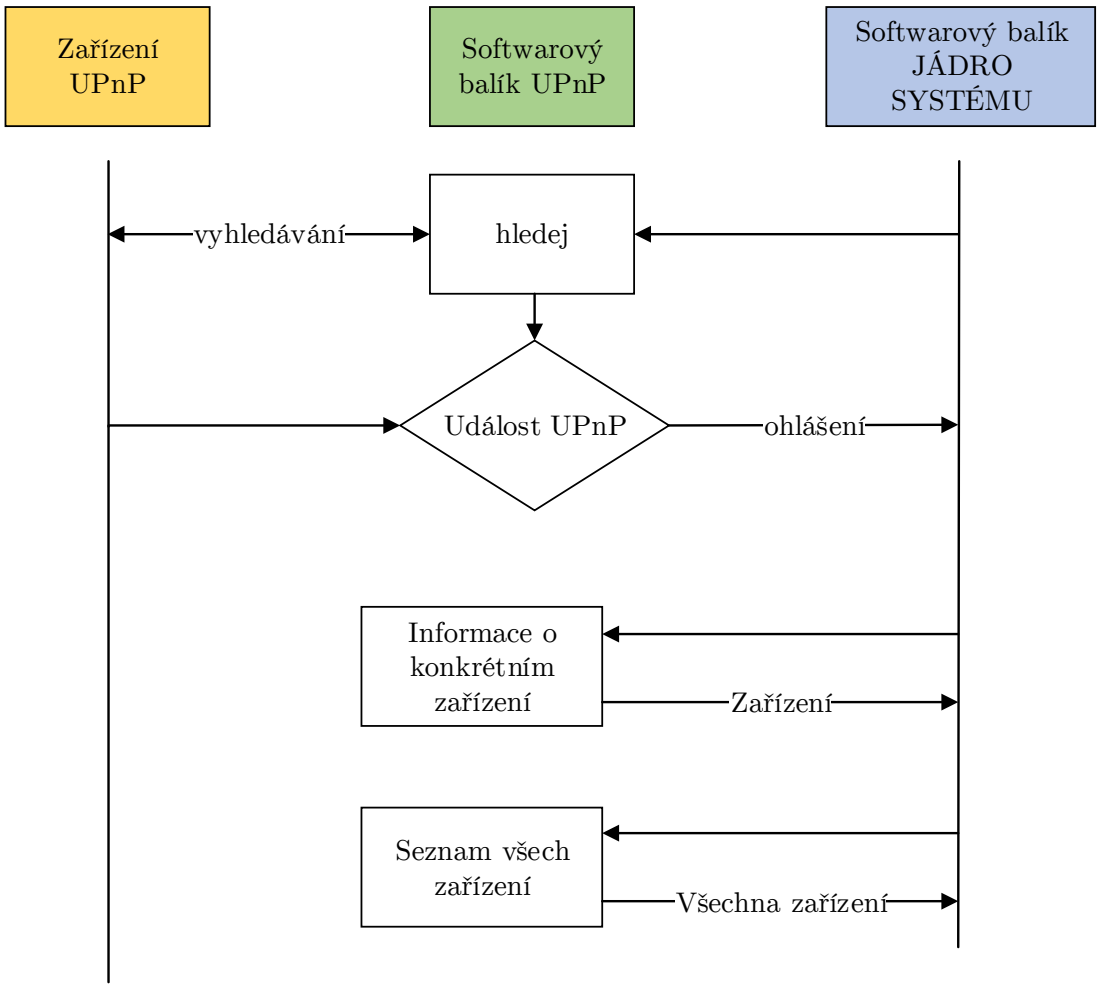
- `localIP` je IP adresa verze 4 lokálního zařízení – IP adresa chytrého domácího serveru,
- `localPort` je lokální port, na kterém aplikace naslouchá SIP komunikaci – standardně se využívá port 5060,
- `transportProtocol` definuje transportní protokol (standardně UDP),
- `proxyIP` vyjadřuje IP adresu SIP Proxy serveru,
- `proxyPort` definuje cílový port SIP Proxy serveru – standardně 5060,
- `registrarIP` je IP adresa registračního serveru SIP použitého k registraci UA,
- `sipUserName` definuje uživatelské jméno SIP (SIP URI),
- `sipPassword` je přístupové heslo k danému SIP účtu.

Registrace probíhá s autentizací dle obrázku 3.23, kdy ověření UA probíhá dle RFC2617 [42]. Registrace se musí periodicky opakovat, nejzazší termín pro opakování je 3600 s.

Po úspěšné registraci se SIP klient nastaví do módu naslouchání a čeká na příchozí SIP zprávy. Port pro naslouchání je definován při registraci v profilu uživatele. Vedle naslouchání je možné odeslat jakoukoliv zprávu SIP prostřednictvím implementované metody `sendMessage`. Při vypnutí zařízení klient odesílá zprávu SIP REGISTER s expirací nastavenou na 0 s, čímž dojde k odregistrování klienta na straně registračního serveru SIP (Registrar server).

### 5.2.2 Softwarový balík UPnP

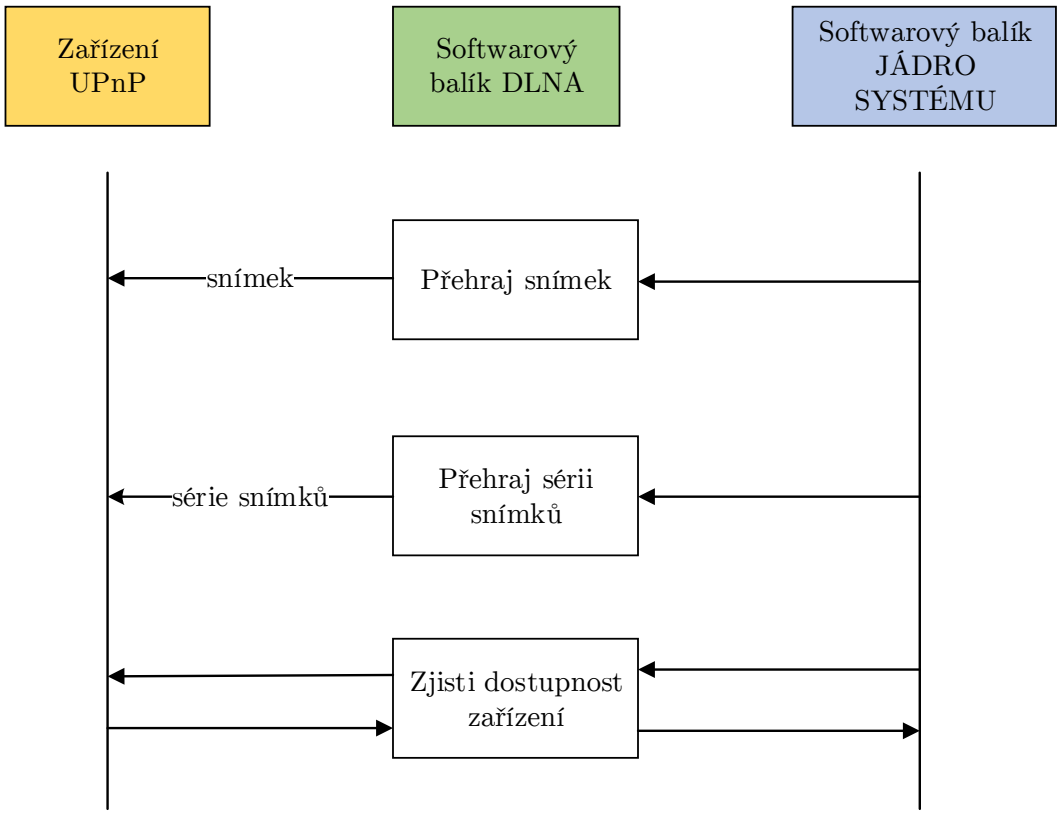
Softwarový balík UPnP umožňuje najít v síti zařízení kompatibilní s UPnP (Universal Plug and Play) definovaném v doporučení RFC 6970 [76]. UPnP zaštiťuje sadu protokolů umožňujících jednoduché vyhledání a připojení externích zařízení v síti. Architektura softwarového balíku v podobě vývojového diagramu je naznačena na obrázku 5.6. Po inicializaci softwarového balíku je okamžitě zahájeno zjišťování ostatních zařízení v síti. Dále je naslouchánо možným událostem (přidání / odebrání zařízení). Tyto sebrané informace okamžitě posílá do jádra systému. Softwarový balík JÁDRO SYSTÉMU si navíc může vyžádat informace o konkrétním zařízení či skupině zařízení prostřednictvím metod `getDevice (String udn)` a `ArrayList <Device> getDeviceList()`, kde `udn` představuje unikátní jméno zařízení. Softwarový balík UPnP je v navrženém řešení využíván zejména pro vyhledávání chytrých televizí, kterým chytrý domácí server může posílat informace o chodu domácnosti.




Obr. 5.6: Architektura softwarového balíku UPnP

### 5.2.3 Softwarový balík DLNA

DLNA (Digital Living Networking Alliance) popisuje technologii odesílání multimediálního obsahu na definované zařízení podporující DLNA a UPnP. Chytrý domácí server mimo hlavní aplikaci provozuje služby, mezi něž patří server DLNA díky němuž lze službu DLNA využít. Softwarový balík DLNA slouží k odesílání jakéhokoliv multimediálního obsahu na lokální DLNA server, který zprávu následně přepošle na konkrétní zařízení. U řešení vyvinutého chytrého domácího serveru je softwarový balík používán k odesílání obrázků z informacemi o chodu domácnosti na televizní obrazovku. Podmínkou je, aby televize byla certifikovaná dle DLNA. Architektura softwarového balíku je popsána vývojovým diagramem na obrázku 5.7.



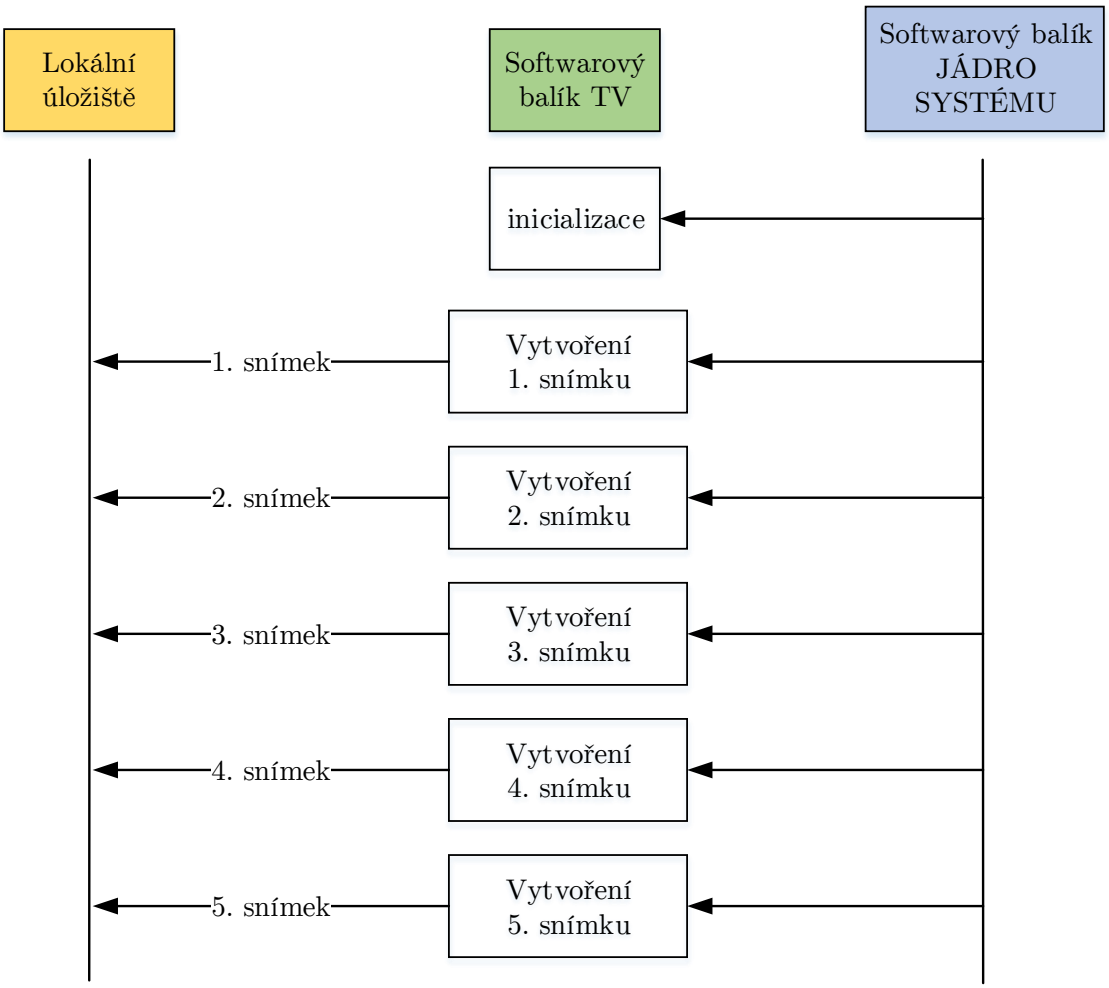

Obr. 5.7: Architektura softwarového balíku DLNA

Softwarový balík DLNA zajišťuje odesílání jednotlivých snímků, odesílání série snímků a ověřuje, zda je zařízení, na které jsou multimediální data odesílána, aktivní. V rámci chytrého domácího serveru tento softwarový balík důležitý k tomu, aby odeslal požadovaný obsah na konkrétní zařízení.

### 5.2.4 Softwarový balík TV

Softwarový balík TV byl vyvinut ke generování obrázků s informacemi nasbíranými z chytrých senzorů, měřičů, apod. Obrázky jsou kódovány do formátu JPEG a dočasně uloženy na lokální úložiště chytrého domácího serveru. Poměr stran každého obrázku je nastaven na 16:9, rozlišení na FullHD (1980x1080 px). Formát JPEG je vybrán z důvodu podpory formátu v zařízeních s certifikací DLNA. V případě potřeby posílat sekvenci snímků je možné využít komprese videosekvence kodérem H.264, který je taktéž certifikován dle DLNA a dle [77] patří ve své kategorii k nejlépe hodnoceným kodekům v poměru kompresní poměr vs. kvalita videosekvence.

K úspoře výpočetního výkonu při generování obrázků jsou v lokální paměti uloženy šablony s pozadím pro každý obrázek. Šablony obsahují nejen pozadí, ale i statický text, symboly apod. Statická data jsou do šablony vložena formou viditelného vodoznaku, pro zabezpečení proti zneužití lze taktéž implementovat nevnímatelné vodoznaky [78]. Vytvořený softwarový balík doplňuje připravené šablony o konkrétní hodnoty získané od sledovaných zařízení v rámci domácí sítě. Základní architekturu a vývojový diagram bundlu naznačuje obrázek 5.8.



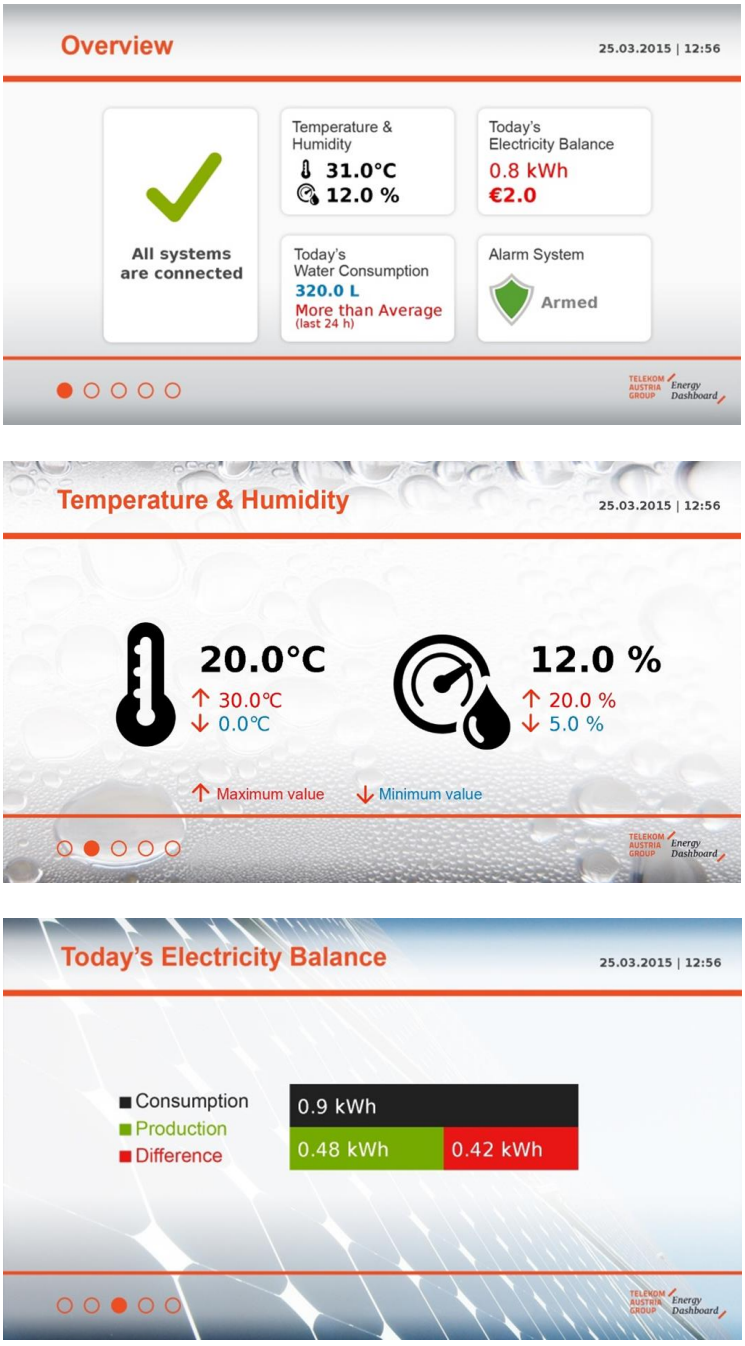

Obr. 5.8: Architektura softwarového balíku TV

Inicializace je opakovaně volána v případě generování nového snímku. Vstupní proměnné při inicializaci jsou:

- `Date` – představuje proměnnou typu String vyjadřující datum a čas generování obrázku ve formátu DD.MM.RRRR | HH:MM,

- `InputFolder` – je proměnná typu String definující cestu k šablonám obrázků na lokálním zařízení,
- `OutputFolder` – je proměnná typu String definující cestu k ukládání vygenerovaných obrázků na lokálním zařízení.

Příklady vytvořených snímků pro televizní obrazovku či jiná zařízení s konkrétními údaji jsou na obrázku 5.9.




Obr. 5.9: Výstupní snímky generované pro TV

### 5.2.5 Softwarový balík SQL

Naměřená data je potřeba ukládat. Otevírají se dvě možnosti, a to ukládání dat na úložiště datového centra operátora, nebo ukládání dat na lokální úložiště chytrého domácího serveru. Obě z uvedených možností mají své výhody i nevýhody.

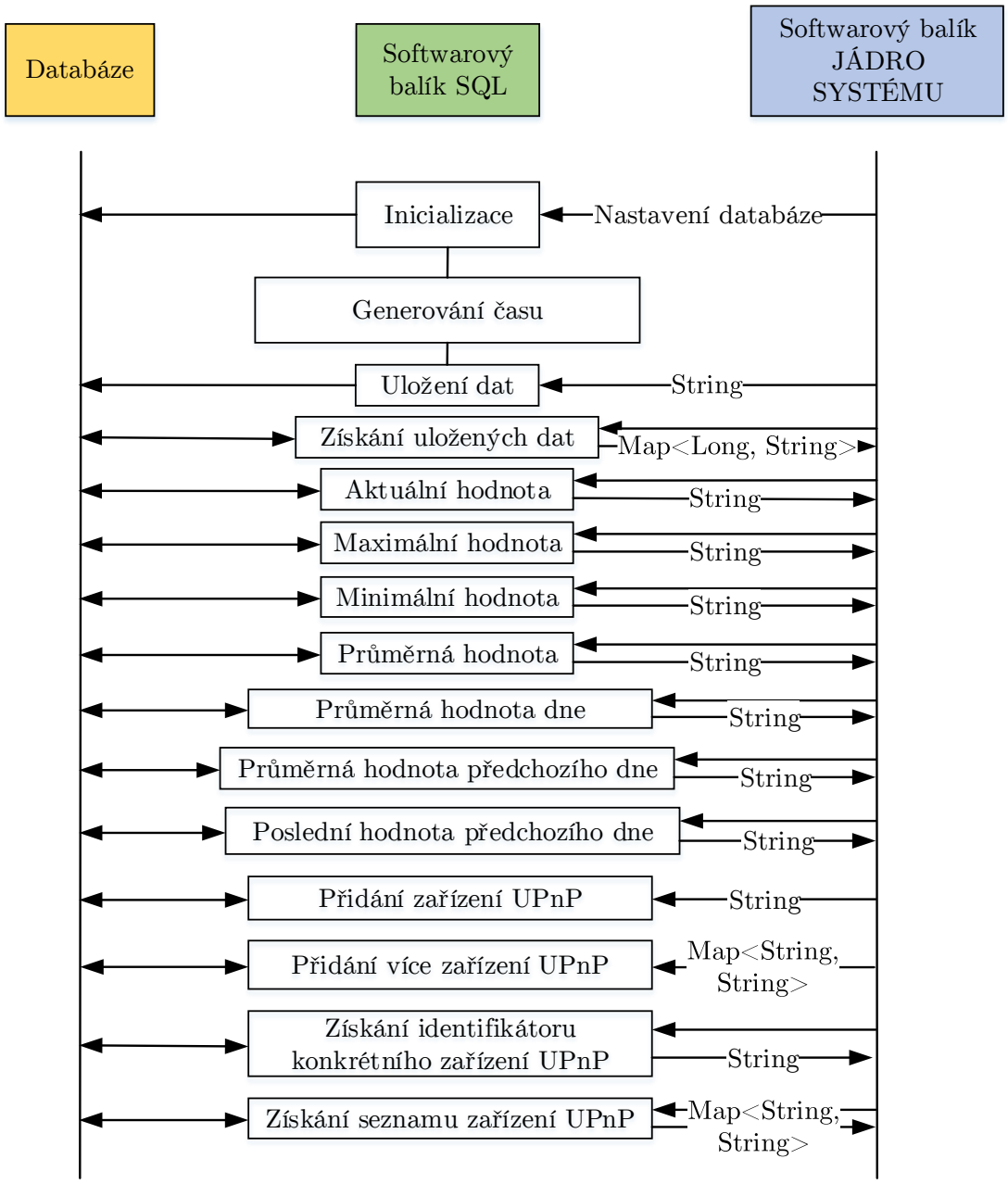
Při použití externí databáze operátora je zapotřebí, aby byl server trvale připojen k Internetu a byl dostupný. V případě nedostupnosti sítě není možné s daty pracovat a tudíž ani informovat uživatelská zařízení v domácnosti. V případě externího přístupu k datům, například prostřednictvím mobilní aplikace či webové stránky má toto řešení výhodu v tom, že data s historií budou dostupná právě i v případě, že domácí chytrý server bude od sítě Internet odpojen.

Použití interní databáze má výhody a nevýhody přesně obrácené. V případě výpadku připojení k Internetu budou všechna data v rámci domácí sítě dostupná, avšak nebudou dostupná prostřednictvím webové služby či mobilní aplikace, která se zpravidla připojuje na službu operátora. Doposud nezmiňovanou výhodu má však řešení interní databáze v tom, že uživatel může mít informace o své domácnosti plně pod kontrolou a neposkytuje je třetí straně.

Právě z důvodu ochrany soukromí byla zvolena metoda lokální databáze v rámci chytrého domácího serveru. O přístupnosti soukromých dat třetí straně rozhoduje uživatel a je mu umožněno, aby data server posílal třetí straně. V tom případě se využívá softwarový balík SIP, který informace prostřednictvím zpráv SIP MESSAGE posílá na uživatelem definovaný SIP účet.

Softwarovou architekturu definuje vývojový diagram na obrázku 5.10. Tabulka 5.1 definuje empiricky získané časové rozlišení uchovávání dat. Levý sloupec „Perioda ukládání“ definuje časový úsek, po který se budou ukládat data v rozlišení stanoveném v pravém sloupci. Prakticky to znamená, že například 2 dny se budou ukládat data do databáze každou minutu. Ostatní data, která nejsou nadále potřebná, se z databáze průběžně mažou. Tímto krokem je zajištěno, že objem dat v databázi nebude narůstat lineárně a dojde k potřebné úspoře kapacity lokálního úložiště.

Při inicializaci softwarového balíku jsou nastaveny cesty k databázi a všem využívaným databázovým tabulkám. Softwarový balík SQL dále umožňuje ukládat / vyčítat konkrétní data do/z databáze. Mimo ukládání / vyčítání konkrétních dat do/z databáze umožňuje softwarový balík manipulovat se záznamy zařízení UPnP, které jsou v databázi taktéž uloženy.




Obr. 5.10: Architektura bundlu SQLite

### 5.2.6 Softwarový balík WMBus

Technologie Wireless M-BUS, jak již bylo zmíněno, je určena k přenosu dat a řízení v oblasti měření a regulace topných systémů, plynu, odběru vody a elektrické energie. Softwarový balík WMBus slouží ke zpracování příchozích dat z těchto zařízení a jejich předání do softwarového balíku JÁDRO SYSTÉMU, který je následně uloží

Tab. 5.1: Uchování dat v databázi

| Perioda ukládání | Rozlišení |
|---|---|
| aktuální hodnota | neukládá se |
| 2 dny | 1 minuta |
| 1 rok | 1 hodina |
| 2 roky a více | 1 den |

do databáze. Při potřebě odesílat data třetí straně je použit popsaný způsob SIP komunikace, kdy ve zprávě SIP MESSAGE jsou získané informace odesílány ve formátu JSON.

## 5.3 Experimentální ověření navržené komunikace

Navržená komunikace chytrého domácího serveru prostřednictvím protokolu SIP byla experimentálně ověřena. Chytrý domácí server byl do sítě zapojen tak, jak ukazuje obrázek 5.3 s tím, že se serverem komunikovaly 2 webové aplikace třetích stran, které shromažďovaly údaje o chodu domácnosti a na základě nich generovaly různé grafy spotřeby. Síťová architektura poskytovatele SIP připojení obsahovala registrační server SIP REGISTRAR a PROXY server. Obě entity byly od sebe oddělené a měly rozdílné IP adresy.

### 5.3.1 Registrace

Při experimentálním testování s reálnými servery byly při registraci chytrého domácího serveru k registračnímu serveru SIP REGISTRAR zjištěny problémy, které lze zobecnit následovně:

1. Požadovaný čas na dobu trvání registrace, pole `expires` ve zprávě SIP REGISTER, registrační server SIP REGISTRAR vždy změnil a definoval vlastní čas expirace.
2. V případě, že má SIP UA přidělenou adresu vnitřní sítě s použitím technologie NAT (Network Address Translation), nelze UA běžným způsobem registrovat.

### Problémy s registrací – změna doby platnosti registrace na registračním serveru SIP REGISTRAR

Analýza problému s registrací proběhla s použitím síťového analyzátoru Wireshark, kterým byla zaznamenána komunikace mezi SIP REGISTRAR serverem a klientem SIP. Na základě experimentálního výzkumu bylo navrženo řešení, které při registraci z finální odpovědi 200 OK získá čas expirace, z něj odečte 2s (dostatečný čas pro režii potřebnou při přenosu zprávy) a výsledek nastaví do pole `expires` při opakované (udržovací) registraci. Pokud tedy SIP REGISTRAR server odešle například `expires` = 300 s, při opakované registraci se automaticky do pole `expires` nastaví 298 s. Je třeba si zapamatovat, že i když ve standardu je dáno, že doba registrace na serveru je udávána v žádosti REGISTER, v praxi tomu tak nemusí být, neboť servery SIP REGISTRAR si můžou upravit čas expirace dle svého nastavení.

### Problémy s registrací – registrace UA z vnitřní sítě za NATem

Problém s registrací zařízení ve vnitřní sítí (za NATem) lze řešit využitím protokolu STUN (Simple Traversal of UDP through NAT) definovaném ve specifikaci RFC 3489 [79]. Server STUN umožňuje klientům vyhledávat jejich veřejnou adresu, typ překladače NAT, a veřejný port přiřazený k překladači NAT s určitým lokálním portem. Po implementaci protokolu STUN a experimentálním testování bylo zjištěno, že při opakovaném posílání zprávy SIP REGISTER v intervalu delším než 30 s zprávy nebyly SIP REGISTRAR serverem akceptovány a registrace byla po dosažení expirační doby ukončena. Důvodem je fakt, že dle RFC 3489 je spojení STUN, tedy svázání vnitřní IP adresy s adresou veřejnou, standardně udržováno právě 30 s [79].

Z uvedených dvou problémů plyne, že v případě klientů používajících NAT není pravda, že doba expirace registrace je jediný parametr, který je potřeba ošetřit. U řešení s protokolem STUN je nutné, aby byla mezi UA a SIP REGISTRAR serverem vyměněna jakákoli zpráva v intervalu do 30 s. V rámci prezentovaného výzkumu byla testována různá softwarová řešení SIP UA a bylo zjištěno, že při aktivaci podpory NAT UA automaticky začaly na SIP REGISTRAR servery pravidelně posílat buď zprávy PUBLISH, nebo REGISTER. Tímto způsobem tedy může být udržováno spojení se zařízeními využívajícími NAT. Ve vyvíjeném řešení bylo implementováno řešení odesílající zprávu SIP REGISTER na SIP REGISTRAR server v intervalu 28 s.

### 5.3.2 Výměna zpráv

Po registraci následovala výměna zpráv mezi chytrým domácím serverem a aplikacemi třetích stran. Zde byl využita navržená struktura JSON společně s kódy OBIS tak, jak byla popsaná v kapitole 3.3.3. Testování probíhalo 30 dnů, kdy server generoval zprávy SIP MESSAGE a odesílal je na předem specifikovaná zařízení. V komunikaci nenastaly během testování žádné problémy. Příklady reálných odesílaných struktur v rámci zpráv SIP MESSAGE a jejich struktury jsou uvedeny v příloze A této práce.

Příklad ukázaný na obrázku A.1 uvádí konkrétní strukturu přenášené zprávy při požadavku externího zařízení o informaci o stavu elektroměru v domácnosti, obrázek A.2 ukazuje odpověď chytrého domácího serveru.

Druhý příklad na obrázcích A.3 a A.4 popisuje strukturu požadavku na nastavení chytrého zařízení řízeného chytrým domácím serverem. Konkrétně se jedná o rozsvícení chytrého světla na její šedesáti procentní výkon.

### 5.3.3 Ostatní komunikace v rámci domácí sítě

Komunikace chytrého domácího serveru v rámci vnitřní sítě probíhala taktéž bez větších komplikací. Součástí testu bylo i posílání obrazových informací na televizní obrazovku prostřednictvím technologií UPnP/DLNA. Zde je třeba podotknout, že ne všechny televize mají správně implementovaný standard DLNA. V případech nestandardní implementace dochází k problémům při přenosech, jenž je možné odchytit pouze metodou reverzního inženýrství. Televize novějšího data, vyrobeny cca od roku 2015, mají většinou vše plně funkční.

# 6 ZÁVĚR

Habilitační práce popisuje základní pohled na systémy IoT (Internet of Things) a analyzuje klíčové aspekty výzkumu a vývoje v této oblasti. Konkrétně je zaměřena na oblast cIoT (Consumer IoT), kde jsou řešeny klíčové aspekty při vývoji chytrého domácího serveru použitelného v sítích telekomunikačních operátorů z hlediska komunikace na aplikační vrstvě modelu TCP/IP. Úvod práce se věnuje popisu variantních komunikačních modelů IoT a jejich použití. Následuje podrobný popis aplikačních protokolů využívaných v cIoT, zejména protokolů MQTT a CoAP. Velký důraz je kladen také na protokol SIP, který byl navržen k signalizaci u služeb internetové telefonie a videokonferencí VoIP (Voice over Internet Protocol). Společně s popisem funkcionalit protokolu SIP je ve třetí kapitole proveden návrh možnosti využití protokolu SIP ke komunikaci chytrého domácího serveru s prvky sítě telekomunikačních operátorů. Mimo vlastní komunikace je zde navržena struktura přenášených zpráv. Závěrem kapitoly jsou diskutovány a vzájemně porovnány protokoly s důrazem na jejich využití v sítích telekomunikačních operátorů.

Druhá část práce se věnuje problematice návrhu chytrého domácího serveru s využitím protokolu SIP. Je zde zvolen vhodný komunikační model pro použití v domácnostech a stručně popsány přenosové technologie, které je možné v rámci chytrých domácností využít. Druhá část také podrobně popisuje hardwarové a softwarové požadavky na realizaci chytrého domácího serveru. V rámci výzkumu byla provedena analýza dostupných frameworků využitelných v cIoT a proběhlo experimentální testování výkonnosti OSGi implementací Eclipse Equinox, Apache Felix a Knopflerfish. Výsledky testů jsou uvedeny v kapitole 4.3.3 habilitační práce. Pro následné experimentální ověření navrženého způsobu komunikace byla v návaznosti na výsledky analýzy zvolena implementace OSGi Knopflerfish.

Závěrečná část práce se věnuje experimentálnímu ověření a testování řešení popsaného v předešlých kapitolách. Je zde popsána jak hardwarová, tak i softwarová struktura vyvinutého chytrého domácího serveru. Důraz byl kladen na experimentální ověření komunikace prostřednictvím protokolu SIP s využitím navržené struktury zpráv popsané v kapitole 3.3.3. Experimentální ověření bylo provedeno v ostrém provozu v síti rakouského telekomunikačního operátora Telekom Austria Group. Poznatky sepsané v habilitační práci jsou výsledkem tříletého výzkumu v oblasti IoT, jehož se autor účastnil. Částečně je taktéž odkazováno na předešlou oblast výzkumu obrazového zpracování, která byla využita zejména při přípravě a distribuci obrazových dat prostřednictvím UPnP/DLNA na televizní obrazovky domácností.

# LITERATURA


[1] MASEK, P.; MASEK, J.; FRANTIK, P.; OMETOV, A.; HOSEK, J.; ANDREEV, S.; MLYNEK, P.; MISUREC, J. A Harmonized Perspective on Transportation Management in Smart Cities: The Novel IoT-Driven Environment for Road Traffic Modeling. *SENSORS*. 2016, roč. 11, č. 1872, s. 1–23. ISSN 1213-1539.

[2] COETZEE, Louis; JOHAN, Eksteen. The Internet of Things - promise for the future? An introduction. In: IEEE (ed.). *IST-Africa Conference Proceedings, 2011*. Gaborone: IEEE, 2011, s. 1–9. ISBN 978-1-905824-26-7.

[3] HERSENT, Olivier; BOSWARTHICK, David; ELLOUMI, Omar. *The internet of things: applications to the smart grid and building automation*. 1. vyd. Hoboken, NJ: Wiley, 2012. ISBN 978-1-119-99435-0.

[4] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020* [online]. Cisco Systems, Inc., [cit. 19. 8. 2016]. Dostupné také z: `http://goo.gl/I9v5Se`.

[5] *Internet of Things Global Standards Initiative* [online]. ITU, [cit. 22. 8. 2016]. Dostupné také z: `http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`.

[6] *Service Offerings: Infosys Digital Smart Home Gateway*. Infosys, [cit. 10. 11. 2016]. Dostupné také z: `https://www.infosys.com/engineering-services/service-offerings/Pages/digital-smart-home-gateway.aspx`.

[7] *Smart home*. DEVELCO PRODUCTS, [cit. 10. 11. 2016]. Dostupné také z: `http://www.develcoproducts.com/business-areas/smart-home/`.

[8] *Smart Home*. Qualcomm, [cit. 10. 11. 2016]. Dostupné také z: `https://www.qualcomm.com/solutions/internet-of-things/smart-home`.

[9] *Smart Home*. Huawei, [cit. 10. 11. 2016]. Dostupné také z: `http://www.huawei.com/minisite/iot/en/smarthome.html`.

[10] The Open Internet. -. 2014, roč. 2014, č. 1, s. 1–9. Dostupné také z: `https://goo.gl/Gf3lxP`.

[11] *OneM2M* [online]. OneM2M, [cit. 26. 9. 2016]. Dostupné také z: `http://www.onem2m.org/`.

[12] *AllSeen Alliance* [online]. AllSeen Alliance, [cit. 26. 9. 2016]. Dostupné také z: `https://allseenalliance.org/`.

[13] *IoTivity* [online]. IoTivity, [cit. 27. 12. 2016]. Dostupné také z: https://www.iotivity.org/.

[14] *Current Members* [online]. OneM2M, [cit. 26. 9. 2016]. Dostupné také z: http://www.onem2m.org/membership/current-members.

[15] *The innovative companies that support AllJoyn®* [online]. Allseen Alliance, [cit. 26. 9. 2016]. Dostupné také z: https://allseenalliance.org/alliance/members.

[16] *The European Commission Rolling plan for ICT Standardisation 2015 section 3.5.6 Internet of Things has a discussion on IoT standards from a competitiveness and policy perspective*. EU: EU, 2015. Dostupné také z: https://ec.europa.eu/digital-agenda/en/rolling-plan-ictstandardisation.

[17] *RFC 7452: Architectural Considerations in Smart Object Networking*. 1. vyd. 2015.

[18] *Bluetooth* [online]. Bluetooth SIG, Inc, [cit. 3. 10. 2016]. Dostupné také z: https://www.bluetooth.com/.

[19] *Z-Wave, the Smartest Choice for your Smart Home* [online]. ZWAVE, [cit. 3. 10. 2016]. Dostupné také z: http://www.z-wave.com/.

[20] *ZigBee* [online]. ZigBee Alliance, [cit. 3. 10. 2016]. Dostupné také z: http://www.zigbee.org/.

[21] *M-BUS* [online]. M-BUS, [cit. 3. 10. 2016]. Dostupné také z: http://www.m-bus.com/.

[22] *ISO/IEC 20922:2016: Information technology —- Message Queuing Telemetry Transport (MQTT) v3.1.1*. 1. vyd. Switzerland: ISO/IEC JTC 1, 2016.

[23] *MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*. IBM, 2010. Dostupné také z: http://www.ibm.com/developerworks/library/ws-mqtt/.

[24] MASEK, Pavel; HOSEK, Jiri; ZEMAN, Krystof; STUSEK, Martin; KOVAC, Dominik; CIKA, Petr; MASEK, Jan; ANDREEV, Sergey; KRÖPFL, Franz. Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience [online]. [cit. 20. 10. 2016]. Dostupné z DOI: 10.1155/2016/8160282.

[25] *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*. 1. vyd. Internet Engineering Task Force, 2008.

[26] *RFC 6455: The WebSocket Protocol*. 1. vyd. Internet Engineering Task Force, 2011.

[27] STANFORD-CLARK, Andy; TRUONG, Hong Linh. *MQTT For Sensor Networks (MQTT-SN): Protocol Specification*. 1.2. vyd. IBM, 2013.

[28] *JavaScript JSON* [online]. W3Schools, [cit. 25. 11. 2016]. Dostupné také z: `http://www.w3schools.com/js/js_json.asp`.

[29] *RFC 7252: The Constrained Application Protocol (CoAP)*. 1. vyd. Universitaet Bremen TZI: Internet Engineering Task Force, 2014.

[30] GLIGORIC, N.; DIMCIC, T.; DRAJIC, D.; KRCO, S.; DEJANOVIC, I.; CHU, N.; OBRADOVIC, A. CoAP over SMS: Performance evaluation for machine to machine communication. In: *Telecommunications Forum (TELFOR), 2012 20th*. 2012, s. 1–4. Dostupné z DOI: `10.1109/TELFOR.2012.6419577`.

[31] COLITTI, W.; STEENHAUT, K.; CARO, N. De; BUTA, B.; DOBROTA, V. Evaluation of constrained application protocol for wireless sensor networks. In: *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. 2011, s. 1–6. ISSN 1944-0367. Dostupné z DOI: `10.1109/LANMAN.2011.6076934`.

[32] *RFC 3261: SIP: Session Initiation Protocol*. East Hanover, NJ: Internet Engineering Task Force, 2012.

[33] CIKA, Petr. *Multimedialni sluzby*. 1. vyd. Brno: Vysoke uceni technicke v Brne, 2012. ISBN 978-80-214-4443-0.

[34] *RFC 4566: SDP: Session Describtion Protocol*. University of Glasgow: Internet Engineering Task Force, 2006. Technická zpráva.

[35] *RFC 6086: Session Initiation Protocol (SIP) INFO Method and Package Framework*. Georgetown University: Internet Engineering Task Force, 2011.

[36] *RFC 3262: Reliability of Provisional Responses in the Session Initiation Protocol (SIP)*. Columbia University: Internet Engineering Task Force, 2002.

[37] *RFC 3265: Session Initiation Protocol (SIP)-Specific Event Notification*. Plano, TX: Internet Engineering Task Force, 2002.

[38] *RFC 3311: The Session Initiation Protocol (SIP) UPDATE Method*. East Hanover, NJ: Internet Engineering Task Force, 2002.

[39] *RFC 3428: Session Initiation Protocol (SIP) Extension for Instant Messaging*. Plano, TX: Internet Engineering Task Force, 2002.

[40] *RFC 3515: The Session Initiation Protocol (SIP) Refer Method*. Plano, TX: Internet Engineering Task Force, 2003.

[41] *RFC 3903: Session Initiation Protocol (SIP) Extension for Event State Publication*. Finland: Internet Engineering Task Force, 2004.

[42] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. 1. vyd. Internet Engineering Task Force, 1999.

[43] *RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format*. 1. vyd. Internet Engineering Task Force, 2014.

[44] *List of Standard OBIS Codes and COSEM Objects*. DLMS User Association, [cit. 10. 1. 2017]. Dostupné také z: `https://www.dlms.com/documentation/listofstandardobiscodesandmaintenanceproces/index.html`.

[45] *Welcome* [online]. HGi, [cit. 5. 1. 2017]. Dostupné také z: `http://www.homegatewayinitiative.org`.

[46] HOSEK, Jiri; MASEK, Pavel; KOVAC, Dominik; KRÖPFL, Franz. M2M gateway: The centerpiece of future home. In: *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. 2014, s. 190–197. ISSN 2157-0221. Dostupné z DOI: `10.1109/ICUMT.2014.7002101`.

[47] *SigFox Vs. LoRa: A Comparison Between Technologies & Business Models* [online]. New York: Link Labs, [cit. 10. 12. 2016]. Dostupné také z: `http://www.link-labs.com/blog/sigfox-vs-lora`.

[48] GALININA, Olga; MIKHAYLOV, Konstantin; ANDREEV, Sergey; TURLIKOV, Andrey. Wireless Sensor Network Based Smart Home System over BLE with Energy Harvesting Capability. Dostupné z DOI: `10.1007/978-3-319-10353-2_37`.

[49] *11 Internet of Things (IoT) Protocols You Need to Know About* [online]. Design Spark, [cit. 11. 12. 2016]. Dostupné také z: `https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about`.

[50] *The ZigBee Vs WiFi Battle For M2M Communication* [online]. LinkLabs, [cit. 8. 1. 2017]. Dostupné také z: `https://www.link-labs.com/blog/zigbee-vs-wifi-802-11ah`.

[51] ZHANG, Qiang; SUN, Yugeng; CUI, Zhenhui. Application and analysis of ZigBee technology for Smart Grid. In: 2010 International Conference on Computer a Information Application, s. 171–174. ISBN 978-1-4244-8598-7. Dostupné z DOI: `10.1109/ICCIA.2010.6141563`.

[52] *IEEE 802.15 WPAN™ Task Group 4 (TG4)* [online]. IEEE 802.15, [cit. 9. 1. 2017]. Dostupné také z: `http://www.ieee802.org/15/pub/TG4.html`.

[53] SPINSANTE, Susanna; PIZZICHINI, Mirco; MENCARELLI, Matteo; SQUARTINI, Stefano; GAMBI, Ennio. Evaluation of the Wireless M-Bus standard for future smart water grids. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. Sardinia: -, 2013, s. 1382–1387. Dostupné z DOI: 10.1109/IWCMC.2013.6583758.

[54] FLAMMINI, Alessandra; RINALDI, Stefano; VEZZOLI, Angelo. The sense of time in Open Metering System. Dostupné z DOI: 10.1109/SMFG.2011.6125767.

[55] ZUKAL, Martin; CIKA, Petr. VoIP softphone v programovacim jazyce Java. *Elektrorevue*. 2010, roč. 12, č. 5, s. 1–6. ISSN 1213-1539.

[56] *OSGi Alliance*. OSGi Alliance, [cit. 30. 12. 2016]. Dostupné také z: https://www.osgi.org/.

[57] LIN, R. T.; HSU, Chin-Shun; CHUN, Tee Yuen; CHENG, Sheng-Tzong. OSGi-Based Smart Home Architecture for heterogeneous network. In: *2008 3rd International Conference on Sensing Technology*. 2008, s. 527–532. ISSN 2156-8065. Dostupné z DOI: 10.1109/ICSENST.2008.4757162.

[58] *Everything can be a bundle - making OSGi bundles of Java legacy code* [online]. Slideshare, [cit. 27. 12. 2016]. Dostupné také z: http://www.slideshare.net/mfrancis/everything-can-be-a-bundle-making-osgi-bundles-of-java-legacy-code-gunnar-ekolin-makewave.

[59] *OSGi Runtime Comparison* [online]. A Day At The Races, [cit. 29. 12. 2016]. Dostupné také z: http://bit.ly/1JBMw7k.

[60] *Enterprise OSGi frameworks: Maturity comparison Apache Aries vs. Eclipse Gemini*. Stack Exchange Inc, [cit. 29. 12. 2016]. Dostupné také z: http://bit.ly/1KC9sh2.

[61] *OSGi core framework*. Eclipse Equinox, [cit. 30. 12. 2016]. Dostupné také z: http://www.eclipse.org/equinox/.

[62] *Open Source OSGi SDK*. Knopflerfish, [cit. 30. 12. 2016]. Dostupné také z: http://www.knopflerfish.org/.

[63] *OSGi Framework and Service Platform*. Apache Felix, [cit. 29. 12. 2016]. Dostupné také z: http://felix.apache.org/.

[64] HOSEK, Jiri; MASEK, Pavel; KOVAC, Dominik; RIES, Michal; KRÖPFL, Franz. IP home gateway as universal multi-purpose enabler for smart home services. *e & i Elektrotechnik und Informationstechnik*. 2014, roč. 131, č. 4, s. 123–128. ISSN 1613-7620. Dostupné z DOI: 10.1007/s00502-014-0209-x.

[65] *JBoss Work on Internet of Things*. JBoss, [cit. 30. 12. 2016]. Dostupné také z: http://www.jboss.org/iot/.

[66] *Raspberry Pi: Model B*. Raspberry Pi Foundation, [cit. 5. 1. 2017]. Dostupné také z: http://www.raspberrypi.org/products/model-b/.

[67] *SPIRENT AVALANCHE - AVALANCHE 3100B*. USA, 2011. Dostupné také z: https://www.info-point-security.com/sites/default/files/spirent_avalanche_3100_datasheet.pdf.

[68] STUSEK, Martin; HOSEK, Jiri; KOVAC, Dominik; MASEK, Pavel; CIKA, Petr; MASEK, Jan; KRÖPFL, Franz. Performance Analysis of the OSGi-based IoT Frameworks on Restricted Devices as Enablers for Connected-Home. In: *2015 7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. Brno: ICUMT, 2015, s. 211–216. ISBN 978-1-4673-9282-2.

[69] *ASM*. OW2 Consortium, [cit. 30. 12. 2016]. Dostupné také z: http://asm.ow2.org/.

[70] *Java: What are the system requirements for Java?* USA: Oracle, [cit. 4. 1. 2017]. Dostupné také z: https://www.java.com/en/download/help/sysreq.xml.

[71] MONCHIERO, Matteo. RISC Microprocessors Examples: MIPS & ARM. In: Milano: Politecnico di Milano.

[72] *ARM architecture*. San Francisco (CA): Wikimedia Foundation, [cit. 5. 1. 2017]. Dostupné také z: https://en.wikipedia.org/wiki/ARM_architecture.

[73] *AVM: FRITZ!Box Fon WLAN 7390*. AVM, [cit. 30. 12. 2016]. Dostupné také z: http://en.avm.de/products/fritzbox/fritzbox-7390/.

[74] *OpenWRT Wireless Freedom*. USA, [cit. 5. 1. 2017]. Dostupné také z: https://openwrt.org/.

[75] STUSEK, Martin; MASEK, Pavel; ZEMAN, Krystof; KOVAC, Dominik; CIKA, Petr; POKORNY, Jiri; KRÖPFL, Franz. A Novel Application of CWMP: An Operator-grade Management Platform for IoT. *International Journal of Advances in Telecommunications Electronics, Signals and Systems*. 2016, roč. 5, č. 3, s. 171–177. Dostupné z DOI: 10.11601/ijates.v5i3.224.

[76] *RFC 6970: Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF)*. 1. vyd. Internet Engineering Task Force, 2013.

[77] CIKA, Petr; KOVAC, Dominik; BILEK, Jan. Objective video quality assessment methods: Video encoders comparison. In: *7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*. Brno: IEEE Computer Society, 2016, s. 335–338. ISBN 978-146739283-9. ISSN 21570221. Dostupné z DOI: 10.1109/ICUMT.2015.7382453.

[78] CIKA, Petr. Survey of frequency domain image watermarking techniques. *International Journal of Circuits, Systems and Signal Processing*. 2015, roč. 2015, č. 9. ISSN 19984464.

[79] *RFC 3489: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. 1. vyd. The Internet Engineering Task Force, 2003.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

| | |
|---|---|
| AMQP | Advanced Message Queuing Protocol |
| API | Application Programming Interface |
| AoR | Address of Record |
| ARM | Advanced RISC Machine |
| BLE | Bluetooth Low Energy |
| B2BUA | Back-to-Back User Agent |
| CoAP | Constrained Application Protocol |
| cIoT | Consumer IoT |
| CPEG | Core Platform Expert Group |
| DCPS | Data Centric Publish Subscribe |
| DDoS | Distributed Denial of Service |
| DDS | Data Distribution Service |
| DLNA | Digital Living Networking Alliance |
| DLRL | Data Local Reconstruction Layer |
| D2C | Device-To-Cloud |
| D2D | Device-To-Device |
| D2G | Device-To-Gateway |
| ETSI | European Telecommunications Standards Institute |
| FTP | File Transfer Protocol |
| HGI | Home Gateway Initiative |
| HTTP | Hypertext Transfer Protocol |
| H.323 | rodina protokolů pro audio/video komunikaci |
| IETF | Internet Engineering Task Force |

| | |
|---|---|
| iIoT | Industrial IoT |
| IM | Instant Messaging |
| IMS | IP Multimedia Subsystem |
| IoT | Internet of Things |
| IoE | Internet of Everything |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| MIPS | Microprocessor without Interlocked Pipeline Stages |
| MQTT | Message Queue Telemetry Transport |
| M2M | Machine-to-Machine |
| NAT | Network Address Translation |
| OBIS | Object Identification System |
| OMG | Object Management Group |
| OSGi | Open Service Gateway initiative |
| PLC | Power Line Communication |
| POJO | Plain Old Java Objects |
| QoE | Quality of Experience |
| QoS | Quality of Service |
| REST | REpresentational State Transfer |
| SDP | Session Description Protocol |
| SIP | Session Initiation Protocol |
| STUN | Simple Traversal of UDP through NAT |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |

| | |
|---|---|
| UA | User Agent |
| UAC | User Agent Client |
| UAS | User Agent Server |
| UDP | User Datagram Protocol |
| UPnP | Universal Plug and Play |
| URI | Uniform Resource Identifier |
| VoIP | Voice over Internet Protocol |
| XML | Extensible Markup Language |
| XMPP | Extensible Messaging and Presence Protocol |

# SEZNAM PŘÍLOH

# A TESTOVÁNÍ KOMUNIKACE SE STRUKTUROU JSON

## A.1 Získání dat z chytrého domácího serveru


```
{
  "system": {
    "data": [
      {
        "method": "get",
        "name": "Current Time",
        "value": null
      }
    ]
  },
  "device": {
    "type": "Smart Meter",
    "objectCodeVersion": "ObisV2.9",
    "id": [
      {
        "name": "Serial Number",
        "value": "0131465200041",
        "objectCode": "00:96.1.0"
      }
    ],
    "timestamp": "20151015T15:32:56+01:00",
    "data": [
      {
        "method": "get",
        "name": "Meter reading total (A+)",
        "value": null,
        "units": null,
        "objectCode": "10:1.8.0"
      }
    ]
  }
}
```


Obr. A.1: Struktura JSON pro požadavek na získání informace o stavu elektroměru

```
{
  "system": {
    "type": "SHGW",
    "location": {
      "info": "Main building, 5th floor",
      "format": "DD",
      "latitude": "47.29716073357847",
      "longitude": "16.217365264892578"
    },
    "data": [
      {
        "method": "response",
        "responseResult": "ok",
        "name": "Current Time",
        "value": "20151015T15:32:56+01:00"
      }
    ]
  },
  "device": {
    "type": "Smart Meter",
    "objectCodeVersion": "ObisV2.9",
    "id": [
      {
        "name": "Serial Number",
        "value": "0131465200041",
        "objectCode": "00:96.1.0"
      }
    ],
    "timestamp": "20151015T15:32:56+01:00",
    "data": [
      {
        "method": "response",
        "responseResult": "ok",
        "name": "Meter reading total (A+)",
        "value": "1246.5",
        "units": "kWh",
        "objectCode": "10:1.8.0"
      }
    ]
  }
}
```


Obr. A.2: Struktura JSON pro odpověď s informací o stavu elektroměru

## A.2 Struktura JSON pro nastavení chytrého zařízení z chytrého domácího serveru – osvětlení


```
{
  "device": {
    "type": "Hue Bulbs",
    "id": [
      {
        "name": "Serial Number",
        "value": "450012"
      }
    ],
    "timestamp": "20151015T15:32:56+01:00",
    "data": [
      {
        "method": "set",
        "name": "Switch",
        "value": "1",
        "units": null
      },
      {
        "method": "set",
        "name": "Brightness",
        "value": "60",
        "units": "%"
      }
    ]
  }
}
```


Obr. A.3: Požadavek na nastavení chytrého osvětlení

```
{
  "device": {
    "type": "Hue Bulbs",
    "id": [
      {
        "name": "Serial Number",
        "value": "450012"
      }
    ],
    "timestamp": "20151015T15:32:56+01:00",
    "data": [
      {
        "method": "response",
        "responseResult": "ok",
        "name": "Switch",
        "value": "on",
        "units": null
      },
      {
        "method": "response",
        "responseResult": "ok",
        "name": "Brightness",
        "value": "60",
        "units": "%"
      }
    ]
  }
}
```


Obr. A.4: Struktura JSON pro odpověď obsahující potvrzení a provedení přijatého požadavku