ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Generická IoT síť

Generic IoT network

Bakalářská práce

Studijní program: Softwarové inženýrství a technologie

Vedoucí bakalářské práce: Ing. Jan Kubr

Jan Zubr

Praha 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zubr** Jméno: **Jan** Osobní číslo: **420391**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Generická IoT síť

Název bakalářské práce anglicky:

Generic IoT Network

Pokyny pro vypracování:

Analyzujte možnosti komunikace v IoT sítích.
Navrhněte a implementujte komunikační model v IoT síti. Při návrhu a realizaci se zaměřte na efektivitu komunikace z pohledu přenosu užitečných dat, na zabezpečení komunikace proti odposlechu a změně dat, na minimalizaci SPOF (Single Point of Failure) a na možnost itegrace s jinými systémy.
Navrhněte a proveďte testy vašeho řešení.

Seznam doporučené literatury:

Coulouris G., Dollimore J., Kindberg T., Blair G.: Distributed Systems: Concepts and Design (5th Edition), Addison-Wesley 2011, ISBN: 0132143011.
Attiya H., Welch J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics, John Wiley and Sons, Inc., 2004. Second Edition, ISBN: 978-0-471-45324-6.
Craig Walls.: Spring Boot in Action. Manning Publications, 2016. ISBN: 978-1617292545

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Kubr, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jan Kubr, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

14. 5. 2019
Datum převzetí zadání

Podpis studenta

PODĚKOVÁNÍ

Děkuji Ing. Janu Kubrovi za vedení bakalářské práce a cenné podněty a připomínky, které v rámci vedení poskytoval. Dále děkuji i rodině za její podporu v průběhu psaní práce.

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci s názvem „*Generická IoT síť*“ vypracoval samostatně a použil k tomu odbornou literaturu a prameny, které uvádím v seznamu přiloženém k bakalářské práci.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V *Praze* dne …… …….

Podpis

Název bakalářské práce: Generická IoT síť

Abstrakt:

Tato práce má za cíl navrhnout generickou IoT síť, která disponuje vysokou dostupností, bezpečností a škálovatelností. Zároveň je kladen důraz na efektivitu celého systému a možnost systém rozšiřovat o další funkce a integrovat jej s jinými systémy. Práce je rozdělena do pěti hlavních kapitol. První kapitola se zabývá obecným úvodem do tématu Internet věcí a definuje cíle práce. V druhé kapitole je provedena analýza existujících možností pro návrh víceuzlových systémů. Třetí kapitola navazuje na výsledky analýzy z druhé kapitoly a představuje návrh řešení pro generickou IoT síť. Čtvrtá kapitola pojednává o implementaci návrhu a představuje klíčové části implementace. V rámci páté kapitoly je implementace otestována a výsledky jsou vyhodnoceny.

Klíčová slova:

distribuované systémy, Internet věcí, redundance, cluster

Bachelor´s Thesis title: Generic IoT network

Abstract:

The goal of this thesis to design a generic IoT network that has high availability, security and scalability. Main focus of design is on the efficiency of the entire system and the ability to extend the system with other functions and integrate it with other systems. The thesis is divided into five main chapters. The first introductory chapter deals with the general introduction to the Internet of Things and defines the goals of the work. In the second chapter there is an analysis of existing possibilities for design of multi-node systems. The third chapter builds on the results of the analysis of the second chapter and presents a proposed design for a generic IoT network. The fourth chapter discusses the implementation of the proposed design and presents key parts of the implementation. In the fifth chapter, the implementation is tested and the results are evaluated.

Key words:

distributed systems, Internet of Things, redundancy, cluster

Obsah

1 Úvod

Počátky Internetu lze datovat už do 60. let 20. století, ale forma, ve které jej známe dnes, začala vznikat spojováním separovaných sítí a utvářením standardů v 80. letech téhož století. Internet přinesl do oblasti počítačů a informační techniky řadu nových možností a výzev, se kterými bylo potřeba se vypořádat. Tyto skutečnosti umožnily lidem navzájem komunikovat a vytvářet systémy, které jsou přes internet provázány a taktéž spolu komunikují.

V posledních letech zaznamenáváme významný nárůst tzv. chytrých zařízení, která se k Internetu připojují a vyměňují si jeho prostřednictvím data. Příkladem mohou být garážová vrata ovládaná přes internetový prohlížeč. Celkovým trendem vývoje je vytvořit z těchto chytrých zařízení provázaný ekosystém, označovaný jako „Internet věcí“, též známý pod anglickým pojmem Internet-of-Things (IoT).

V tomto ekosystému si zařízení připojená k Internetu navzájem vyměňují data a podle nich reagují, přizpůsobují se, učí se a vzájemně se integrují. Jako příklad lze uvést dům, který uzavře okna a uzamkne dveře, pokud se jeho obyvatelé nenacházejí v blízkosti. Zde se logicky musí jednat o souhru několika akcí, které jsou vykonány různými členy systému na základě nějaké události.

Způsob výměny informací včetně bezpečnosti přenosu mezi jednotlivými členy obecného IoT systému představuje novou výzvu, se kterou je potřeba se vypořádat. Existuje velmi mnoho možností, jakými lze k řešení této výzvy přistoupit. Vzhledem k rozmanitosti zařízení, která mohou v IoT sítích komunikovat, nelze zvolit jeden, ten „nejlepší“ způsob komunikace. Je třeba brát ohledy na výkon zařízení, energetickou spotřebu, konektivitu a celkové finanční náklady na jeho provoz. V rámci této práce se pokusím analyzovat aktuální možnosti pro komunikaci v IoT sítích a navrhnout optimální řešení pro lokální IoT síť v rámci jedné oblasti (např. budovy, bytu, zahrady).

1.1 Cíle práce

Cílem této práce je navrhnout univerzální komunikační model pro lokální IoT síť. Důraz bude kladen hlavně na efektivitu přenosu dat v síti tak, aby užitečná data byla co nejmenší a zabírala co největší podíl v rámci přenášeného bloku. Dále se v návrhu modelu zaměřím na zabezpečení této IoT sítě, aby nehrozil únik dat či manipulace se sítí. Důležitým cílem návrhu je eliminace jediného bodu selhání (Single-Point-Of-Failure, SPOF), aby síť zůstala v provozu

i při vyřazení několika klíčových prvků. To položí základ pro vznik distribuovaného systému využívajícího principy vysoké dostupnosti se značnou mírou rozšiřitelnosti a přizpůsobitelnosti.

V rámci práce bude provedena analýza existujících technologií a vhodné technologie budou při návrhu využity. Tento krok umožní snadnou čitelnost a údržbu systému a také jeho integraci s již existujícími systémy a službami. Lze uvažovat o integraci například se službami If-This-Then-That (IFTTT), Google Home a podobně.

Nakonec bude celý implementovaný systém otestován v praktickém nasazení a výsledky testů budou analyzovány s ohledem na výše uvedené cíle.

2 Analýza současného stavu

V současnosti se pro realizaci IoT systémů využívá velké množství různých přístupů. Každý přístup lze pojmout z několika pohledů. Z pohledu fyzické architektury jsou všechny systémy velmi podobné. Vždy se jedná o soubor fyzických zařízení s internetovou konektivitou na různých místech. Fyzické provedení konektivity není rozhodující. Další pohledy zahrnují způsob výměny dat a logický model.

Výměnu dat mezi jednotlivými členy – uzly IoT systému lze řešit širokým spektrem protokolů. Některé protokoly jsou vhodnější pro výměnu dat mezi výkonnějšími uzly, jako jsou průmyslové počítače založené na architektuře x86 nebo ARM, na kterých je nasazen standardní plnohodnotný operační systém. Tyto protokoly dokáží přenášet velmi komplexní data, například XML soubory. Jiné protokoly jsou vhodné pro mikroprocesorové uzly, které nedokáží zpracovat velké množství dat v reálném čase a místo operačního systému mají specializovaný firmware. Tyto protokoly jsou optimalizovány pro přenos malých binárních dat.

Logický model IoT systému představuje základní členění celého systému. Při návrhu modelu musíme brát v potaz následující faktory: umístění částí – uzlů systému, umístění dat v systému, funkční role jednotlivých částí – uzlů systému, komunikační vzory a samotnou softwarovou architekturu. Důležitými charakteristikami modelu jsou možnosti interakce mezi uzly, odolnost proti selhání a bezpečnost celého systému.

2.1 Logické modely pro IoT systémy

2.1.1 Client – Server

V modelu Client – Server vždy komunikuje klient se serverem pomocí počítačové sítě. Server je většinou specializovaná aplikace, která je naprogramována ke zpracování konkrétních požadavků klienta. Komunikaci vždy inicializuje klient a server na požadavky klienta odpovídá. Sám po ukončení spojení s klientem komunikaci nenavazuje. Příkladem takového modelu může být emailová služba nebo webová stránka. [1]

2.1.2 Master – Slave

Model Master – Slave je ve své podstatě logickým opakem modelu Client – Server. Uzel, který je v roli Master, zahajuje, řídí a ukončuje veškerou komunikaci se svými podřízenými uzly, které jsou v roli Slave. Tohoto modelu využívá například databázové úložiště, kdy jedna instance databáze je označena jako Master a ostatní instance jako Slave. Master instance do Slave instancí replikuje změny. [2]

2.1.3 Publish – Subscribe

V tomto modelu existuje prostředník, nebo skupina prostředníků – brokeři, se kterými uzly komunikují. Uzly se u brokerů registrují do komunikačních kanálů – témat a určují, zda dané téma chtějí odebírat, či do něj publikovat. Publikující uzel odesílá brokerovi zprávu s identifikací tématu. Broker tuto zprávu distribuuje všem uzlům, které dané téma odebírají. Tento model sdílí množství charakteristik s modelem Client – Server. Nicméně broker, který vykonává roli serveru, je univerzální a jedna instance může být použita i pro více různých aplikací. Tento model využívají různé middleware aplikace, například Apache Kafka. [1]

2.1.4 Peer – to – Peer

Model Peer – to – Peer (v překladu: rovný s rovným) je model, u kterého nelze určit jeden centralizovaný prvek tak, jak to bylo možné u všech předcházejících modelů. Zde může přímo komunikovat každý uzel s každým a fungovat jako klient a server zároveň. Velkou výhodou tohoto modelu je to, že je odolný proti výpadkům jednotlivých uzlů. Na druhou stranu je implementace takového systému náročná. Na principu Peer – to – Peer je založena světoznámá síť pro sdílení souborů BitTorrent. [1]

2.1.5 Cloud

Model cloud je založen hlavně na internetových službách třetích stran. Většinou se zde využívá přístupu Platform as a Service – Platforma jako služba (PaaS), kdy dochází k pronájmu již hotové platformy pro IoT. K této platformě se poté pomocí poskytovatelem nadefinovaného postupu připojí jednotlivé uzly. V samotné platformě se nadefinují jednotlivá pravidla pro distribuci a zpracování dat. O zbytek se již stará samotná služba. Velkou výhodou tohoto přístupu je relativně nízká náročnost konfigurace a implementace systému. Selhání na straně poskytovatele je téměř vyloučeno z důvodu jeho rozsáhlé infrastruktury. Mezi nevýhody patří zejména nutnost neustálé a dobré konektivity k Internetu, což může v řadě případů vytvořit

úzké hrdlo a jediný bod selhání. Tím může dojít k vyřazení celého systému z provozu. Příklady takovýchto služeb jsou například Amazon AWS (https:// aws.amazon.com), Microsoft Azure (https://azure.microsoft.com) nebo IBM Bluemix (https://console.bluemix.net/). [1]

2.2 Komunikační protokoly pro IoT systémy

2.2.1 Message Queuing Telemetry Transport – MQTT

MQTT je transportní protokol pro výměnu zpráv využívající Publish – Subscribe model. Jde o nenáročný, otevřený, velice jednoduchý protokol, který je navržen pro snadnou implementaci. Tyto vlastnosti jej činí ideálním pro využití ve spoustě situací, včetně použití pro M2M (Machine – to – Machine) nebo IoT aplikace, kde je výpočetní výkon zařízení omezen a objem přenesených dat limitován.

Protokol funguje přes Transmission Control Protocol (TCP) nebo přes další síťové protokoly, které poskytují seřazený, bezztrátový a obousměrný přenos dat. Jeho výhodou je, že nevyžaduje konkrétní formát přenášených dat. Lze tedy jeho prostřednictvím přenášet prostý text, JavaScript Object Notation (JSON) a XML. Dále protokol umožňuje tři úrovně kvality doručení zpráv: nejvýše jednou, právě jednou, alespoň jednou. [3]

2.2.2 Constrained Application Protocol – CoAP

CoAP je specializovaný přenosový webový protokol pro použití na uzlech s malým výpočetním výkonem a v sítích s omezeným datovým tokem. Uzly mohou být osmibitové mikroprocesory s malým množstvím RAM a ROM, zatímco sítě mohou mít vysokou chybovost a maximální datový tok v desítkám kbit/s. Protokol je navržen pro M2M aplikace, například pro chytré budovy.

CoAP využívá mezi uzly model přístupu požadavek – odpověď, podporuje automatickou detekci služeb a zdrojů a zahrnuje klíčové koncepty webu, jako jsou URI a typy internetového média. CoAP lze velmi jednoduše propojit s Hypertext Transfer Protocol (HTTP) pro integraci s webovými službami. Přitom CoAP zachovává speciální požadavky, jako je podpora multicast, velmi nízkou režii a jednoduchost. [4]

2.2.3 Extensible Messaging and Presence Protocol – XMPP

Extensible Messaging and Presence Protocol (XMPP) je protokol pro výměnu zpráv a zjištění stavu. Využívá formátu XML a komunikačního modelu Client – Server. Protokol umožňuje výměnu strukturovaných a rozšiřitelných dat v téměř reálném čase mezi dvěma a více uzly. Původně byl navržen pro instantní Messenger Jabber, později se začal využívat i na vzájemnou komunikaci mezi aplikacemi. V roce 2004 byl Internet Engineering Task Force (IETF) přijat jako standard RFC-6120. [5]

2.2.4 Advanced Message Queuing Protocol – AMQP

Advanced Message Queuing Protocol (AMQP) je otevřený internetový protokol pro výměnu business zpráv a pracuje v aplikační vrstvě. Byl vytvořen hlavně pro bankovní sektor a na jeho vzniku se podílely Bank of America, JPMorgan, Credit Suisse Barclays, Deutsche Börse Systems a Goldman Sachs. Implementace tohoto protokolu poskytují RedHat, Microsoft, VM Ware, Cisco a Inetco. Hlavními vlastnostmi AMPQ je řazení zpráv, spolehlivost, podpora směrování (podporuje Point – to – Point i Publish – Subscribe model) a zabezpečené komunikace. Výhodou protokolu je, že nevyžaduje konkrétní formu přenášených dat. Lze tak přenášet prostý text, JSON i XML. [6]

2.2.5 Representational State Transfer – REST

Representational State Transfer (REST) není sám o sobě protokol jako výše uvedené příklady, ale spíše architektonický styl. Proto nemá vlastní standard, ale za dobu jeho existence se de facto stal komunikačním standardem. Využívá řady Request for comment (RFC) dokumentů týkajících se HTTP protokolu. Kompletní seznam RFC dokumentů, na kterých REST staví, lze nalézt na http://standards.rest. Jedná se v podstatě o nástavbu nad HTTP protokolem. Autorem je Roy Fielding, který jej definoval v roce 2000 ve své dizertační práci. REST přenáší data jako HTTP dotazy, takže stejně jako HTTP protokol je bezstavový. Také nevyžaduje konkrétní formát přenášených dat a lze tak přenášet prostý text, JSON nebo XML.

2.2.6 Websocket – WS

Websocket protokol umožňuje obousměrnou komunikaci mezi klientem a serverem v rámci jednoho TCP spojení. Tento protokol vznikl jako náhrada a alternativa za HTTP polling ze strany klienta, kterým se obousměrná komunikace řešila v HTTP protokolu. Díky nenáročnosti si tento protokol našel využití nejen na webových stránkách, kam byl původně

navržen, ale i v ostatních oblastech, jako jsou například mobilní aplikace a Internet věcí. U tohoto protokolu, stejně jako u většiny výše uvedených, nezáleží na formátu přenášených dat, a tak lze přenášet prostý text, JSON a XML. [7]

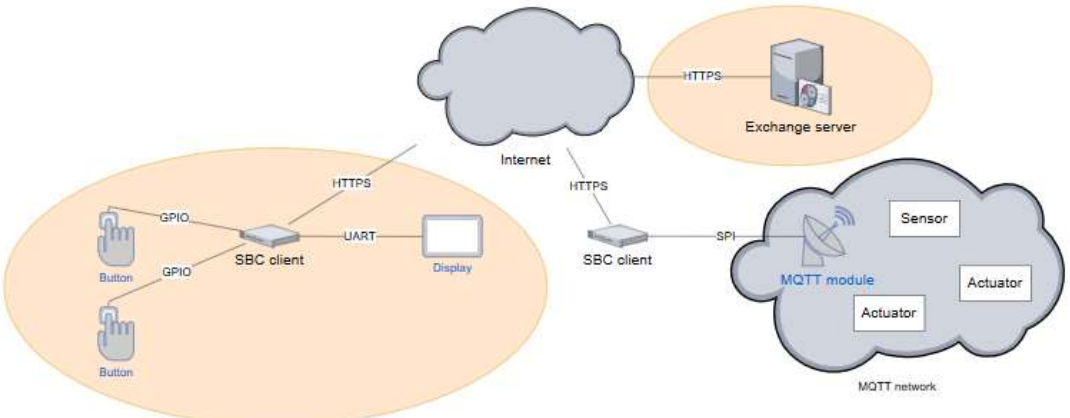
2.3 Výchozí stav implementace

Tato práce navazuje na výsledky individuálního semestrálního projektu a rozšiřuje je. V rámci tohoto projektu byl vytvořen základní koncept a provedena implementace univerzální IoT sítě. Ta byla v testovacím provozu nasazena od listopadu 2018 do února 2019 a bylo zjištěno několik zásadních nedostatků.

Prvním nedostatkem je existence jediného bodu selhání v podobě jednoho serveru s veškerou logikou. Pokud dojde k výpadku serveru, okamžitě přestane celá síť fungovat. Díky tomu může dojít a také v minulosti došlo ke ztrátě důležitých dat.

Dalším nedostatkem, který testovací provoz odhalil, je enormní datový přenos. Během 48 hodin vyměnil jeden uzel se serverem 3 GB dat a 99,93 % těchto dat byla výměna Secure Sockets Layer / Transport Layer Security (SSL/TLS) klíčů. To je způsobeno využitím komunikace prostřednictvím HTTPS protokolu, kdy se uzel serveru neustále aktivně dotazuje. Při tomto dotazování pokaždé vytváří nové HTTPS spojení a při navazování spojení dochází k výměně SSL/TLS klíčů, které jsou několikanásobně větší než samotná přenášená data.

Třetím nedostatkem byl problém se správou paměti. Serverová logika byla v rámci semestrálního projektu implementována v PHP 7.2. Testovací provoz ale odhalil, že v PHP serveru Apache2 existuje chyba, která způsobuje postupné zaplnění operační paměti. To mělo za následek, že bez pravidelných manuálních zásahů došlo k pádu serveru Apache2, a tím k nefunkčnosti celé IoT sítě.



Obrázek 2.1 Architektura semestrálního projektu

Na obrázku 2.1 lze vidět architekturu, kterou využíval původní systém IoT sítě vytvořené v rámci semestrálního projektu. Oranžově označené části byly implementovány. Veškerá logika systému se nacházela na Exchange[1] serveru, který zajišťoval zpracování zpráv. Jednodeskové počítače, označené jako SBC klienti, se k serveru připojovaly v pravidelném intervalu pomocí REST rozhraní přes HTTPS.

[1] Nejedná se o Microsoft Exchange server.

3 Návrh

3.1 Volba logického modelu

Jako první je třeba zvolit logický model celé IoT sítě. Cílem je, aby síť byla spolehlivá a co nejvíce univerzální. Měl by být tedy zvolen takový model, který umožní celému systému fungovat i v případě, že dojde k výpadku některých jeho částí. Cloud vypustíme rovnou, kvůli jeho závislosti na třetí straně a také požadavku na dobrou konektivitu do vnějšího světa. Model Peer – to – Peer by se mohl zdát jako vhodný, ale své využití nalezne spíše v symetrických systémech. Jeho další nevýhodou je náročnost implementace. Tento model také nevypadá jako příliš vhodný pro očekávaný záměr. Master – Slave model také vyřadíme, protože vyžaduje, aby uzel v roli Master mohl otevřít spojení se všemi svými uzly v roli Slave. Toto může být v některých případech velká překážka, zvláště, je-li přítomen překlad IP adres (NAT). Zbývá volba mezi Publish – Subscribe a Client – Server. Z těchto dvou modelů je výhodnější Publish – Subscribe, který také bude použit. Jeho velkým pozitivem je, že se jedná o velmi rozšířený model, který je pro aplikace IoT často využíván. Tento model například využívá IBM Bluemix. Publish – Subscribe řeší samotnou distribuci dat na příslušné uzly a snižuje provázanost jednotlivých částí. Lze tak celý systém rozšiřovat připojováním dalších nových částí bez nutnosti zásahu do částí již existujících.

3.2 Architektura systému

3.2.1 Základní architektura

Dalším krokem je volba samotné architektury. V síti, resp. systému, je potřeba oddělit hardwarovou realizaci od logické. Vezměme v úvahu místnost v domě, ve které jsou instalována čidla na vlhkost, kouř a pohyb. Může se dokonce jednat o sestavu čidel v jediném pouzdře. Z ekonomického a praktického hlediska lze předpokládat, že tato čidla budou obsahovat aktivní prvek zajišťující jejich obsluhu, nebo budou k takovému prvku připojena. Pokud budeme mít takových místností v domě více, lze vytvořit tři různé systémy (protipožární, zabezpečovací a klimatický), a to napříč všemi místnostmi.

Výše uvedeného požadavku lze dosáhnout tak, že budou v systému existovat 2 typy uzlů, fyzický a logický. Fyzický uzel bude mít za úkol sběr dat a ovládání fyzických prvků, jako například relé a stykače. Logický uzel bude obsahovat logiku a veškerou funkcionalitu. Fyzický uzel předá sesbíraná data logickému uzlu, ten data zpracuje a výsledek v podobě příkazu k akci předá fyzickému uzlu, který jej vykoná. Zároveň by bylo vhodné, aby všechny uzly mezi sebou

byly propojeny pouze volně a nevznikla tak potřeba, aby jednotlivé uzly věděly o těch ostatních. Na druhou stranu by bylo velice praktické, aby každý mohl komunikovat s každým. Toho lze dosáhnout komunikací přes prostředníka, nebo komunikací přes sdílené médium.

Příklad možné podoby architektury v praktickém nasazení lze vidět na souhrnném obrázku 3.1 na konci kapitoly 3.2. V horní části se nachází uskupení tří logických uzlů, které tvoří logický cluster. V dolní části se nacházejí dva fyzické uzly připojené k různým zařízením, která buď ovládají, nebo z nich získávají data.

3.2.2 Návrh výměny dat

Vzhledem k volbě logického modelu Publish – Subsribe je třeba vhodně navrhnout výměnu dat mezi uzly. Bylo by možné toto řešení implementovat od základu, ale je to zcela zbytečné. Existuje již několik hotových řešení, která lze přímo využít. Tato řešení se nazývají middleware a mezi nejčastěji využívané patří ActiveMQ, RabbitMQ, Apache Kafka, Eclipse Mosquitto a řada dalších. Pro výměnu dat mezi uzly byla vybrána Apache Kafka. Jde o streamovací a logovací distribuovanou platformu s vysokou datovou propustností, která umožňuje předávat data mezi různými systémy v téměř reálném čase, s naprostou spolehlivostí. Lze ji tedy chápat jako virtuální sběrnici. Výhodou Apache Kafka je, že dokáže automaticky řešit škálování a vyvažování zátěže. Jejím využitím výrazně klesá náročnost na implementaci celého systému.

Protože se v modelu Publish – Subsribe pracuje s tématy, ke kterým se přihlašují klienti v rolích producenta, nebo konzumenta, lze výměnu dat zefektivnit a zpřehlednit vhodným nastavením témat. V systému bude potřeba vytvořit tři komunikační kanály. První kanál zajišťuje přenos dat od fyzického uzlu k logickému, druhý od logického uzlu k fyzickému a třetí výměnu dat mezi logickými uzly. Musí tedy vzniknout tři odpovídající témata.

V tématu pro přenos dat od fyzického uzlu k logickému (dále téma označeno jako Node-to-Server, zkr. N2S) budou všechny fyzické uzly producenti a logické uzly konzumenti. Tím se zamezí zbytečnému přenosu dat určených pro logické uzly na ostatní fyzické uzly.

Další bude téma pro opačný směr komunikace (označeno jako Server-to-Node, zkr. S2N), tedy od logických uzlů k fyzickým uzlům. Zde budou všechny fyzické uzly konzumenti a logické uzly producenti. Vzniká však otázka, jestli je vhodné, aby zpráva od logického uzlu určená jednomu konkrétnímu fyzickému uzlu byla doručena všem fyzickým uzlům. Na ni nelze jednoznačně odpovědět. Apache Kafka umožňuje nastavení filtrace na konzumentech a zároveň

je očekáván velmi malý přenos dat v porovnání s tématem N2S. Pro specifické případy s velkým objemem komunikace by bylo možné vytvořit mechanismus, kdy se logický a fyzický uzel dohodnou a vytvoří separátní téma. Zatím ale neexistuje reálná potřeba pro tento mechanismus z důvodu chybějícího měření datového přenosu či přílišného zatížení celého systému.

Poslední je téma pro komunikaci mezi logickými uzly (dále označeno jako Server-to-Server, zkr. S2S). Zde mezi sebou budou komunikovat logické uzly a vyměňovat si režijní a servisní zprávy.

Další záležitostí, kterou je třeba v rámci výměny dat vyřešit, je otázka, jaká data budou přenášena. Celý systém je tedy rozdělen na dvě části – fyzické a logické uzly. Cílem je, aby veškerá logika byla umístěna na logických uzlech. Z toho vyplývá, že není třeba, aby fyzický uzel tato data uměl interpretovat. Fyzický uzel se z pohledu dat bude chovat transparentně a data, která přijme na fyzických rozhraních, nepozměněná předá logickým uzlům. Přenášet se budou surová data zapouzdřená do vhodně formátovaných zpráv.

3.2.3 Návrh řešení s ohledem na spolehlivost

Aby bylo možno zajistit vysokou dostupnost a spolehlivost celého systému, je třeba vyřešit otázku odolnosti proti výpadku. V případě výpadku fyzického uzlu není pravděpodobně možné počítat s jeho nahrazením jiným uzlem z důvodu ekonomického i praktického, protože by to znamenalo znásobení hardwaru. Reálně bude výpadek fyzického uzlu vyžadovat řešení v podobě fyzického zásahu. Z principu fungování systému ovšem výpadek fyzického uzlu nepředstavuje kritické riziko, které by znamenalo celkovou nefunkčnost. Výpadkům logických uzlů lze předejít, a to jejich znásobením a vzájemným propojením do clusteru. Zde je znásobení hardwaru dokonce žádoucí, protože lze provádět vyvažování zátěže a škálování. Každý logický uzel může zpracovávat data od několika fyzických uzlů a v případě jeho výpadku může jeho činnost snadno převzít jiný logický uzel.

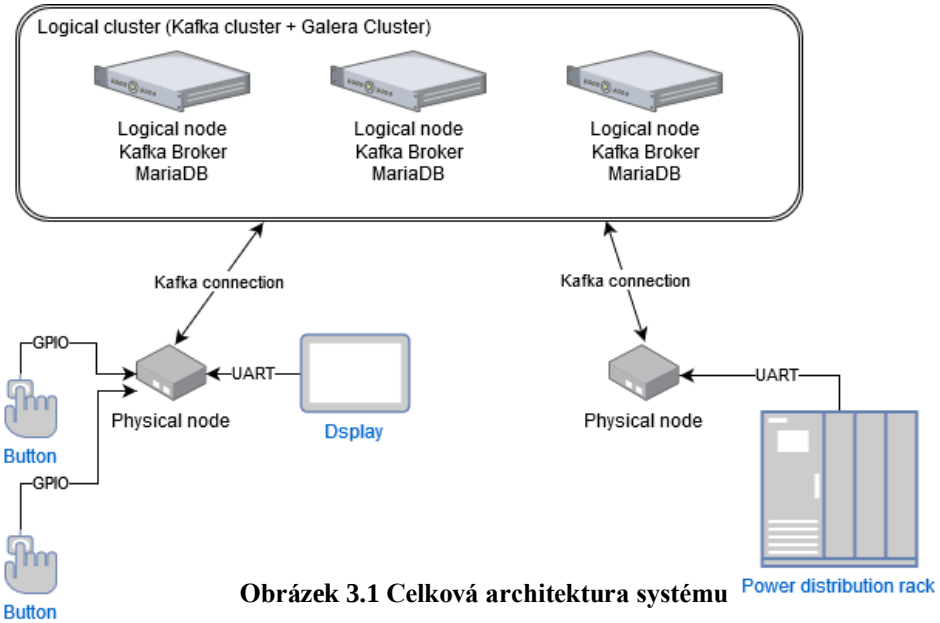
Pro řízení clusteru tvořeného logickými uzly je možné použít symetrický algoritmus, ale pro navrhovaný systém je vhodnější symetrii rozbít a využít nesymetrického přístupu, kdy v clusteru bude existovat vůdce. Ten bude zodpovědný za řízení clusteru a činnosti, které stačí v celém clusteru provádět pouze na jednom uzlu.

3.2.4 Návrh fyzického uzlu

Na samotný fyzický uzel máme několik požadavků. Musí disponovat konektivitou k internetu nebo lokální síti pomocí běžně využívaných technologií, a to Wi-Fi 2.4G, Wi-Fi 5G a Ethernet RJ45. Nemusí podporovat všechny technologie, ale minimálně jednu z výše uvedených ano. Dále musí disponovat standardními rozhraními, která se využívají v elektronice. Jde zejména o General Purpose Input Output (GPIO) piny, Universal Synchronous / Asynchronous Receiver and Transmitter (USART nebo UART) a Inter-Integrated Circuit (I2C), dále třeba Serial Peripheral Interface (SPI), Two Wire Interface (TWI). Podpora GPIO a UART je minimální nutná u všech uzlů. Dále by měl fyzický uzel disponovat jistou rozšiřitelností z pohledu dodatečného softwaru. Na takovém uzlu může být z důvodu zajištění spolehlivosti nasazen dohledový systém. Zároveň by měl být fyzický uzel cenově dostupný pro případ jeho fyzického selhání a snadné náhrady. Jako přijatelnou částku stanovme maximálně 1500 Kč. Všechny tyto požadavky nejlépe dokáže splnit jednodeskový minipočítač typu Raspberry Pi a jeho klony. Díky podpoře standardního Linuxu, nízké pořizovací ceně, velkému množství rozhraní, velké škále variant a široké podpoře a komunitě jde o ideální řešení pro fyzické uzly. Přes tyto minipočítače lze připojit prakticky jakékoli elektronické zařízení k internetu.

3.2.5 Návrh logického uzlu

Požadavků na logický uzel je výrazně méně, ale je potřeba na nich důrazně trvat, protože logický uzel je klíčový prvek celého systému. U logického uzlu požadujeme konektivitu k internetu nebo lokální síti výhradně přes metalické nebo optické médium. Dále očekáváme



Obrázek 3.1 Celková architektura systému

rychlé zpracování dat přicházejících z fyzických uzlů a perzistenci a synchronizaci některých provozních dat i po restartu logického uzlu. Na logickém uzlu bude spuštěno několik služeb zároveň. Middleware server Apache Kafka, databázový server MariaDB a samotná aplikace logického uzlu. Apache Kafka uvádí v dokumentaci minimální požadavek 2 GB RAM. Pro databázový server MariaDB neexistují minimální ani doporučené požadavky, ale obecně je defaultní konfigurace nastavena na 1 GM RAM. Tyto požadavky nejlépe splní generický server ideálně se čtyř jádrovým procesorem a minimálně 4 GB RAM. Server může být i virtuální, běžící na VM hostiteli.

3.3 Formát přenášených dat

Existuje velké množství formátů pro přenos strukturovaných dat. Nejčastěji se využívají formáty XML a JavaScript Object Notation (JSON). Výhodou JSON formátu je jeho efektivita, oproti XML, který je velmi verbalistický. U XML tak může dojít k situaci, kdy užitečná přenášená data jsou menší než jejich obálka v podobě XML značek. Tento problém u formátu JSON může sice také nastat, ale jen velmi výjimečně, díky minimalistickému způsobu, jakým JSON data přenáší. Z těchto důvodů budou data mezi uzly přenášena jako zprávy ve formátu JSON. Každá zpráva bude mít tři povinné údaje – identifikátor uzlu, časovou známku a typ zprávy. V závislosti na typu zprávy budou následovat další přenášená data.

3.3.1 Zprávy přenášené mezi fyzickými a logickými uzly

3.3.1.1. Identifikátory fyzických rozhraní

Pro identifikaci jednotlivých fyzických rozhraní uzlů je potřeba vytvořit jednotný popisný mechanismus, který umožní jednoznačnou identifikaci rozhraní na fyzickém uzlu. Obecně platí v mikroprocesorové elektronice v označování rozhraní nepsaný koncensus.

GPIO piny se označují pomocí písmena registru, ke kterému jsou připojeny, a čísla bitu v daném registru jsou indexována od nuly. Někdy se na začátku uvádí ještě písmeno P jako port. U GPIO pinů se tedy lze setkat s označením B4 i PB4, což označuje pátý bit registru B. Většina výrobců ARM procesorů ještě v dokumentaci uvádí převod z uvedeného označení na číselné. Například u 32bitových ARM procesorů je převod založen na vzorci (Rovnice 3.1). Proměnná x označuje číselný identifikátor pinu, p je pořadové číslo písmene portu, v abecedě indexováno od nuly (A = 0, B = 1, …) a n je číslo bitu. Podle této rovnice je pro pin PB4 číselné označení 36.

$$x = p * 32 + n$$

Rovnice 3.1

Pro rozhraní UART se používá pořadové číslování od nuly. Tedy UART0, UART1 atd. Stejné schéma využívají i ostatní rozhraní jako SPI, TWI, I2C a podobně. Vyskytují se i alternativní označení, která jsou produktem vynalézavosti výrobce. Místo UART0 se lze setkat i se Serial0.

Z výše uvedeného vyplývá, že vzhledem k velké rozlišnosti označování fyzických rozhraní by práce s původním označením byla velmi problémová. Podobný problém byl již v minulosti řešen v oblasti distribučních energetických sítí (elektrárenství, vodárenství, plynárenství). Zde se řešila jednoznačná identifikace proměnných a parametrů v rámci měřících přístrojů (elektroměry, vodoměry, plynoměry) z důvodu sjednocení a kompatibility mezi jednotlivými výrobci. Výsledkem této snahy je protokol Device Language Message Specification / Companion Specification for Energy Metering, zkráceně DLMS/COSEM, též jen DLMS, specifikován normou IEC62056 [8]. Zde došlo k vyřešení problému jednoznačným hierarchickým roztříděním proměnných a parametrů do skupin a podskupin.

Totéž lze provést i s fyzickými rozhraními. Poslouží k tomu 6 bajtů dlouhý identifikátor. Jednotlivé bajty se při zápisu oddělí tečkou. První bajt (index 0) označuje typ rozhraní, zda jde o GPIO pin, UART, SPI apod. Druhý bajt označuje umístění rozhraní. Například UART může být nativní přímo v procesoru fyzického uzlu, nebo připojen přes Universal Serial Bus - USB převodník. Toto může mít celou řadu dopadů, které je potřeba zohlednit. Poslední 4 bajty jsou rezervovány pro detailnější popis rozhraní a jejich význam závisí na typu rozhraní v prvním bajtu. Konkrétní řešení a význam jednotlivých bajtů pro jednotlivá rozhraní lze nalézt v tabulce 3.1 níže.

Tabulka 3.1 Identifikátory rozhraní

Rozhraní	0. byte	1. byte	2. byte	3. byte	4. byte	5. byte
GPIO	0	0	0	0	0-255: číslo portu	0-255: číslo bitu
UART	1	0: Nativní rozhraní 1: USB převodník	0	0	0	0-255: systémové číslo UART
I2C	2	0: Nativní rozhraní 1: USB převodník	0	0	0	0-255: systémové číslo I2C
SPI	3	0	0	0	0	0-255: systémové číslo SPI

3.3.1.2. Konfigurační zpráva

Konfigurační zpráva slouží pro inicializaci fyzického uzlu. Jde o zprávu, ve které logický uzel nově připojenému fyzickému uzlu předá konfiguraci, podle které se má fyzický uzel nastavit. Součástí konfigurační zprávy je konfigurační zpráva rozhraní a interval, v jakém má fyzický uzel logickému zasílat zprávu o aktivitě. Konfigurační zprávu odesílá jako první fyzický uzel logickému uzlu jako žádost o konfiguraci. Ten ji vyplní a odešle zpět fyzickému uzlu. Struktura zprávy je zobrazena v tabulce 3.2.

Tabulka 3.2 Struktura konfigurační zprávy

Hodnota	Datový typ	Popis
timestamp	Instant	Vznik zprávy
nodeLogin	String	Login fyzického uzlu
boardName	String	Uživatelský název fyzického uzlu
configRequest	boolean	Požadavek konfigurace (defaultně 0, od fyz. uzlu 1)
commInterval	Integer	Komunikační interval fyzického uzlu
interfacesConfig	InterfaceConfigMessage	Konfigurace fyzických rozhraní uzlu

3.3.1.3. Konfigurační zpráva rozhraní

Konfigurační zpráva rozhraní obsahuje seznam rozhraní, která má fyzický uzel aktivovat, a jejich příslušné parametry. Pro GPIO pin například směr (vstup, výstup) a pro UART přenosovou rychlost, paritu a další. Tuto zprávu posílá logický uzel fyzickému uzlu. Ten ji neodesílá, protože prioritu má vždy konfigurace uložená na logickém uzlu. Struktura zprávy je zobrazena v tabulce 3.3.

Tabulka 3.3 Struktura konfigurační zprávy rozhraní

Hodnota	Datový typ	Popis
timestamp	Instant	Vznik zprávy
nodeLogin	String	Login fyzického uzlu
interfacesConfig	Map <String, Map<InterfaceSettingsParams,String>>	Konfigurace fyz. rozhr.

3.3.1.4. Stavová zpráva rozhraní

Stavová zpráva rozhraní má dva významy. Pokud jde ve směru od logického uzlu k fyzickému, jedná se o data, která má fyzický uzel na svá výstupní rozhraní zapsat. Pokud jde od fyzického uzlu k logickému, obsahuje data, která fyzický uzel přečetl na svých vstupních rozhraních. Jedná se o nejčastěji vyměňovanou zprávu mezi logickým a fyzickým uzlem. Zprávy se odesílají pouze tehdy, pokud na vstupním rozhraní dojde ke změně stavu, nebo je potřeba stav výstupního rozhraní změnit. Případně si logický uzel může zprávu o stavu rozhraní konkrétního fyzického uzlu vyžádat. Struktura zprávy je zobrazena v tabulce 3.4.

Tabulka 3.4 Struktura stavové zprávy rozhraní

Hodnota	Datový typ	Popis
timestamp	Instant	Vznik zprávy
nodeLogin	String	Login fyzického uzlu
interfacesStates	Map <String,List<Integer>>	Mapa dat (identifikátor rozhraní => seznam bajtů)

3.3.1.5. Zpráva o aktivitě

Zprávu o aktivitě zasílá fyzický uzel logickému v předem stanoveném intervalu. Tato zpráva slouží pro kontrolu stavu fyzického uzlu v době, kdy neodesílá data na logický uzel. Logický uzel tak může sledovat stav fyzického uzlu a reagovat na jeho výpadek. Struktura zprávy je zobrazena v tabulce 3.5.

Tabulka 3.5 Struktura zprávy o aktivitě

Hodnota	Datový typ	Popis
timestamp	Instant	Vznik zprávy
nodeLogin	String	Login fyzického uzlu

3.3.2 Zprávy přenášené mezi logickými uzly

3.3.2.1 Připojení logického uzlu

Zprávu o připojení logického uzlu do sítě odesílá nově připojený logický uzel do tématu S2S. Tato zpráva obsahuje informace, jako je mimo jiné jeho identifikátor, a je zpracována všemi uzly v síti. Slouží pro zaregistrování nově připojeného logického uzlu do clusteru. Struktura zprávy je zobrazena v tabulce 3.6.

Tabulka 3.6 Struktura zprávy připojení logického uzlu

Hodnota	Datový typ	Popis
targetID	Integer	ID cílového logického uzlu (0 = všechny uzly)
senderID	Integer	ID zdrojového logického uzlu
timestamp	Instant	Vznik zprávy
connect	boolean	Typ zprávy (1 připojení uzlu, 0 odpojení uzlu)
groupID	String	Skupinový identifikátor (viz kapitola 4.2.2.4)

3.3.2.2 Volba vůdce

Volební zprávu odesílá každý logický uzel do tématu S2S v době nepřítomnosti vůdce. Tato zpráva slouží pro provedení volby nového vůdce a zároveň obsahuje informace o kandidátovi na jeho pozici. Struktura zprávy je zobrazena v tabulce 3.7.

Tabulka 3.7 Struktura zprávy volby vůdce

Hodnota	Datový typ	Popis
targetID	Integer	ID cílového logického uzlu (0 = všechny uzly)
senderID	Integer	ID zdrojového logického uzlu
timestamp	Instant	Vznik zprávy
leaderCandidate	Integer	ID uzlu kandidujícího na vůdce clusteru

3.3.2.3 Informace o zvoleném vůdci

Zpráva o zvoleném vůdci obsahuje informace o novém vůdci clusteru. Odesílá ji vždy vůdce do tématu S2S a pouze ve dvou případech. V prvním případě, že se připojil nový logický uzel, ve druhém, že vůdce vyhrál volbu a oznamuje ostatním logickým uzlům své vítězství. Struktura zprávy je zobrazena v tabulce 3.8.

Tabulka 3.8 Struktura zprávy informace o zvoleném vůdci

Hodnota	Datový typ	Popis
targetID	Integer	ID cílového logického uzlu (0 = všechny uzly)
senderID	Integer	ID zdrojového logického uzlu
timestamp	Instant	Vznik zprávy
electedLeader	Integer	ID uzlu, který se stal vůdcem

3.3.2.4 Zpráva o aktivitě

Zprávu o aktivitě odesílá každý logický uzel do tématu S2S a slouží ostatním logickým uzlům pro kontrolu aktivity. Pokud logický uzel přestane zprávu o aktivitě posílat, je po určité době považován za odpojený. V případě, že logický uzel, který přestal posílat zprávu o aktivitě, byl zároveň vůdce, dojde k nastartování volby nového vůdce. Struktura zprávy je zobrazena v tabulce 3.9.

Tabulka 3.9 Struktura zprávy o aktivitě

Hodnota	Datový typ	Popis
targetID	Integer	ID cílového logického uzlu (zde vždy 0 = broadcast)
senderID	Integer	ID zdrojového logického uzlu
timestamp	Instant	Vznik zprávy

3.3.2.5 Zpráva o následnictví

Zpráva o následnictví je odesílána vůdcem logického clusteru. Obsahuje pořadí, ve kterém logické uzly volí nového vůdce. Tato zpráva je odeslána ve třech případech, a to pokud se připojí nový uzel, odpojí se uzel nebo je zvolen nový vůdce. Struktura zprávy je zobrazena v tabulce 3.10.

Tabulka 3.10 Struktura zprávy o následnictví

Hodnota	Datový typ	Popis
targetID	Integer	ID cílového logického uzlu (zde vždy 0 = broadcast)
senderID	Integer	ID zdrojového logického uzlu
timestamp	Instant	Vznik zprávy
topology	Map<Integer, Integer>	Mapa následnictví (uzel => jeho následník)

3.4 Návrh zabezpečení komunikace

Důležitým prvkem celého systému je také zabezpečení komunikace mezi jednotlivými uzly. Při návrhu zabezpečení je potřeba nalézt a zajistit maximum možných vektorů útoku v podobě odposlechu a podstrčení komunikace. To je možné důslednou analýzou existujících komunikačních kanálů v systému. Takové kanály existují v systému dva. Prvním je veškerá komunikace prostřednictvím Apache Kafka a druhým je komunikace databázového clusteru Galera.

3.4.1 Zabezpečení Apache Kafka

Apache Kafka v defaultní instalaci pracuje zcela bez zabezpečení. Data přenáší nešifrovaná a neprovádí autentizaci a autorizaci připojených klientů. Veškeré uvedené funkce ale podporuje, a tak je jejich využití otázkou správné konfigurace.

Nejvhodnější a nejčastěji využité řešení na zabezpečení komunikace a autentizaci klientů je použití SSL certifikátů. Každému klientovi (konzumentovi nebo producentovi) a Apache Kafka serveru je vydán certifikát podepsaný certifikační autoritou. To umožňuje uzlům clusteru ověřit identitu klientů. Zároveň lze pomocí stejných certifikátů zašifrovat komunikaci jak mezi klienty a uzly clusteru, tak i uvnitř clusteru.

Autorizace v rámci Apache Kafka řeší oprávnění klienta přistupovat k určitým tématům, číst je a zapisovat do nich. Autorizace je řešena pomocí seznamů řízení přístupů – Access Control Lists (ACL). V těchto seznamech se definuje pro každého klienta jeho přístup k tématům. Tyto seznamy se udržují manuálně. Vzhledem k zamýšlené dynamičnosti celého navrhovaného systému je ale manuální zpráva autorizace zcela nepraktická. Dále je potřeba zmínit, že všechny logické a fyzické uzly přistupují ke clusteru a tématům zcela bez rozdílů a cluster je určen exklusivně pro navrhovaný systém. V tomto konkrétním případě lze tvrdit, že pokud proběhne úspěšná autentizace klienta, lze na klienta hledět jako autorizovaného k přístupu k tématům v clusteru. V případě, že by cluster nebyl využíván exklusivně pouze systémem navrhovaným v této práci, výše uvedený předpoklad nelze použít. V takovém případě by bylo nezbytné vyřešit správu ACL. Zde se nabízí řešení v podobě automatizovaného mechanismu, který by správu ACL zajistil. Tato práce však využije předpokladu exklusivního využití clusteru a autorizaci řešit nebude.

3.4.2 Zabezpečení Galera clusteru

Galera cluster podporuje zabezpečené šifrované spojení mezi jednotlivými uzly clusteru a klienty pomocí SSL protokolu. V základní konfiguraci jsou tato spojení nezabezpečená a lze je velmi snadno odposlechnout a pozměnit. Zabezpečení komunikace mezi klientem a clusterem je řešeno v rámci konfigurace jednotlivých MariaDB instancí. Zabezpečení komunikace mezi jednotlivými uzly clusteru je třeba nakonfigurovat zvlášť v rámci konfigurace Galera clusteru. Tento fakt může u nepozorného uživatele – administrátora vést k nabytí falešného dojmu, že po zabezpečení komunikace mezi klientem a clusterem zajistil dostatečnou bezpečnost. Data však mohou být kompromitována během jejich replikace mezi uzly Galera clusteru.

Konfigurace celého clusteru probíhá staticky a totéž platí i pro zabezpečení. Je třeba uvést IP adresu každého uzlu clusteru a definovat cesty k SSL klíčům a certifikátům. V praxi to znamená vytvoření jednoho konfiguračního souboru, který je poté kopírován na všechny uzly clusteru.

V návrhu logického uzlu v kapitole 3.2.5 bylo uvedeno, že instance Galera clusteru, tedy server MariaDB, bude součást logického uzlu. Proto bude konfigurace zabezpečení clusteru řešena pouze na úrovni komunikace mezi instancemi MariaDB na jednotlivých logických uzlech. Komunikace mezi aplikací logického uzlu - klientem a instancí MariaDB bude vedena lokálně v rámci systému pomocí lokální IP adresy 127.0.0.1, známé jako localhost. Tato varianta zabezpečení je dostačující, protože odposlech a podstrčení komunikace vedené přes localhost znamená nutnost ovládnutí lokálního stroje[2]. V případě ovládnutí lokálního stroje má útočník k dispozici jiné, silnější vektory útoku než útok na komunikaci vedenou přes localhost.

3.5 Návrh integrace s jinými systémy

Aby bylo možno využít veškerý potenciál systému navrhovaného v této práci, je potřeba zahrnout možnost komunikace se systémy již existujícími. Těch je velké množství a způsoby, kterými komunikují, jsou velmi rozmanité. Je také nutno vzít v potaz fakt, že tyto externí systémy také procházejí vývojem. Z těchto důvodů je nutno k návrhu vhodného řešení integrace přistupovat s jistou mírou opatrnosti a abstrakce.

Celý systém je navrhován jako vysoce modulární a logicky se nabízí, aby řešení integrace s ostatními systémy mělo také modulární podobu. Tyto integrační moduly by měly mít podobu samostatné integrační mikroslužby. Takové řešení je velmi výhodné, a to z několika důvodů. Prvním důvodem je, že cluster logických uzlů je dynamický a může v něm docházet ke změnám a výpadkům. Druhým důvodem je, že logický cluster obsahuje veškerá data. Třetím důvodem je rozšiřitelnost. Samostatná mikroslužba může na změny v logickém clusteru reagovat a vybrat nejvhodnější uzel v logickém clusteru, ze kterého bude získávat potřebná data. Dále lze velmi snadno danou mikroslužbu horizontálně škálovat. Přidání nové integrační možnosti je triviální důsledek výše uvedeného. Stačí vytvořit a nasadit novou mikroslužbu.

[2] Tým Project Zero ze společnosti Google uveřejnil v roce 2014 chybu CVE-2014-9295, která umožňovala z lokální sítě útočníkovi pomocí IPv6 a služby ntpd podstrčení komunikace na localhost oběti běžící na OS X a některých linuxových distribucích. Tato chyba již byla opravena. Zdroj: (https://googleprojectzero.blogspot.com/2015/01/finding-and-exploiting-ntpd.html)

4. Implementace

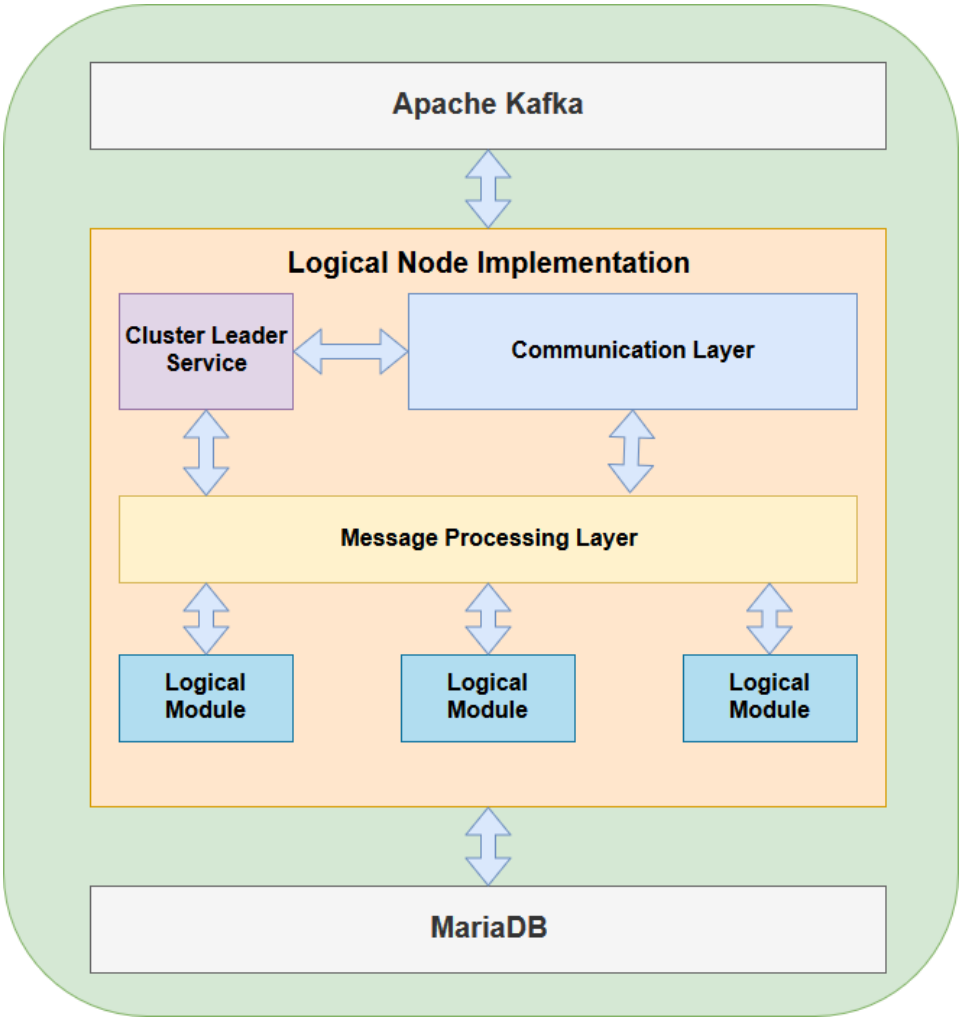
4.1 Obecné informace k implementaci

Jako operační systém použitý na logických a fyzických uzlech byl zvolen Debian 9, a to pro své nízké hardwarové nároky, univerzálnost a širokou podporu a znalostní bázi.

Veškeré části systému jsou implementovány v jazyce Java ve verzi 11. Ten byl zvolen z důvodu jeho rozšířenosti a široké paletě knihoven, které jsou k dispozici. Jistě by bylo vhodnější pro implementaci na fyzickém uzlu využít jazyk C/C++, který je pro práci s hardwarem mnohem lépe připravený. To by ale vedlo ke zbytečně vyšší složitosti implementace. Navíc Java disponuje funkcionalitou Java Native Interface (JNI), která umožňuje využívání C/C++ knihoven. Tak lze i v Javě pracovat s hardwarem, ale dochází ke ztrátě její hlavní výhody, kterou je možnost běhu na různých platformách. Toto ale není omezení, protože systém je navrhován pro konkrétní soubor ARM procesorů výrobce Allwinner. Jednotlivá specifika ARM procesorů jiných než Allwinner Java řešit nedokáže a je třeba tato specifika zohlednit při samotné implementaci.

4.2 Implementace logického uzlu

Logický uzel se skládá ze dvou částí. První část je software třetích stran, který je pro funkčnost celého systému nezbytný. Druhou částí je samotná implementace logiky uzlu. Tento přístup je též známý jako mikroslužby, častěji označován anglickým pojmem mircoservices. Jeho výhodou je, že nevzniká monolitická aplikace, která je náročná na údržbu. Zároveň tento přístup umožňuje znovu použít již existující řešení v nových projektech. Další výhodou je možnost rozložení jednotlivých mikroslužeb napříč dostupnou infrastrukturou. V návrhu logického uzlu v kapitole 3.2.5 je uvedeno, že na logickém uzlu se počítá s během služeb Apache Kafka a MariaDB. To ale není jediné možné řešení. Lze například Apache Kafka nainstalovat jako cluster na tři servery, MariaDB jako Galera cluster na pět serverů a samotnou logiku spustit na čtyřech serverech náhodně vybraných z výše uvedené množiny osmi serverů. Úpravou vhodných konfiguračních souborů dosáhneme stejné funkce jako v návrhu. Zde je vidět výhoda mikroslužeb, která spočívá ve flexibilitě a škálovatelnosti. Dále ale bude využíván návrh z kapitoly 3.2.5. Kompletní implementace logického uzlu je zobrazena na obrázku 4.1 na následující straně.



Obrázek 4.1 Implementace logického uzlu

4.2.1 Software třetích stran

Jako první je potřeba nainstalovat balíčky Java Development Kit 11 určené pro systém Debian 9. Tento soubor balíčků nainstaluje na logický uzel Java Virtual Machine, která je nezbytná pro běh Apache Kafka a samotné logiky serveru.

Jako další se provede instalace middleware Apache Kafka ve verzi 2.2.0. Při instalaci je doporučeno postupovat dle dokumentace na webových stránkách daného softwaru. (https://kafka.apache.org), protože Apache Kafka není součástí standardních repositářů systému Debain. Apache Kafka pro svou funkci vyžaduje ještě další softwarovou službu a tou je Apache Zookeeper. Jeho instalaci není třeba řešit, protože je do Apache Kafka přibalen v základní konfiguraci. Ta ale není vhodná pro provoz Apache Kafka clusteru, protože v ní Zookeeper běží pouze v jedné instanci a tvoří tak jediný bod selhání. Je tedy nutno upravit konfiguraci pro Zookeeper. Oficiální webové stránky neposkytují příliš informací ohledně toho,

jak nakonfigurovat Apache Kafka a Apache Zookeeper do clusteru. To může být pro nového uživatele matoucí. Naštěstí se většina informací nachází v podobě komentářů přímo v konfiguračních souborech zookeeper.properties a server.properties. Zde je potřeba nastavit IP adresy jednotlivých logických uzlů s vhodnými porty.

Posledním instalovaným softwarem je databázový server MariaDB ve verzi 10.1. Zde je instalace výrazně jednodušší, protože MariaDB je součástí standardních repositářů systému Debian. Instalaci lze tedy provést pomocí příkazu apt-get. Na rozdíl od standardní instalace MariaDB serveru je potřeba provádět instalaci balíčků obsahujících podporu pro Galera cluster. Jedná se o balíčky mariadb-galera-server, mariadb-client a galera. Součástí instalace je průvodce instalací, který provádí uživatele skrze všechny nezbytné úkony a konfigurační kroky. Po úspěšné instalaci databázového serveru je nutno ještě provést konfiguraci Galera clusteru. To se provádí v souboru galera.cnf, který je umístěn v instalační cestě MariaDB serveru. V tomto souboru je nutno přidat adresy dalších logických uzlů tak, jak to bylo uvedeno v předchozím odstavci.

4.2.2 Implementace logiky

Logika je implementována v jazyce Java 11 s využitím frameworku Spring Boot a nástroje Maven. Framework Spring Boot umožňuje tvorbu microservice aplikací a jejich rapidní vývoj. Obsahuje celou řadu již implementovaných řešení nutných pro chod microservice aplikace. Je nutno pouze doplnit aplikační logiku a není třeba psát kód, který se stará o samotnou režii aplikace a nesouvisí přímo s logikou. Nástroj Maven slouží pro správu, řízení a automatizaci buildů aplikací. Umožňuje správu závislostí a knihoven použitých v projektu a sestavení celého projektu je tak výrazně jednodušší.

Celou logiku lze rozdělit do několika vrstev. První je komunikační vrstva, která řeší komunikaci pomocí Apache Kafka. Tato vrstva má za úkol příjem a odesílání zpráv. Druhá vrstva je zpracování zpráv. V ní jsou přijaté zprávy roztříděny podle typu a zpracovány. Pokud je daný logický uzel jediný v síti, nebo je vůdce clusteru, je navíc prováděna činnost vůdce clusteru, která spočívá v kontrole aktivity fyzických a logických uzlů a reakci na jejich případné výpadky. Jednotlivé vrstvy a jejich propojení jsou vyobrazeny na obrázku 4.1 ve střední části označené jako Logical Node Implementation.

4.2.2.1 Komunikační vrstva

Pro implementaci komunikační vrstvy je využita oficiální knihovna org.springframework.kafka pro Apache Kafka, která je součástí frameworku Spring. Tato vrstva je nakonfigurována jako konzument témat N2S a S2S a jako producent do témata S2S. Dále komunikační vrstva využívá knihovny Jackson pro převod z JSON dat na Java objekty a vice versa.

4.2.2.2 Vrstva zpracování zpráv

Vrstva zpracovávající zprávy třídí příchozí zprávy podle typu. Pokud se jedná o servisní zprávu od jiného logického uzlu, pak je zpracována přímo v této vrstvě. Zprávy od logických uzlů jsou zpracovávány v logických modulech. Logický modul je třída implementující konkrétní logiku. V případě přijetí zprávy od logického uzlu, je třeba nejprve pomocí databáze a dalších informací zjistit, jakým způsobem mají být data od uzlu interpretována a který logický modul má zpracování obstarat. Poté je vytvořena instance příslušného logického modulu a té jsou data předána ke zpracování.

4.2.2.3 Logický modul

Logický modul je část vrstvy zpracování zpráv a klíčová část implementace systému. Zde jsou již interpretována a zpracována data z fyzických uzlů. Logický modul je implementován pomocí rozšíření abstraktní třídy iot.system.iotserver.modules.AbstractModule a klíčové jsou zde dvě metody.

První metoda je processMessage. V ní dochází ke zpracování zprávy o stavu rozhraní fyzického uzlu, které spočívá v extrakci dat a jejich uložení do příslušné privátní proměnné modulu.

Druhou je metoda execute, která vykonává samotnou logiku modulu. Zde může dojít k výpočtům dalších hodnot, které nelze extrahovat přímo z přijatých dat, dále k odeslání zprávy jinému fyzickému uzlu nebo k vykonání logiky souvisejícího logického modulu. Tento přístup postupného zpracování a řetězení je velmi podobný návrhovému vzoru známému jako Command [9], nejde však o jeho přesnou implementaci.

4.2.2.4 Činnost vůdce a jeho volba

V rámci clusteru logických uzlů existuje vůdce. Vůdce je logický uzel, který nad rámec standardní činnosti zpracování dat z fyzických uzlů vykonává ještě navíc dohled nad ostatními logickými uzly a všemi fyzickými uzly. Pokud vůdce po definované době nezaznamená aktivitu nějakého uzlu (fyzického či logického), podnikne příslušná opatření. V případě výpadku fyzického uzlu jej označí jako neaktivní – offline a zapíše výpadek do logu. U logického uzlu oznámí všem ostatním logickým uzlům výpadek a nové pořadí následnictví pro volbu vůdce.

Vůdce clusteru je také zodpovědný za připojení nových logických uzlů do clusteru. Pokud by se nově připojovaný uzel choval při připojení nestandardně, pošle vůdce připojujícímu se uzlu příkaz k odpojení a ukončení aktivity. K tomu dojde například v situaci, kdy by se logický uzel připojoval pod jiným skupinovým identifikátorem[3], než využívá vůdce, či se stejným identifikátorem[4], který již využívá jiný uzel.

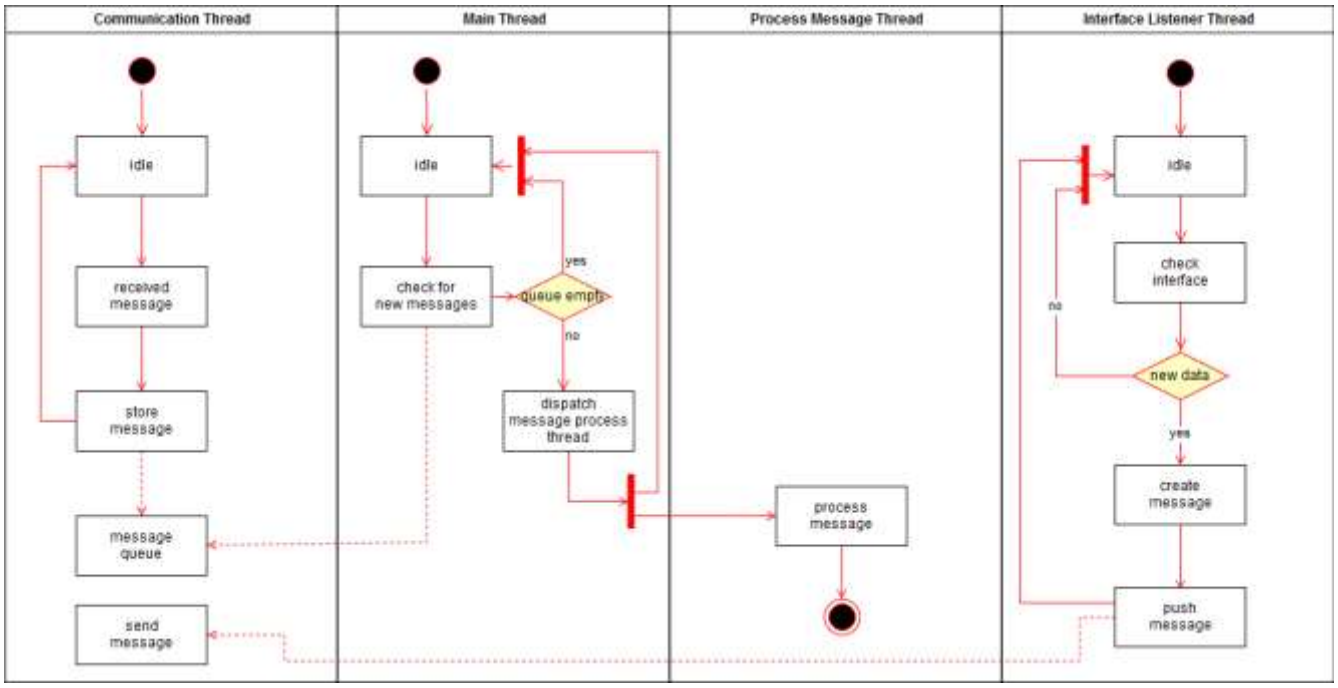
Standardně se vůdcem stává první uzel v clusteru logických uzlů. V případě, že vůdce z clusteru vypadne, ostatní uzly zahájí volbu vůdce podle algoritmu Chang – Roberts [1].

4.3 Implementace fyzického uzlu

Fyzický uzel je implementován velice jednoduše pomocí tří částí. První část je samostatné vlákno, obsluhuje komunikace s Apache Kafka clusterem. Druhá část je také vlákno zpracovávající příchozí zprávy a zapisující data na fyzická rozhraní. Třetí část sbírá data z fyzických rozhraní a sestavuje zprávy k odeslání. Celkový diagram implementace lze vidět na obrázku 4.2. Jednotlivé části jsou popsány v následujících podkapitolách.

[3] Pokud má cluster více uzlů – konzumentů, je příchozí zpráva doručena pouze jednomu uzlu ze skupiny. Dochází tak k rovnoměrnému vyvažování zátěže. Více na https://kafka.apache.org/documentation/#intro_consumers

[4] Identifikátor konkrétního uzlu – klienta připojeného k Apache Kafka.



Obrázek 4.2 Implementace fyzického uzlu

4.3.1 Komunikační vlákno

Pro implementaci komunikačního vlákna je využita stejná knihovna jako pro implementaci logického uzlu. Komunikační vlákno je zapouzdřeno v třídě exchange.ExchangeKafka, která je potomkem abstraktní třídy Exchange. Třída Exchange a její potomci mají pro jednoduchost pouze dvě veřejné metody sloužící k výměně zpráv, a to pushMessage a fetchMessage. První slouží k odeslání zprávy a druhá slouží k načtení zprávy.

Výměna zpráv mezi komunikačním vláknem a zbytkem aplikace je řešena pomocí blokující fronty LinkedBlockingQueue [10], která je thread-safe, a díky tomu dojde k zajištění synchronizace mezi vlákny.

4.3.2 Vlákno zpracování příchozích zpráv

Vlákno pro zpracování příchozích zpráv má na za úkol odběr příchozích zpráv z fronty a jejich zpracování v závislosti na typu. Nejzásadnější je zpracování zpráv s daty, která mají být propagována na fyzická rozhraní. Tyto zprávy jsou popsány v kapitole 3.3.1.4.

Zpracování zprávy s daty pro fyzické rozhraní je závislé na typu fyzického rozhraní. Propagace dat na GPIO pin bezpochyby využívá jiných funkcí operačního systému než propagace dat na UART rozhraní. U všech rozhraní lze nalézt tři společné činnosti. Čtení, zápis a konfigurace rozhraní. Této myšlenky bylo využito při objektovém návrhu, kde byla vytvořena

abstraktní třída BoardUtils s metodami readInterface a writeInterface. Tato třída umožňuje jednotný přístup ke všem rozhraním bez ohledu na jejich typ. Data jsou předávána pomocí třídy Interface. Třída BoardUtils je abstraktní, aby se do budoucna zajistila maximální kompatibilita aplikace s procesory a operačními systémy, které mohou být k dispozici. Potomci abstraktní třídy BoardUtils implementují konkrétní přístup k fyzickému rozhraní dle jeho typu, jenž lze určit pomocí identifikátoru popsaného v kapitole 3.3.1.1.

4.3.3 Část zpracování dat na fyzických rozhraních

V rámci práce jsou implementována pouze dvě rozhraní, a to GPIO piny a UART. Implementaci pro přístup k GPIO pinům zajišťuje souborový systém operačního systému a přístup k rozhraní UART je řešen pomocí knihovny jSerialComm, která je k dispozici přes Maven a je jediná pravidelně aktualizovaná. Další rozhraní implementována nebyla, protože v době vzniku této práce neexistovala žádná podpora v podobě knihovny, která by umožňovala z jazyku Java přístup k těmto rozhraním na procesorech Allwinner. Teprve v nedávné době byla oznámena experimentální podpora projektem Pi4J (https://pi4j.com) pro procesory Allwinner.

Zápis dat na fyzické rozhraní je prováděn pomocí metody writeInterface deklarované v abstraktní třídě BoardUtils. V případě GPIO pinu se jedná o zápis do příslušného souboru, který reprezentuje daný pin. Tento soubor je sledován operačním systémem a data zapsaná do něj jsou propagována na samotný pin. V případě rozhraní UART je zápis řešen metodou writeBytes třídy SerialPort z knihovny jSerialComm.

Mnohem větší výzvou bylo vyřešit čtení dat z rozhraní. Linuxové operační systémy mají filozofii, že vše je soubor, a tedy i zařízení a fyzická rozhraní jsou v systému reprezentována souborem. Prakticky stačí sledovat změny vhodného souboru a data z něj přečíst. Problémem je, že takový přístup je velmi drahý s ohledem na systémové prostředky a pomalý. Tyto problémy lze řešit přístupem přímo do paměti.

Čtení dat je zajištěno pomocí metody readInterface deklarované v abstraktní třídě BoardUtils. Pro UART je čtení velice triviální díky použité knihovně. Stačí při konfiguraci rozhraní UART reprezentovaného třídou SerialPort připojit k rozhraní vhodný listener (resp. observer) [11], který bude automaticky zavolán v případě příchozích dat. Pro získání dat z GPIO pinů byl zvolen přístup čtení dat ze souboru reprezentujícího konkrétní pin. Všechny vstupní piny, resp. příslušné soubory, jsou pravidelně vyčítány v intervalu 200 ms. V obou případech (GPIO pin a UART) jsou při detekci nových dat sestaveny stavové zprávy rozhraní a předány do komunikačního vlákna k odeslání logickým uzlům.

5. Testování a výsledky

Testování takto komplexního systému je třeba rozdělit do několika částí. Je nutno provést samostatné testy fyzického uzlu, logického uzlu, logického clusteru, komunikace mezi fyzickými uzly a logickým clusterem. Zároveň je třeba otestovat chování systému jako celku.

5.1 Testovací konfigurace

Pro testování byly vytvořeny dvě konfigurace. Jedna sloužila pouze k otestování logického clusteru a druhá pro testování systému jako celku v reálném nasazení. Toto rozdělení bylo nutné z důvodu omezených hardwarových zdrojů pro reálné nasazení.

5.1.1 Testovací konfigurace clusteru

Testování clusteru bylo prováděno na stroji s následující konfigurací:

- Procesor Intel Core i5 8250U 3,4 GHz/ 4 jádra – 8 vláken
- Operační paměť 16 GB RAM
- Operační systém Windows 10

Na něm bylo pomocí virtualizačního softwaru VMware Workstation vytvořeno 5 virtuálních strojů s následujícími zdroji:

- 2 procesorová vlákna
- Operační paměť 2 GB RAM

Jednotlivé virtuální stroje jsou nakonfigurovány jako logické uzly dle kapitoly 3.2.5. Pro všechny uzly clusteru byla vytvořena dedikovaná virtuální LAN, také pomocí nástroje VMware Workstation. Dále byl použit emulovaný fyzický uzel pro testování komunikace mezi fyzickým uzlem a clusterem.

5.1.2 Testovací konfigurace v reálném nasazení

Pro testování v reálném nasazení byla zvolena konfigurace skládající se z jednoho logického uzlu a dvou uzlů fyzických. Všechny uzly jsou připojeny na dedikovanou virtuální lokální síť. Jeden logický uzel byl zvolen z důvodu nedostatku hardwarových prostředků, kdy byl k dispozici pouze jeden fyzický server a logický uzel byl provozován jako virtuální stroj na tomto serveru.

Vytížení serveru dalšími aplikacemi a službami nedovoluje provoz více než jednoho logického uzlu. Konfigurace testovacího serveru je následující:

- Procesor AMD Opteron X3216 1,6 GHz / 2 jádra – 2 vlákna
- Operační paměť 8 GB RAM
- Operační systém Unraid 6.6.6

Logickému uzlu jsou přiřazeny následující zdroje:

- 2 procesorová vlákna
- Operační paměť 3 GB RAM
- Operační systém Debian 9

Ostatní zdroje serveru jsou již využity jinými virtuálními stroji. Tato konfigurace je zajímavá zejména tím, že má horší parametry, než bylo požadováno v kapitole 3.2.5. To umožní otestovat systém v nepříznivých podmínkách a může poskytnout přínosné informace pro další optimalizaci v budoucnosti.

Jako fyzické uzly byly použity dva jednodeskové počítače. Prvním je jednodeskový počítač Cubieboard2 a druhým je Orange Pi Zero. Oba počítače disponují rozhraním Ethernet RJ-45. Konfigurace jsou následující:

- Cubieboard2
  - Procesor Allwinner A20 / 2 jádra – 2 vlákna
  - Operační paměť 1 GB RAM
  - Operační systém Debian 9 – mutace Armbian
- Orange Pi Zero
  - Procesor Allwinner H2 / 4 jádra – 4 vlákna
  - Operační paměť 512 MB RAM
  - Operační systém Debian 9 – mutace Armbian

Výše uvedené jednodeskové počítače mají v testování přiřazené specifické role. Cubieboard2 zastává roli ovládacího panelu se dvěma tlačítky a dvěma kontrolkami. Orange Pi Zero je v roli aktivního prvku, který má přes rozhraní UART připojen soubor vstupních a výstupních rozhraní. Tento soubor obsahuje dvě teplotní čidla a jeden výstup pulsně-šířkové modulace (PWM).

5.2 Testovací metodika

Jednotlivé testovací scénáře byly navrhovány k otestování základní funkcionality systému a jeho celků. Byly využity převážně integrační a systémové testy. Také byl systém hodnocen z pohledu výkonnosti a vysoké dostupnosti. Cílem bylo zjistit, zda se podařilo odstranit zjištěné nedostatky uvedené v kapitole 2.3, zejména eliminovat jediný bod selhání a snížit objem přenášených dat.

5.2.1 Testování fyzického uzlu

Fyzický uzel by měl být z pohledu systému transparentní a předávat data z fyzických rozhraní přímo logickému clusteru. Vzhledem k jednoduchosti implementace fyzického uzlu zajistí otestování této funkcionality pokrytí všech zásadních částí implementace.

Testování se provádí ve dvou směrech, vstupním a výstupním. Testování ve vstupním směru spočívá v přivedení dat na fyzické rozhraní a následném sledování tématu N2S, zda se data z fyzických rozhraní propagují do Apache Kafka neporušená a kompletní, v příslušném formátu. Testování ve výstupním směru spočívá v zapsání vhodných zpráv do tématu S2N a sledování, zda se propagují na fyzická rozhraní příslušného uzlu.

5.2.2 Testování logického uzlu

Testování logického uzlu lze provést pouze velmi omezeně. Důvodem je, že logika zpracování dat přicházejících z fyzických uzlů je modulová nadstavba (viz kapitola 4.2.2.3) a nelze ji brát jako pevnou součást systému. Lze tedy otestovat pouze podpůrné části, jako jsou vlákna pro komunikaci s Apache Kafka, třídění zpráv, poskytnutí konfigurace fyzickému uzlu a načítání logických modulů.

Otestování logického uzlu je prováděno manuálně pomocí procesních testů na konfiguraci dle kapitoly 5.1.2. Logický uzel je testován proti reálnému fyzickému uzlu. Je testováno primárně komplexní chování logického uzlu, tedy poskytnutí konfigurace pro nově připojený fyzický uzel a zpracování zpráv přicházejících z fyzického uzlu. Reálný fyzický uzel v pravidelném časovém intervalu generuje data. Na výpisu konzole a v logu logického uzlu se ověřuje, zda celý proces proběhl korektně. Zároveň se kontroluje, jestli logický uzel vykonává činnosti vůdce clusteru, jelikož v testovací konfiguraci je jediným logickým uzlem.

5.2.3 Testování logického clusteru

Cílem testování logického clusteru je ověřit funkčnost částí implementace, které jsou zodpovědné za fungování víceuzlového logického clusteru. Jedná se zejména o volbu vůdce a reakce na změny v clusteru, jako například připojení dalšího logického uzlu nebo výpadek logického uzlu.

Testování je prováděno manuálně na konfiguraci uvedené v kapitole 5.1.1. Celkem jsou testovány tři situace. První situací je přidání nového uzlu do clusteru. Druhou je výpadek uzlu, který není vůdcem clusteru. Třetí situací je výpadek vůdce clusteru a následná volba nového vůdce. Ve všech případech musí dojít k zotavení clusteru a návratu k plné funkčnosti.

5.2.4 Testování komunikace mezi fyzickým uzlem a logickým clusterem

Testování komunikace spočívá v ověření správnosti konfigurace Apache Kafka clusteru. Cílem je ověřit, že je zajištěna komunikace mezi logickými uzly a logickým clusterem i v případě poruch.

Testování je prováděno manuálně na konfiguraci uvedené v kapitole 5.1.1. Během aktivní komunikace mezi fyzickým uzlem a logickým clusterem jsou jednotlivé instance Apache Kafka na logických uzlech vypínány, a to způsobem korektním (pomocí signálu SIGTERM) i nekorektním (pomocí signálu SIGKILL). Zjišťuje se, zda dojde ke ztrátě zpráv či duplikaci zpráv. Ani k jedné situaci by dojít nemělo a každá zpráva by měla být doručena právě jednou. Toto testování je prováděno pomocí generování posloupnosti dat na UART rozhraní fyzického uzlu. Tyto zprávy jsou poté na logických uzlech logovány. Takto lze ověřit, že veškeré zprávy odeslané z fyzického uzlu byly doručeny do logického clusteru.

5.2.5 Testování systému jako celku – Testovací provoz

Testovací provoz probíhá na konfiguraci dle kapitoly 5.1.2. Pro testovací provoz je vytvořen logický modul Radiator. Tento modul pomocí PWM zajišťuje řízení ventilátorů umístěných na spodní straně deskového radiátoru. Dále jsou na radiátoru dvě teplotní čidla. Jedno je umístěno na přívodní trubce radiátoru a druhé na výstupní. Čidla a ventilátory se připojují přes rozhraní UART k jednodeskovému počítači Orange Pi Zero. Dále má modul dvě kontrolky a dvě tlačítka. Kontrolky a tlačítka se připojují k jednodeskovému počítači Cubieboard2, který představuje jednoduchý ovládací panel.

Modul Radiator sleduje rozdíl mezi dvěma teplotními čidly. Pokud je tento rozdíl příliš malý a teploty dostatečně vysoké, dojde ke spuštění ventilátorů. Zároveň je hlídán stav čidel a v případě poruchy je rozsvícena příslušná kontrolka na ovládacím panelu. Ten dále umožňuje přepnutí z automatického řízení na manuální. Hodnoty z čidel jsou sbírány každou sekundu.

V rámci testovacího provozu se sleduje chování systému v reálném nasazení, a to celková výkonost v podobě přenesených dat, doba jejich přenosu na trase fyzický uzel → logický cluster → fyzický uzel a spolehlivost, resp. výpadky systému. Dále jsou během testovacího provozu v systému manuálně vytvářeny chyby (odpojení čidla, ztráta síťové konektivity, ztráta napájení atd.) a sleduje se reakce systému.

5.3 Výsledky testování

V rámci testování dle kapitol 5.2.1 až 5.2.5 (vyjma kapitoly 5.2.3) se podařilo celý systém otestovat a nebyly nalezeny žádné zásadní problémy či rozpory s návrhem. Chyby, které byly odhaleny v rámci testování, bylo možno velmi rychle opravit a byly zejména algoritmického charakteru.

Výjimku však tvoří výsledky testování logického clusteru dle kapitoly 5.2.3. Zde bylo testování navrženo pro testování clusteru o pěti logických uzlech. Tento návrh vycházel z množství dostupných zdrojů v podobě operační paměti a procesorových vláken. Během testování se ale ukázalo, že provoz pěti uzlů zároveň na jednom fyzickém stroji je velmi problematický, ve většině pokusů neproveditelný. Přestože se vytížení operační paměti pohybovalo mezi 13 GB až 14 GB z možných 16 GB, docházelo k pádům virtuálních strojů z důvodu nedostatku paměti, jejich zamrzání a výskytu neočekávaných a nereplikovatelných chyb. Při snížení počtu uzlů v logickém clusteru na čtyři se problémy již tak výrazně neprojevovaly, ale stále občas docházelo k náhodnému zamrznutí některého uzlu na dobu 1 až 3 sekund. Tím se sice podařilo otestovat chování clusteru na náhodné nekontrolované odpojování a připojování uzlů, ale nejednalo se o žádoucí chování. Nejlepších výsledků bylo dosaženo při testování clusteru se třemi uzly. V této konfiguraci se podařilo otestovat všechny scénáře bez vnějších neřízených zásahů ze strany hostitelského systému a virtualizačního nástroje.

Testovací provoz v reálném nasazení probíhá od března 2019 a pokračuje až do současnosti. Během něj je systém sledován pomocí nástrojů Wireshark a Ubiquiti DePacket Inspection. Za celou dobu nebyly zaznamenány žádné závažné problémy se systémem. Výsledky testovacího provozu je nutno porovnat s problémy uvedenými v kapitole 2.3.

Nejzávažnější problém v podobě velkého objemu přenesených dat se podařilo významně ovlivnit. V obou řešeních se jedná o přenos 6 bajtů dat každou sekundu. Odtud můžeme určit teoretické minimum 1 MB surových užitečných dat za 48 hodin. Pro porovnání, pokud by se tato data přenášela přes UDP rámce pomocí IPv4 protokolu, bude objem dat cca 5,8 MB. Původní řešení vzniklé v rámci semestrálního projektu přenese 3 GB dat za 48 hodin. Nyní se přenese pouze 0.15 GB dat za 48 hodin při stejné frekvenci a délce přenášených dat. To je způsobeno tím, že nedochází k opakovanému navazování a ukončování spojení mezi fyzickým a logickým uzlem. V současné verzi otevře fyzický uzel při svém spuštění spojení s logickým clusterem, které uzavře v případě svého ukončení. Při výpadku konektivity fyzický uzel spojení obnoví. Jedná se tedy o úsporu 95 % přeneseného objemu dat. Takto významné úspory bylo dosaženo pouze změnou způsobu komunikace bez zásahů do komunikačního protokolu. Zároveň také došlo ke zrychlení výměny zpráv mezi fyzickým uzlem a logickým clusterem (resp. uzlem), z průměrných[5] původních 120 ms na 40 ms.

Druhým velmi závažným problémem byla správa paměti. V rámci testovacího provozu nebyla zaznamenána žádná komplikace a využití paměti se na logickém uzlu pohybovalo kolem 2 GB z 3 GB možných. Jedná se také o značnou optimalizaci oproti původnímu stavu, kdy využití paměti serveru dlouhodobě rostlo až na maximum. Souhrnný přehled všech sledovaných hodnot lze vidět v tabulce 5.1 níže.

Třetím řešeným problémem byla spolehlivost systému a redundance. V původní implementaci se počítalo pouze s jediným logickým serverem, což představovalo jediný bod selhání celého systému. Řešení, navržené v této práci, jediný bod selhání eliminuje. To se vzhledem k omezeným hardwarovým prostředkům v praktickém nasazení podařilo otestovat pouze omezeně, ale dostatečně. Systém poskytuje širokou možnost redundance s ohledem na výpadky v logickém clusteru. Odstranění jediného bodu selhání se jeví jako velmi těžce splnitelný úkol. Během testování došlo k odhalení tří takových bodů, a to síťový switch, síťový router a jistič elektrického rozvodu. Tyto body selhání nelze odstranit v běžných podmínkách bez vhodné infrastruktury.

Tabulka 5.1 Porovnání řešení

	Využití RAM [GB]	Doba komunikace [ms]	Objem dat za 48 h [GB]
Původní řešení	∞	120	3,00
Nové řešení	2	40	0,15

[5] Průměrná doba výměny zprávy 120 ms byla zjištěna během testování implementace semestrálního projektu před zahájením bakalářské práce.

5.4 Návrhy úprav na základě výsledků testování

Během testování se ukázal prostor pro další úpravy, které by měly vést ke zlepšení redundance, modulárnosti a správy celého systému.

První navrhovanou úpravou je oddělení clusteru Apache Kafka (včetně Apache Zookeeper) od logického clusteru. Důvodem je, že Apache Kafka (resp. Apache Zookeeper) vyžaduje seznam všech uzlů v konfiguračním souboru. Pokud je instance Apache Kafka součástí logického uzlu, přidání dalšího logického uzlu vyžaduje úpravu konfiguračních souborů Apache Kafka na všech logických uzlech. To činí správu logického clusteru obtížnou. Vyčleněním Apache Kafka clusteru do samostatného celku lze logický cluster spravovat mnohem jednodušeji.

Další navrhovanou úpravou je implementace logických modulů, například v podobě Python skriptu, který by byl napříč uzly v logickém clusteru distribuován pomocí distribuovaného souborového systému. V současném stavu je třeba logický modul implementovat jako třídu přímo v kódu logického uzlu. To znamená, že při změně v logickém modulu je potřeba implementaci na všech logických uzlech znovu nasadit. Díky této navrhované úpravě by stačilo logický modul nasadit na jeden logický uzel, který by automaticky zajistil distribuci na další logické uzly. To by umožnilo také změny logiky bez nutnosti restartu logických uzlů.

6. Závěr

Cílem této práce bylo analyzovat možnosti komunikace v IoT sítích, navrhnout IoT síť s ohledem na efektivitu a zabezpečení přenášených dat, integraci s jinými systémy a redundanci, která by eliminovala jediný bod selhání.

V rámci analýzy v kapitole 2 byly prozkoumány a zhodnoceny komunikační protokoly a modely, které lze využít v distribuovaných systémech, kam lze zařadit i Internet věcí. Z této analýzy poté vznikl v kapitole 3 návrh, který zohledňuje požadavky a kombinuje je s již existujícími možnostmi a zároveň respektuje návrhové vzory a obecné zvyklosti při vývoji softwaru a distribuovaných systémů. Zásadní části implementace návrhu byly popsány v rámci kapitoly 4 a vzniklý systém byl poté otestován v kapitole 5. Zadání a cíle práce se tedy podařilo splnit.

Práce má velký potenciál uplatnit se v praxi, jak prokázal několik měsíců probíhající testovací provoz, který trvá až do současnosti. Vývoj tohoto systému bude pokračovat i nadále, a to zejména úpravami navrženými v kapitole 5.4. Lze očekávat, že testovací provoz plynule přejde v plné nasazení a stane se v budoucnu základem pro chytrou domácnost.

Seznam použité literatury

1. Coulouris G., Dollimore J., Kindberg T., Blair G. *Distributed Systems: Concepts and Design (5th Edition).* Boston : Addison-Wesley, 2011. 0132143011.

2. Sommerville, Ian. *SOFTWARE ENGINEERING, 9th Edition.* London : Pearson, 2011. 978-0-13-703515-1.

3. ISO/IEC 20922:2016. *Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1.* Ženeva : International Organization for Standardization, 2016.

4. Z. Shelby, K. Hartke, C. Bormann. The Constrained Application Protocol (CoAP). *RFC-7252.* Bremen : Internet Engineering Task Force, 2014. 2070-1721.

5. Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Core. *RFC-6120.* Denver : Internet Engineering Task Force, 2011. 2070-1721.

6. ISO/IEC 19464:2014. *Information technology -- Advanced Message Queuing Protocol (AMQP) v1.0 specification.* Ženeva : International Organization for Standardization, 2014.

7. I. Fette, A. Melnikov. The WebSocket Protocol. *RFC-6455.* Hampton : Internet Engineering Task Force, 2011. 2070-1721.;

8. IEC 62056-6-1:2017. *Electricity metering data exchange - The DLMS/COSEM suite - Part 6-1: Object Identification System (OBIS).* místo neznámé : International Electrotechnical Commission, 2017.

9. Shvets, Alexander. SourceMaking: Command Design Pattern. *SourceMaking.* [Online] 2007 - 2019. [Citace: 20. 04 2019.] https://sourcemaking.com/design_patterns/command.

10. LinkedBlockingQueue (Java Platform SE 8). *Java Platform Standard Edition 8 Documentation.* [Online] Oracle, 1993 - 2019. [Citace: 30. 04 2019.] https://docs.oracle.com/javase/8/docs/api/?java/util/concurrent/LinkedBlockingQueue.html.

11. Shvets, Alexander. SourceMaking: Observer Design Pattern. *SourceMaking.* [Online] 2007 - 2019. [Citace: 20. 04 2019.] https://sourcemaking.com/design_patterns/observer.

Seznam zkratek

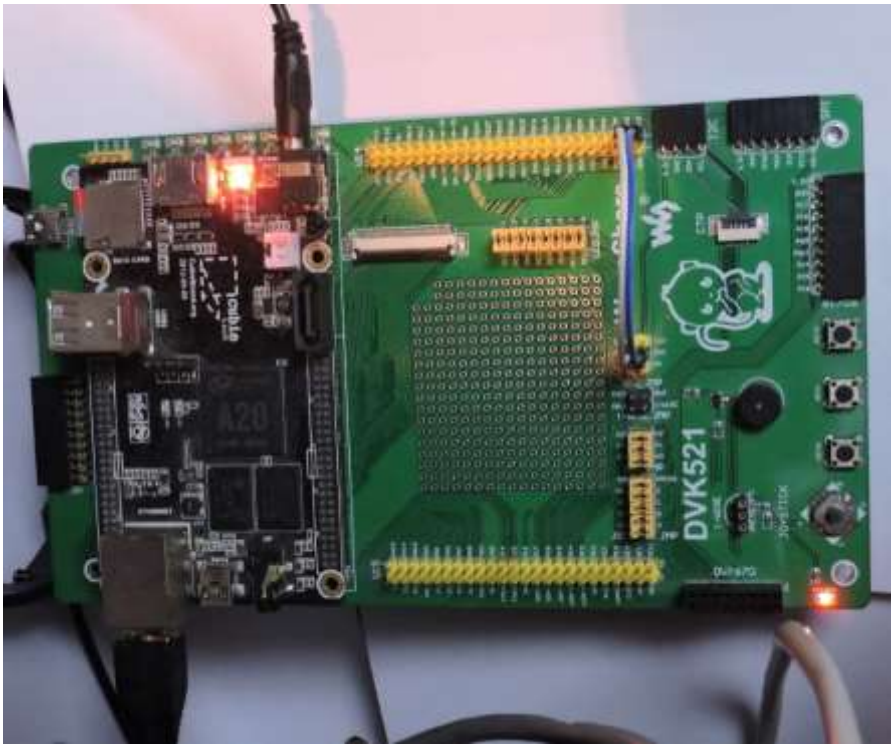
Zkratka	Význam
IoT	Internet Of Things
SPOF	Single point of failure
IFTTT	If this then that – Internetová služba https://ifttt.com/
XML	Extenible Markup Language
PaaS	Platform as a Service
MQTT	Message Queuing Telemetry Transport
M2M	Machine to machine
TCP	Transmission Control Protocol
JSON	JavaScript Object Notation
CoAP	Constrained Application Protocol
HTTP / HTTPS	Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure
XMPP	Extensible Messaging and Presence Protocol
AMPQ	Advanced Message Queuing Protocol
REST	Representational State Transfer
RFC	Request For Comment
NAT	Network Address Translation
N2S	Node to Server – Apache Kafka téma
S2N	Server to Node – Apache Kafka téma
S2S	Server to Server – Apache Kafka téma
GPIO	General Purpose Input Output
USART / UART	Universal Synchronous / Asynchronous Receiver and Transmitter
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
TWI	Two Wire Interface
RAM	Random Access Memory
DLMS/COSEM	Device Language Message Specification / Companion Specification for Energy Metering
USB	Universal Seriál Bus
SSL / TLS	Secure Sockets Layer / Transport Layer Security
ACL	Access Control Lists
JNI	Java Native Interface
PWM	Pulse Width Modulation

Seznam obrázků

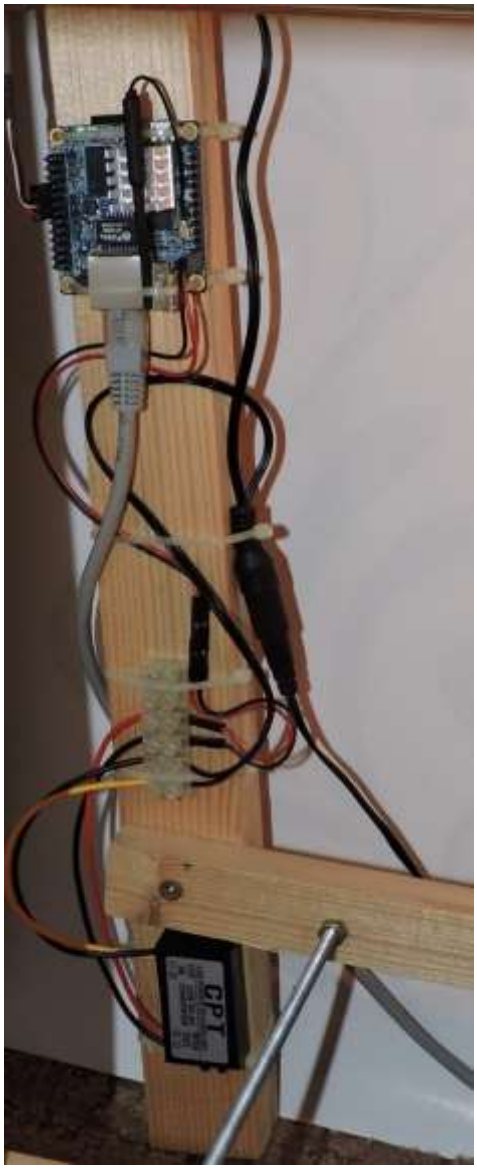
Seznam tabulek

Příloha A: Fotky reálného nasazení

Fotografie A: Server – Logický uzel



Fotografie B: Fyzický uzel – Kontrolní panel



Fotografie C: Fyzický uzel – Aktivní člen

\+

Fotografie D: Soubor čidel

Příloha B: Seznam elektronických příloh

Součástí práce je elektronická příloha ve formě zip souboru.

- 3rd_party_conf – ukázky konfiguračních souborů pro Apache Kafka a Galera cluster
- physical_node – implementace fyzického uzlu
    - src/main/java - zdrojové kódy
        - board – třídy pro řízení fyzických rozhraní
        - config – konfigurace uzlu
        - exceptions – výjimky
        - exchange – třídy pro komunikaci s logickým uzlem, resp. clusterem
        - main – hlavní vlákno fyzického uzlu
        - support – třídy pro podporu běhu programu
    - pom.xml – maven soubor pro sestavení projektu
- logical_node – implementace logického uzlu
    - src/main/java/iot/system/iotserver – zdrojové kódy
        - config – konfigurační třídy
        - enums – výčtové typy
        - json – implementace převodníků java objektů do json formátu a vice vrsa
        - message – třídy pro komunikaci mezi jednotlivými uzly
        - model – datový model pro databázi
        - modules – logické moduly
        - repository – repozitáře pro komunikaci s databází
        - rest – rest kontroléry pro přístup k informacím o logickém uzlu
        - runners – úlohy pravidelně prováděné na pozadí
        - services – konfigurace služeb
        - support – podpůrné třídy