



F3

**Fakulta elektrotechnická
Katedra měření**

Bakalářská práce

Distribuovaný systém IoT zařízení řešící problém konsenzu

Petr Kučera
Otevřená informatika
Internet věcí

Květen 2022, leden 2023
<https://github.com/petrkucera/rafting-button>
Vedoucí práce: doc. Ing. Jiří Novák, Ph.D.

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kučera** Jméno: **Petr** Osobní číslo: **499325**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra měření**
Studijní program: **Otevřená informatika**
Specializace: **Internet věcí**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Distribuovaný systém IoT zařízení řešící problém konsenzu

Název bakalářské práce anglicky:

A distributed system of IoT devices solving the consensus problem

Pokyny pro vypracování:

1. Definujte funkční a další požadavky na systém
2. Navrhněte možná algoritmická řešení, diskutujte jejich přednosti a nedostatky
3. Zvolte optimální variantu a volbu zdůvodněte
4. Na základě předchozího kroku zvolte vhodné technické řešení, systém navrhněte a realizujte.
5. Implementujte programové vybavení
6. Otestujte funkce komponent i celého systému

Seznam doporučené literatury:

- [1] ONGARO, Diego a John OUSTERHOUT. In Search of an Understandable Consensus Algorithm: Extended Version, [online]. May 20, 2014 [cit. 2023-02-14]. Dostupné z: <https://raft.github.io/raft.pdf>
[2] D. Yang, I. Doh and K. Chae, 'Cell Based Raft Algorithm for Optimized Consensus Process on Blockchain in Smart Data Market,' in IEEE Access, vol. 10, pp. 85199-85212, 2022, doi: 10.1109/ACCESS.2022.3197758.
[3] 'IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,' in IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008) , vol., no., pp.1-499, 16 June 2020, doi: 10.1109/IEEESTD.2020.9120376.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jiří Novák, Ph.D. katedra měření FEL

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

Datum zadání bakalářské práce: **16.02.2023**

Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce:
do konce letního semestru 2023/2024

doc. Ing. Jiří Novák, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu mé bakalářské práce doc. Ing. Jiřímu Novákovi, Ph.D. za trpělivost a veškerý čas, který do mne investoval. Svému bratrovi Jakubovi za pomoc při měření vlastnosti protokolu síťové infrastruktury v prostředí divoké přírody a parádní fotografie. Rád bych také poděkoval své rodině, kamarádům a spolubydlícím za podporu během mého studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 26. 5. 2023

.....

Abstrakt / Abstract

Tato práce se zabývá problematikou konsenzu v distribuovaných IoT sítích. Konkrétně si klade za cíl vytvořit koncept zařízení, které se bude schopné uspořádat jednotlivé události podle pořadí tak, aby byla zachována jejich kauzalita.

Klíčová slova: distribuovaný systém, konsezus, IoT, ESP-NOW, ESP32, synchronizace času

This work deals with the issue of consensus in distributed IoT networks. The goal is to create a concept for a device that will be able to arrange individual events according to order in such a way that their causality is preserved.

Keywords: distributed system, consensus, IoT, ESP-NOW, ESP32, time synchronization

Title translation: A distributed system of IoT devices solving the consensus problem

Obsah /

1 Úvod	1
1.1 Motivace	1
1.2 Popis cíle bakalářské práce	1
1.3 Popis hlasovacího zařízení	1
1.4 Stanovení cílů a plánů	2
2 Rešerše	3
2.1 Vymezení základních pojmu	3
2.1.1 Internet věcí	3
2.1.2 M2M komunikace	3
2.1.3 Industry 4.0	3
2.1.4 Wireless communication	3
2.2 Trendy ve světě IoT	4
2.2.1 Problematika spotřeby energie při bezdrátové komunikaci	4
2.2.2 Low-Power Wifi	7
2.2.3 Optimalizace nízké spotřeby na zařízení	8
2.3 Struktura sítě	8
3 Specifikace funkčních a dalších požadavků na systém	10
3.1 Funkční požadavky na systém	10
3.2 Popis užití systému v praxi	11
4 Síťová infrastruktura	12
4.1 Volba vhodného modulu	12
4.2 Požadavky na síťovou infrastrukturu	13
4.3 Protokol ESP-NOW	13
4.3.1 Popis technologie	14
4.3.2 Formát rámce	14
4.3.3 Módy provozu wifi	15
4.3.4 Popis fungování	15
4.3.5 Limity technologie	16
4.3.6 ESP-NOW pro Arduino framework	17
4.4 Architektura síťové infrastruktury	17
4.5 Parametry síťové infrastruktury	18
5 Algoritmus	19
5.1 Požadavky na algoritmus	19
5.1.1 Obecné požadavky na distribuovaný systém	19
5.2 Představení známých algoritmů	20
5.2.1 Kauzalita a čas	20
5.2.2 Konsenzus	21
5.3 Rozbor problému	25
5.4 Problematika synchronizace času	26
5.4.1 Detailní popis zvoleného algoritmu	26
5.4.2 Simulace fungování algoritmu	28
5.4.3 Získání parametrů pro simulaci	29
5.4.4 Simulace s reálnými parametry	30
5.4.5 Měření přesnosti synchronizace času	32
5.5 Distribuce logů a seznamu zařízení DS	33
5.6 Návrh celkového algoritmu	34
5.6.1 Registrace zařízení	35
5.6.2 Běžný chod	35
5.6.3 Terminace zařízení	35
6 Zařízení z pohledu UX	36
7 Implementace a testování	38
7.1 Implementace	38
7.1.1 Struktura rámce zprávy	39
7.1.2 Proces zpracování zpráv	40
7.1.3 Poznámky k implementaci	41
7.2 Testování	41
8 Závěr	42
Literatura	43
A Slovníček pojmu	47
B Detailní výsledky měření síťové infrastruktury	48
B.1 Scénář A	48
B.2 Scénář C	50
B.3 Scénář D	52
C Schéma modulů	54
D Měření latence	59
E Synchronizace času	62
E.1 Simulace	62
E.2 Měření	64

**F Slovní komentáře průzku-
mu podoby zařízení**

68

Tabulky / Obrázky

2.1 Orientační hodnoty pro porovnání režimů činností dle spotřeby energie	4
3.1 Možné stavy zařízení	10
4.1 Srovnání rozdílných parametrů modulů	12
5.1 Parametry měření latence	30
7.1 Priority procesů	39
B.1 Přehled parametrů pro A1x scénáře	48
B.2 Přehled parametrů pro A2x scénáře	49
B.3 Přehled parametrů pro Cx scénáře	51
B.4 Přehled parametrů pro D scénář	52
2.1 Srovnání parametrů LPWAN technologií	6
2.2 Vizualizace fungování BSS barvení ve Wifi-6	8
2.3 Vizualizace typické (hierarchické) struktury IoT projektu ..	9
4.1 ESP-NOW Action Frame	14
4.2 ESP-NOW Content Frame	15
4.3 Schéma odesílání dat pomocí ESP-NOW	16
4.4 Workflow dat v síťové infrastruktuře	18
5.1 Stavový diagram změny rolí při volbě nového lídra	23
5.2 Schéma synchronizace doby přenosu	26
5.3 Schéma synchronizace času	27
5.4 Schéma odesílání zpráv v simulaci	29
5.5 Simulace fungování algoritmu pro synchronizaci času - rozdíl čas	31
5.6 Měření synchronizace času - rozdíl času	33
5.7 Komunikační náročnost při distribuci logů	34
6.1 Ilustrační obrázky obalu vygenerované modelem AI	36
6.2 Graf výsledků z průzkumu podoby zařízení	37
7.1 Struktura rámce zprávy	40
7.2 Struktura rámce zprávy - sousedé	40
B.1 Vizualizace parametrů pro měření scénářů A	48
B.2 Graf výsledků měření scénářů A1	49
B.3 Graf výsledků měření scénářů A2	50
B.4 Vizualizace parametrů pro měření scénářů C	50
B.5 Graf výsledků měření scénářů C	51
B.6 Vizualizace parametrů pro měření scénářů D	52

B.7	Graf výsledků měření scénáře D	53
C.8	Rozměry ESP32-S2-Pico	54
C.9	Schéma ESP32-S2-Pico	55
C.10	Schéma rozložení ESP32-S2- Pico	56
C.11	Schéma pinu ESP32-S2-Pico... ...	57
C.12	Schéma pinu ESP32-S2-LCD ..	58
D.13	Zapojení obvodu pro měření latence	59
D.14	Schéma zapojení obvodu pro měření latence	60
D.15	Výsledky měření latence při scénáři A	60
D.16	Výsledky měření latence při scénáři B.....	61
E.17	Simulace fungování algorit- mu pro synchronizaci času - chyba	62
E.18	Simulace fungování algorit- mu pro synchronizaci času - doba přenosu	63
E.19	Simulace fungování algorit- mu pro synchronizaci času - rozdíl čas	63
E.20	Průběh měření synchronizace času	64
E.21	Měření synchronizace času - schéma	64
E.22	Měření synchronizace času - zapojení.....	65
E.23	Měření synchronizace času - chyba	66
E.24	Měření synchronizace času - doba přenosu	66
E.25	Měření synchronizace času - rozdíl času	67

Kapitola 1

Úvod

V první kapitole vysvětlují motivace, které mě vedly k volbě daného téma. Specifikují cíl bakalářské práce a vysvětlují cíl v širší souvislosti. V závěru definují jednotlivé kroky, podle kterých budu během práce postupovat.

1.1 Motivace

Už od útlých dětských let jezdím na letní tábory a přes rok organizuji mládežnické akce. Na Hudebním týdnu, jedné z akcí pro mladé, tradičně hráváme hru, ve které jsou různé týmy, které mezi sebou soutěží. Cílem je se jako první přihlásit o slovo a zodpovědět otázku. Doposud se nám nepovedlo vymyslet efektivní způsob, jak určit pořadí týmů. Ve výsledku vždy použijeme zvoneček, který při odpovědí více týmů ve stejný čas není vhodným řešením.

1.2 Popis cíle bakalářské práce

Cílem bakalářské práce je vytvořit koncept hlasovacího tlačítka, které bude fungovat jako autonomní zařízenídistribuovaném systému. Systém pomocí vhodných algoritmů bude řešit problematiku konsenzu, respektive určení pořadí stisku tlačítek.

1.3 Popis hlasovacího zařízení

Aby bylo jasné porozumět cíli, ke kterému koncept hlasovacího tlačítka směruje, domnívám se, že je důležité si popsat ideální představu výsledného zařízení.¹

Z uživatelského pohledu by zařízení mělo být kompaktní krabička, kterou si bude moci skupina položit před sebe na stůl.² Krabička by na sobě měla mít tlačítko, které bude odolné i vůči veliké síle způsobené bouchnutízápalu hry. Zařízení bude možné přepínat mezi dvěma módy, tj. *PRESENTER* a *NORMAL*. V módu *NORMAL* bude zařízení řešit, jakém pořadí se týmy přihlásily o slovo. V módu *PRESENTER* navíc pomocí webového serveru vykreslí výsledky.

Samotná hra pak bude probíhat tak, že bude položena otázka, hráči se přihlásí o slovo stiskem tlačítka. Informace se rozdistribuuje a zařízenímodu *PRESENTER* zobrazí výsledky. Hlasování se vyresetsuje dlouhým stisknutím hlasovacího tlačítka na jakémkoliv zařízení.

¹ Následující specifikace tedy zahrnuje i požadavky přesahující tuto bakalářskou práci.

² Koncept odolné krabičky pro celou skupinu je dle výsledků drobné ankety popsané kapitole 6 více žádaný než hlasovací tlačítko, které budou uživatelé moci držet ruce.

1.4 Stanovení cílů a plánů

Problém řešený této bakalářské práci je poměrně komplexní, proto jsem se ho rozhodl rozložit na následující dílčí úkoly, podle kterých jsem také postupoval.

- Provedte rešerši využití Wifi technologie pro IoT aplikace.
- Definujte funkční a další požadavky na systém a zvolte vhodné hardwarové řešení.
- Navrhněte síťovou infrastrukturu řešení, zjistěte její parametry a ty následně experimentálně ověřte.
- Navrhněte algoritmické řešení, které následně experimentálně ověříte.
- Realizujte řešení na reálném hardwaru.

Kapitola 2

Rešerše

Druhá kapitola se věnuje trendům technologií aplikovaných v IoT řešení. Popisuje největší výzvy a nejčastější problémy, se kterými se v praxi setkáváme. Dále se zaměřuje na hierarchickou strukturu IoT sítě.

2.1 Vymezení základních pojmu

Než přejdeme k samotné diskuzi problematiky, domnívám se, že je důležité v rychlosti definovat základní pojmy jako je IoT či Industry 4.0 v kontextu tohoto projektu.

2.1.1 Internet věcí

Pojem **internet věcí** (*Internet of Things* či zkráceně IoT) označuje síť fyzických zařízení. Každé zařízení získává data pomocí senzorů nebo zpracovává již naměřená data či ovládá akční člen (např. DC motor). Zařízení si mezi sebou vyměňují informace skrze komunikační síť, jako je např. *Internet*. [1–3]

2.1.2 M2M komunikace

Pojem **Machine to Machine** komunikace (M2M) označuje bezdrátovou i drátovou komunikaci mezi dvěma a více zařízeními napřímo bez jakéhokoliv zásahu člověka. *M2M* je součástí *IoT*. [4]

2.1.3 Industry 4.0

Neexistuje žádná přesná definice pro **Průmysl 4.0** (*Industry 4.0*), která by dle mého názoru objektivně popisovala vše, co se pod tímto pojmem skrývá. Jedná se o stále probíhající období, na které se nemůžeme dívat s odstupem. Obecně můžeme tvrdit, že při specifikování průmyslu čtvrté generace, se definice shodují v hlavních rysech [5–6] popisujících tuto *evoluci*:¹

- **Aditivní výroba** (opak obrábění, předmět vzniká přidáváním materiálu²),
- **Digitální transformace** (smíšená realita, digitální dvojčata atd.),
- **IoT** (síť fyzických zařízení spojující stroje, lidi a věci),
- **Automatizace** s prvky **AI** a **prediktivní analýzy** na základě naměřených dat,
- **Cloud Computing** umožňující horizontální a vertikální škálování,
- **Kybernetická bezpečnost** systémů.

2.1.4 Wireless communication

Pojem **bezdrátová komunikace** (*Wireless communication*) označuje přesun informace z místa A do místa B jinak než pomocí elektrických či optických vláken. Typicky se jedná o vlnění přenášené prostorem. Jméno a typ vlnění závisí na konkrétních parametrech. Nejčastěji se jedná o *rádiové vlny*. [7]

¹ Pojem evoluce jsem použil schválně. Chtěl jsem tak poukázat na to, že z našeho pohledu současné situace se opravdu nejedná o revoluci nýbrž průběžný a postupný rozvoj technologií.

² Více informací na: https://cs.wikipedia.org/wiki/Aditivní_výrobní_proces

2.2 Trendy ve světě IoT

Zavádění IoT zařízení do denního osobního života a fungování firem přináší výhody jako je zvýšení efektivity, produktivity a nutnosti nedělat rutinní věci. Trendem je sbírat co největší množství dat, aby bylo možné data efektivně vytěžit. S tímto trendem souvisí zvyšování množství IoT zařízení, jako jsou například jednoduché MCU se senzorem pro měření jedné veličiny a modulem pro bezdrátovou komunikaci.

Takovéto zařízení typicky funguje tak, že se jednou za určitou dobu vzbudí. Pomocí senzoru naměří data, která zpracuje do podoby, aby byla interpretovatelná³ a data uloží. Následně se data hned odešlou nebo pokračuje zařízení pokračuje ve spánku a po naměření a elementárním zpracování většího množství dat, dojde k odeslání do hierarchicky nadřazeného zařízení.⁴ K odeslání se obvykle využívá bezdrátového spojení. Po odeslání dat zařízení opět usne a celý cyklus se opakuje.

Typicky jsou takováto zařízení drobná, lze je umístit na jakékoli místo. Jsou optimalizována pro minimální spotřebu energie a jsou napájena z baterie. Díky tomu nezávisí na externím zdroji energie.

Aby mohl být popsáný provoz realizován, je nutné všechny vrstvy maximálně optimalizovat pro úsporu energie. Nejvíce dominantní při celém životním cyklu je ve většině případů vysílání zprávy. Při ní je nutné využívat energii. Proto byly navrženy speciální technologické postupy, které pomáhají snížit spotřebu.

Pro základní představu je v tabulce 2.1 specifikována orientační hodnota spotřeby proudu.⁵

REŽIM ČINNOSTI	SPOTŘEBA
aktivní provoz (měření, ukládání do paměti, ...)	$5mA$
režim vysílání (odesílání, případně příjem dat)	$0.045A$
režim spánku (čekání vzbuzení)	$1\mu A$

Tabulka 2.1. Orientační hodnoty pro porovnání režimů činností dle spotřeby energie.

Je možné, že pokrok technologií přinese řešení napájení. Jedna z možných cest je, že MCU bude odebírat potřebnou energii z okolních zdrojů využívající elektromagnetickou energii.⁶ V současnosti není zvykem takováto zařízení využívat ani se nedomnívám, že se jedná o univerzálně využitelné řešení. To především kvůli tomu, že IoT zařízení jsou umístěna i na místech, kde se nepředpokládá přítomnost dostatečně silného signálu.

2.2.1 Problematika spotřeby energie při bezdrátové komunikaci

Jak bylo výše popsáno, největší energetickou spotřebu na činnosti IoT zařízení má ve většině případů bezdrátová komunikace, konkrétně vysílání. Při hledání řešení, jak vybudovat síťovou infrastrukturu pro samostatný projekt, jsem se setkal s řešením této problematiky. Pro moji aplikaci není v počáteční fázi nikterak kritická. Rozhodl jsem se ale prozkoumat základní technologie, které jsou v dané oblasti k dispozici. V podstatě mapují technologie používané pro bezdrátovou komunikaci.

³ Zpracováním dat je myšleno interpretovat data tak, aby dávala smysl i mimo kontext senzoru. Například při měření vzdálenosti pomocí vířivých proudu je třeba naměřené napětí vyčistit *lock-in detekcí* a kalibrovat, aby vzdálenost byla v jednotkách délky.

⁴ Gateway či router, které data odesírají dále nebo koncentrátor. Který data ukládá a dále zpracovává.

⁵ Hodnota proudu není dynamicky specifikována v závislosti na čase činnosti. Pro stanovení přesné hodnoty pro spotřebu po dobu n s, by bylo nutné hodnotu vynásobit n , kde $[n] = As$.

⁶ <https://cdr.cz/clanek/intel-ukazal-iot-cip-ktery-si-vystaci-s-energiemi-z-radiovych-vln>

Podle požadavků fungování IoT zařízení je vhodné volit danou technologii. Pokud budou odesílaná data v desítkách až stovkách bajtů a zařízení bude nuceno fungovat na větší vzdálenost, pravděpodobně zvolíme technologie označované jako **Low Power Wide Area Network (LPWAN)**. Mezi ně patří například LoRaWAN⁷, Sigfox⁸ nebo NB-IoT.

LoRaWAN je komunikační protokol a systémová architektura postavená na technologii LoRa. Zkratka LoRa pochází z anglického *long range*, v překladu tedy dlouhý dosah. [8] LoRa používá modulaci CSS (*chirp spread spectrum*), která rozprostře signál po celém vysílacím kanálu s lineární změnou kmitočtu. LoRa funguje v pásmech ISM 433, 868 a 915 MHz. LoRaWAN umožňuje tři módy provozu, které se liší podle energetické náročnosti a počtu přenesených dat. [9]

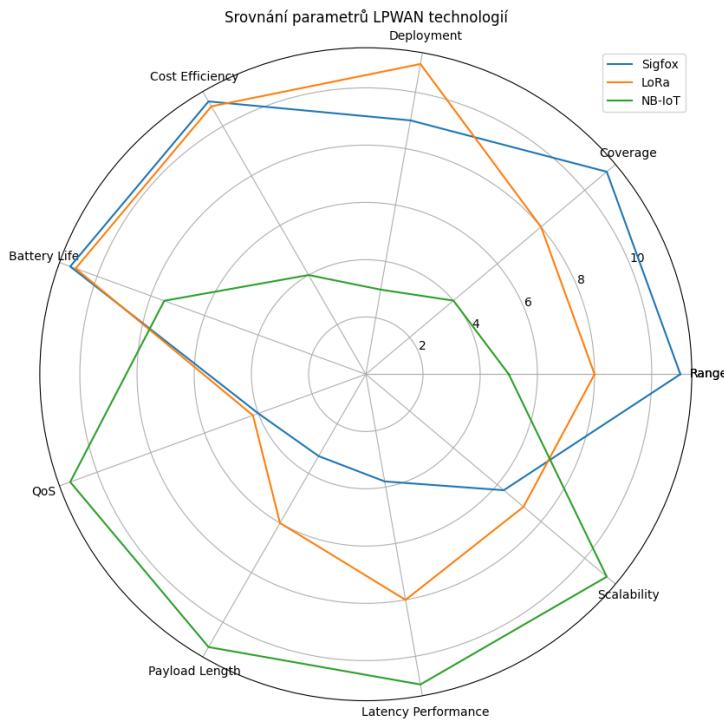
Sigfox je francouzská společnost, která se věnuje komunikacím s nízkou spotřebou energie. Poskytují službu Sigfox, která podobně jako LoRaWAN slouží především k sbírání dat z IoT zařízení. Sigfox se v praxi používá například v elektroměrech či chytrých pračkách, které jednou za čas odesílají malé množství dat. [10] Na rozdíl od technologie LoRa používá UNB (*Ultra Narrow-Band*). Řešení Sigfox se liší také v tom, že neumožňuje data odesílat do vlastní infrastruktury, nýbrž je třeba využívat unifikovaný cloud přímo od společnosti. [11]

Narrowband IoT (NB-IoT) je technika, kdy úzkopásmová síť využívá již vybudované síťové infrastruktury poskytovatelů mobilní signálu. Pásma, ve kterém NB-IoT může fungovat, je široké 200 kHz. [12] Přenosová rychlosť se liší dle generace komunikace.⁹ Trendu užití již vybudovaných IoT sítí se budu věnovat v další kapitole. [13]

⁷ Webové stránky organizace starající se o udržování LoRaWAN <https://lora-alliance.org/>.

⁸ Webové stránky společnosti Sigfox: <https://www.sigfox.com/>

⁹ Detailní přehled jednotlivých typů NB-IoT s informací o přenosové rychlosti: https://en.wikipedia.org/wiki/Narrowband_IoT#3GPP_LPWAN_standards.



Obrázek 2.1. Srovnání parametrů LPWAN technologií. [14]

Technologie Sigfox vyniká dosahem, pokrytím a nízkou energetickou náročností. LoRa je stejně jako Sigfox vhodná pro aplikace s požadavkem na extrémně nízkou energetickou náročnost. Navíc dokáže odeslat více zpráv. NB-IoT nevyniká v dosahu. Zvládne ale dosahovat mnohem větších přenosových rychlostí, kapacit a jeho latence je ze všech dosud zmiňovaných technologií, nejnižší. [14]

Pokud jsou požadavky na IoT technologii takové, že vyžadujeme vysoce energeticky nenáročnou komunikaci, která je na malou vzdálenost, připadají v úvahu především technologie jako Bluetooth, ZigBee či wifi, které jsou definovány v rodině standardů IEEE 802. [15] Analogicky podél LPWAN bychom skupinu takových zařízení mohli pojmenovávat jako **Low Power Personal Area Network (LPPAN)**.

Fyzická vrstva **ZigBee** je popsána standardem IEEE 802.15.4. Jedná se o bezdrátovou komunikaci, která funguje do dosahu 75 m na frekvencích ISM 868 MHz, 902 - 928 MHz a 2,4 GHz. Pro směrování používá multiskokové *ad-hoc* směrovací metody. Díky tomu zvládá dosahovat velkých vzdáleností. Na rozdíl od Bluetooth jsou jeho typickou aplikací průmyslové senzorové sítě. [16]

Technologie **Bluetooth** původně měla nahradit drátové sériové rozhraní RS-232. Je definována standardem IEEE 802.15.1. Pracuje v ISM pásmu na frekvenci 2,4 GHz. Nejnovější verze Bluetooth přenáší data rychlostí až 2 Mbit/s na vzdálenost 200 m a velikost zprávy může být až 255 bajtů. Bluetooth se těší veliké oblibě pro komunikaci s drobnou elektronikou, jako jsou například bezdrátová sluchátka, domácí spotřebiče či elektrické hudební nástroje, u kterých umožňuje provádět konfiguraci. [17]

Wifi technologie je skupina bezdrátových protokolů, která vychází ze standardu IEEE 802.11. V počátcích svého vývoje by se dala nazvat jako dvojče IEEE 802.3 (Ethernet). V průběhu času nové verze wifi přinášejí novinky, které v Ethernetu nejsou a z podstaty

ani nikdy nebudou. Wifi funguje v pásmu ISM 2,4 GHz a 5 GHz.¹⁰[18] Široká veřejnost zná wifi jako nástroj, který slouží pro komunikaci s Internetem. V praxi má ale mnohem širší využití.

Zajímavou demonstrací postupných technologických novinek je vývoj wifi jako technologie v čase. [19] Jednotlivé verze ukazují na trendy jako je zvyšování výpočetního výkonu a požadavků na propustnost sítě. Možnosti protokolů a vznik speciálních nových protokolů, například i s IoT technologiemi.¹¹ [18]

Ze všech tří vyjmenovaných technologií rozsahu osobní sítě (PAN) se wifi těší největší oblibě. Dle mého názoru je tato skutečnost způsobena především univerzálností, která pramení z:

- rozšíření wifi sítě (77% domácností v rámci svého domu nebo bytu rozvádějí wifi pomocí *routeru* nebo *modemu*¹²),
- a vysokými rychlostmi přenosu a nízkou latencí.

Jak uvádí autoři v úvodní části [21], výzva je vybudovat robustní ale zároveň energeticky nenáročnou síť. Takovouto aplikaci technologie wifi označujeme pojmem **Low-Power Wifi**.

■ 2.2.2 Low-Power Wifi

Pokud se podíváme na vývoj verzí *IEEE 802.11*, je si vhodné položit otázku, jaká verze je pro IoT aplikace nejvíce vhodná. Dle článku [22] je *802.11n* (Wifi-4) vhodnejší pro aplikaci v IoT zařízení než *802.11ac* (Wifi-5). Verze *802.11n* totiž nabízí 2 pásmo pro komunikaci (2,4 GHz a 5 GHz). Oproti tomu pozdější verze *802.11ac* pouze jeden kanál na 5 GHz. Pro IoT aplikace je užitečnější 2,4 GHz, protože signál lépe proniká materiálem a má větší dosah. Navíc verze *802.11ac* má vyšší energetickou náročnost, která je způsobena komplikovanějším algoritmem sloužícím k zrychlení přenosu, které je pro většinu IoT aplikací nepotřebné.

Nově do hry také přichází nejnovější verze **Wifi-6** definovaná ve standardu *802.11ax*. Slibuje hned několik novinek, které by měly IoT zařízením rapidně pomoci.

Nejčastější překážka, kterou v této práci zmiňuji, je úspora energie. Wifi-6 implementuje **Target Wait Time (TWT)** mechanismus, který umožnuje IoT zařízení při nečinnosti usnout, a tak ušetřit značné množství energie (viz 2.1).

Druhou novinkou je **OFDMA**,¹³ který umožňuje odesílat stejná data do více zařízení ve stejný čas.

Poslední novinkou, na kterou bych rád upozornil, je **BSS coloring**.¹⁴ Novinka umožňuje *obarvit* signál z různých *přístupových bodů*, a tak předejít vzájemnému rušení signálů. [23]

¹⁰ Od Wifi-6 i v 6 GHz.

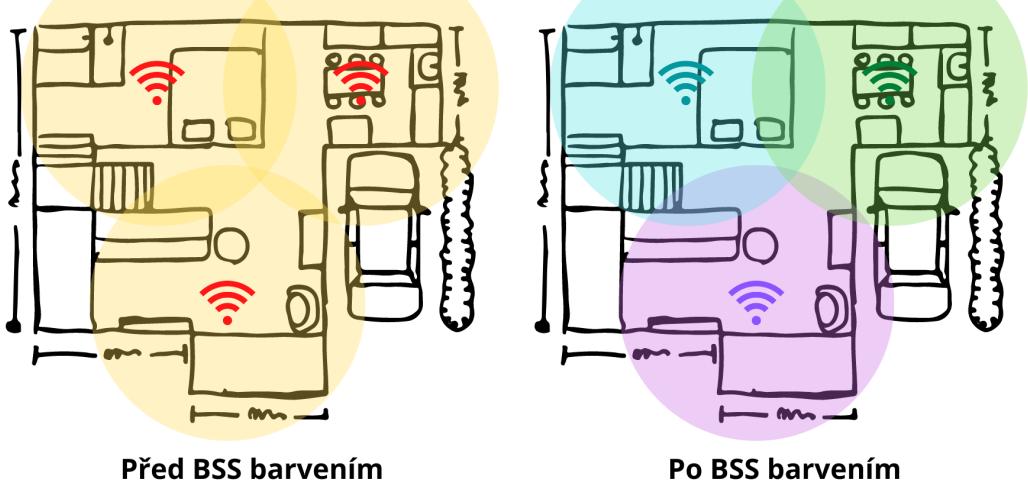
¹¹ Rychlý průlet verzemi wifi technologie je například na webové stránce: <https://www.pcmag.com/encyclopedia/term/80211-versions>.

¹² Hodnoty v ČR k roku 2022 [20]

¹³ Orthogonal frequency-division multiple access

¹⁴ Basic Service Set coloring

BSS coloring



Obrázek 2.2. Vizualizace fungování **BSS barvení** ve Wifi-6.

2.2.3 Optimalizace nízké spotřeby na zařízení

Energii pro delší fungování zařízení lze především ušetřit správnou volbou MCU a vhodnou konfigurací. Existuje několik pravidel, které mohou pomoci snížit spotřebu energie a umožnit IoT zařízení fungovat lépe:

- volba vhodného modulu či MCU,
 - uvedení zařízení do *deep sleep módu*,¹⁵
 - vypínání wifi při neaktivitě,
 - vhodná optimalizace kódu.

Na první pohled by se mohlo zdát, že se vyplatí použít všechna pravidla. To ale nemusí být vždy nejvhodnější řešení. Příkladem může být experiment Přeučila a Novotného, který demonstroval kombinaci vypínání wifi a přecházení do *deep sleep* módu. [24]

2.3 Struktura sítě

Pro IoT řešení je typická **hierarchická struktura sítě**. Tato povaha je dána účelem řešení, nejtypičtěji vytěžování dat.

- embedded zařízení, senzory, akční členy
 - koncentrátor, routery a gateway
 - server, cloud computing

První úroveň se stará o měření dat pomocí senzorů, která jsou zpracovávána v embedded zařízeních nebo o vykonávání činností na základě přijatých dat.¹⁶ V IoT scénářích se ale častěji jedná o měření. Tato úroveň následně odesílá informace do hierarchicky nadřazené struktury.

¹⁵ Procesor přejde do tzv. *deep sleep* módu a vzbudí se až při zachycení interruptu.

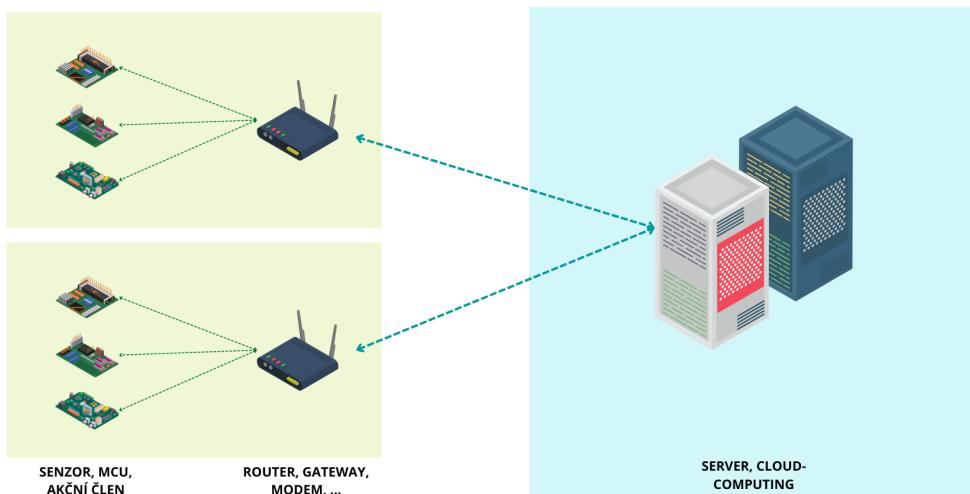
¹⁶ Například pomocí generovaného PWM signálu rozsvícení adresovatelné LED diody

Druhá úroveň sbírá data z více embeded zařízení a zprostředkovává odeslání do úrovně třetí. Data mohou být na této úrovni i předzpracována. Typická zařízení pro tuto úroveň jsou router, koncentrátor, které bývají vstupní bránou (gateway) do třetí úrovně.

Na **třetí úrovni** typicky běží server, který může být onpremisového či cloudového charakteru. Výhodou onpremisu je to, že mám zařízení u sebe. To může přinášet ekonomickou či bezpečnostní výhodu. Výhodami cloudu je naopak možné *horizontálního* a *vertikální škálování*¹⁷ či *Pay As You Go*¹⁸ subskripce. Server data zpracovává, ukládá a připravuje vizualizace pomocí nástrojů, jako jsou například open-source *Kibana*¹⁹ a *Grafana*.²⁰ Kromě vizualizace jsou data na této úrovni analyzována pomocí algoritmů. Mezi ně může patřit i strojové učení. Cílem je přeměnit data na znalost, aby je bylo možné interpretovat.

Data jsou přenášena skrz strukturu oběma směry. Pro IoT je ve většině případech spíše vytěžování směr nahoru, respektive z embeded zařízení na server.

Tradiční struktura IoT řešení



Obrázek 2.3. Vizualizace typické (hierarchické) struktury IoT projektu.

¹⁷ Horizontální škálování znamená přidávání nových instancí. Vertikálním se myslí navýšování a snižování výkonu.

¹⁸ Zaplat, jen co využiješ.

¹⁹ <https://www.elastic.co/kibana/>

²⁰ <https://grafana.com/>

Kapitola 3

Specifikace funkčních a dalších požadavků na systém

Kapitola navazuje na popis finálního řešení v kapitole 1.3, specifikuje další požadavky a zdůvodňuje jednotlivé stanové parametry.

Ne všechny funkční požadavky jsou v této bakalářské práci realizované. Pro kompletní pochopení zamýšleného konceptu se domnívám, že je nutné je zde specifikovat.

3.1 Funkční požadavky na systém

Výsledný systém by měl sloužit k **určení pořadí** toho, jak se jednotlivé týmy přihlásily o slovo, respektive seřazení událostí stisku tlačítka na jednotlivých zařízení v systému. Zařízení by měla být kompaktní, ale zároveň odolná, a to i proti silným nárazům. Měla by fungovat minimálně 6 hodin v kuse.¹

Systém bude tvořen maximálně deseti zařízeními, a to z důvodu toho, že při podobných typech her nebývá více než 6 týmů. Měl by tedy limit deseti vyhovovat.

Zařízení by měla fungovat ve dvou módech: *PRESENTER* a *NORMAL*. V módu *NORMAL* funguje normálně, tedy řeší určení pořadí. V módu *PRESENTER* navíc vykresluje výsledky, a to pomocí webového serveru, na které se může uživatel připojit pomocí otevřeného *wifi access pointu*.

Stav hlasování bude signalizován LED diodami, přičemž může být ve dvou stavech *ACTIVE* a *USED*. První možnost znamená, že v daném kole se tým ještě nepřihlásil o slovo. Druhá, že se již o slovo přihlásil.

mód zařízení	stav hlasování
<i>PRESENTER</i>	<i>ACTIVE</i>
<i>NORMAL</i>	<i>USED</i>

Tabulka 3.1. Možné stavы zařízení.

Zařízení se ovládají pomocí hlasovacího tlačítka, které by mělo být odolné i vůči silným nárazům.²

Zařízení v systému budou komunikovat pomocí krátkých zpráv.³

Požadavky na přesnost rozlišení jsou takové, aby bylo možné zaručit správnost pořadí.⁴

¹ Minimální doba 6 hodin byla stanovena na základě úvahy možných aplikací. Pokud by měl systém sloužit například pro realizaci večerního hospodského kvízu, bude nutné udržet zařízení při hře po celý večer, tedy od 18 do 24 hodin.

² Koncept odolné krabičky pro celou skupinu je dle výsledků drobné ankety popsané v kapitole 6 více žádaný než hlasovací tlačítko, které budou uživatelé moci držet v ruce.

³ V rádech desítek Bajtů.

⁴ Daná problematika přesnosti je dále diskutovaná v kapitole 5.4, která se podrobně stanovuje požadavky na algoritmus, konkrétně na synchronizaci času.

Celý systém by měl fungovat v prostředí hospody, kde může být rušen cizím signálem. Musí také fungovat bez jakékoliv jiné infrastruktury, například v lese.

Bezdrátová komunikace musí fungovat pomocí wifi technologie,⁵ která bude užívaná na co nejnižší úrovni.⁶

3.2 Popis užití systému v praxi

Průběh hry se systémem by měl být následující.

Moderátor hry položí otázku. Zařízení jsou na stolech před jednotlivými týmy. Týmy se poradí. Jakmile se domnívají, že znají správnou odpověď, stisknou tlačítko a přihlásí se tak o slovo.

V ten moment se zaznamená daná událost, algoritmus ji zpracuje a v rámci celého distribuovaného systému určí její pořadí v daném kole. Výsledky jsou průběžně zobrazované na zařízení v módu *PRESENTER*.

Pro ukončení kola se na zařízení *PRESENTER* stiskne tlačítko po delší dobu.⁷ Tím se na všech zařízeních vymaže historie a systém je připrav pro další kolo.

⁵ Wifi technologií se myslí *IEEE 802.11*.

⁶ Ideálně adresovat pomocí MAC adres a nevyužívat nic vyššího nad druhou vrstvu OSI modelu.

⁷ Například 10 s. Tato hodnota musí být později doladěna podle uživatelského zážitku, což není obsahem této bakalářské práce.

Kapitola 4

Síťová infrastruktura

Kapitola popisuje volbu vhodného modulu, návrh, realizaci a měření parametrů síťové infrastruktury.¹

4.1 Volba vhodného modulu

Požadavky na síťovou infrastrukturu jsou takové, že by měla zařízení komunikovat pomocí wifi technologie a krátkých zpráv. Těmto parametry je nutné podřídit volbu vhodného hardwaru. Pro něj je navíc potřeba, aby zařízení mohla fungovat minimálně po dobu 6 hodin pouze na baterie. Nedává smysl přemýšlet o zařízeních s vyšším výkonem a větší spotřebou, ale spíše se zaměřit na jednoduché moduly a mikrokontrolery splňující definované požadavky.

Zařízení by tedy mělo:

- podporovat standardy IEEE 802.11 b/g/n,
- mít možnost pracovat s MAC rámci,
- mít vstupní a výstupní brány pro zapojení tlačítek a dalších nutných periferií.

Z průzkumu trhu mi vyšlo šest možných zařízení,² Velice rychle jsem zavrhl poslední dvě zmiňovaná, a to především z důvodu toho, že předchozí čtyři zařízení se těší větší oblibě a na internetu je větší komunita, která mi v případě nejasností s modulem může poradit.

název	CPU	Flash	RAM
ESP32 Espressif	dual-core Xtensa LX6 (160 nebo 240 MHz)	pouze externí	320 kB (SRAM)
ESP32-S2 Espressif	single-core Xtensa LX7 (240 MHz)	pouze externí	320 kB (SRAM)
ESP8266 Espressif	Xtensa L106 (80 nebo 160 MHz)	pouze externí	32 kB instrukce 80 kB data
Pico W Raspberry Pi	dual-core Cortex-M0+ (133 MHz)	16MB (off-chip flash)	264kB

Tabulka 4.1. Srovnání rozdílných parametrů modulů.

Všechna zařízení v tabulce 4.1 podporují požadovanou verzi IEEE 802.11 b/g/n.

Pro testování síťové infrastruktury jsem se rozhodl využít zařízení ESP32-S2. To především z důvodu rychlosti procesoru, možnostem optimalizace spotřeby energie a dostupnosti na trhu. Konkrétně se jednalo o moduly *ESP32-S2-pico* a *ESP32-S2-LCD-0.96inch* s displejem pro účely ladění.

¹ Kapitola částečně vychází z práce a závěrů mého závěrečném projektu absolvovaném v rámci zimního semestru 2022 na ČVUT FEL.

² ESP32, ESP32-S2, ESP8266, PI Pico. CC3200 a ATSAMW25.

Jedná se o wifi vývojové desky se základními periferiemi jako je ADC převodník, nebo komunikační rozhraní I2C, SPI či UART. Deska integruje *low-power Wifi System on Chip* (SoC). Oproti ESP32, které má 2 jádra procesoru, ESP32-S2 má pouze jeden Xtensa single-core 32-bit procesor novější generace a podporuje frekvenci hodin 240 MHz.

Další specifikace je možné najít na stránkách dodavatele.³ Do přílohy C přikládám relevantní schémata.

V pozdější fázi projektu při implementaci algoritmu jsem byl nucen přejít na ESP32,⁴ a to z důvodu poškození modulů ESP32-S2. Díky této změně jsem ze single-coru přešel na dual-core. Umožnilo mi to lépe rozložit výpočetní kapacitu a zefektivnit tak celý provoz zařízení. Negativem byla implementační náročnost, protože přibyla nutnost užití RTOS, konkrétně FreeRTOS.

4.2 Požadavky na síťovou infrastrukturu

Pro správné fungování zařízení je třeba vybudovat síťovou infrastrukturu a zjistit její klíčové parametry. To především z důvodu abstrahování problému síťové infrastruktury.

Síť by dle požadavků v kapitole 3.1 měla komunikovat pomocí wifi technologie, která bude fungovat na co nejnižší úrovni. Informace by se měly posílat pomocí krátkých zpráv. Síť musí fungovat i v prostředí bez jakékoliv jiné infrastruktury. Nedává tudíž smysl využívat typickou hierarchickou strukturu sítí.⁵

4.3 Protokol ESP-NOW

Při studiu možností IEEE 802.11 pro IoT zařízení jsem se setkal pouze s hierarchickou síťovou strukturou. Proto jsem se rozhodl pro distribuovanou síť, kde si budou všechna zapojená zařízení na počátku rovna. Toto rozhodnutí také podpořil požadavek, aby technologie fungovala nezávisle na jiné síťové infrastruktuře, která by v případě aplikace hierarchické struktury byla potřebnou.

Při tvorbě rešerše jsem objevil nemalé množství protokolů, s nimiž by mohla zařízení mezi sebou komunikovat. Nejvhodnější pro mou aplikaci by byl klasický TCP/IP. Po diskusi s vedoucím práce jsem se rozhodl pro experimentálnější způsob. Nepoužiji žádný ze zmiňovaných protokolů, nýbrž se pokusím implementovat vlastní protokol, který by dle definice IEEE 802.11⁶ posílal MAC rámce. Celá síť by byla adresovaná na úrovni MAC adres.⁷

Po podrobnější analýze jsem zjistil, že implementovat vlastní protokol je zcela mimo rozsah této bakalářské práce. Objevil jsem ale protokol ESP-NOW, který je definováný přímo pro mnou zvolené moduly, zajišťuje komunikaci pomocí malých zpráv s nízkou latencí a funguje na spojové vrstvě OSI modelu.

³ <https://www.waveshare.com/wiki/ESP32-S2-Pico>

⁴ Konkrétně se jednalo o vývojové desky ESP-WROOM-32 ESP32 ESP-32S 2.4GHz dostupnou na <http://www.briv.cz/p/5349/esp-wroom-32-esp32-esp-32s-2-4ghz-vyvojarska-deska-s-wifi-bt> a ESP-WROOM-32 2.4GHz Dual-Mode WiFi+Bluetooth rev.1, CP2102 dostupnou na <https://www.laskakit.cz/iot-esp-32s-2-4ghz-dual-mode-wifi-bluetooth-rev-1--cp2102/>.

⁵ Více informací o hierarchické struktuře sítě je možné dohledat v kapitole 2.3, která se dané problematice přímo věnuje.

⁶ Konkrétně druhé úrovně síťového modelu OSI (linkové).

⁷ Nepovedlo se mi totiž žádný takový existující protokol vyhledat. Ani po diskusi s odborníky na telekomunikační technologie z řad učitelů, jsem se nedozvěděl o jakémkoliv již funkčním způsobu posílání dat pouze na druhé vrstvě síťového OSI modelu.

■ 4.3.1 Popis technologie

Protokol **ESP-NOW** je definovaný firmou Espressif speciálně pro procesory ESP32, ESP8266, ESP32-S a ESP32-C. Je také unikátní tím, že horních pět vrstev OSI modelu spojuje do jedné zvané ESP-NOW. Díky tomu je protokol sice méně robustní, naopak je díky němu možné dosáhnout vyšších rychlostí za menší cenu výpočetní sily.

Protokol je flexibilní, protože dokáže přenášet data jak v *unicast* tak v *broadcast* módu a podporuje jak *one-to-many*, tak *many-to-many* komunikaci.

■ 4.3.2 Formát rámce

Výchozí přenosová rychlosť (*bit rate*) ESP-NOW je 1 MBps. Rychlosť lze měnit pomocí funkcí definovaných protokolem. Formát rámce vychází z definice *IEEE 802.11*,⁸ konkrétně tzv. *Action Format*. [25–26]

- **MAC Header** (24 bytes)
- **Category Code** (1 byte) indikující rámc typu *Action Frame*. V případě ESP-NOW se jedná o tzv. *vendor-specific action frame*, a proto je hodnota nastavena na 127.
- **Organization Identifier** (3 bytes) je jedinečný identifikátor. V případě ESP-NOW jde o hodnotu 0x18fe34, která tvorí první 3 byty MAC adresy příslušící firmě Espressif.
- **Random Values** (4 bytes) je náhodná hodnota, která se používá k prevenci před relay útokem.⁹
- **Vendor Specific Content** (7~257 bytes) je samotný obsah protokolu, který je tvořen vloženým rámcem, a je zobrazen na obrázku 4.2.
- **Element ID** (1 byte) hodnota je nastavena na 221 pro indikaci začátku *vendor-specific* části.
- **Length** (1 byte) je celková velikost částí *Organization Identifier*, *Type*, *Version* a *Body*.
- **Organization Identifier** (3 bytes) stejná hodnota jako v nadřazeném rámc, do kterého je tento zapouzdřen. Jedná se o hodnotu 0x18fe34, která odpovídá prvním třem bajtům MAC adres příslušících firmě Espressif.
- **Type** (1 byte) určuje typ action rámce od Espressifu. V případě užití ESP-NOW je hodnota rovna 4.
- **Version** (1 byte) pole pro nastavení konkrétní verze.
- **Body** (0~250 bytes)¹⁰ tělo samotného rámce obsahující zprávu.
- **FCS** (4 bytes) frame check sequence

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content	FCS
24 bytes	1 byte	3 bytes	4 bytes	7~257 bytes	4 bytes

Obrázek 4.1. Vizualizace ESP-NOW Action Frame.

⁸ IEEE 802.11 kapitola 9.6.7.11 *Vendor Specific Public Action frame format*.

⁹ **Relay útok** je kyberbezpečnostní útok využívající techniky *man-in-the-middle*. Pro více podrobností doporučuji navštívit https://en.wikipedia.org/wiki/Relay_attack.

¹⁰ V oficiální dokumentaci byla v době, kdy jsem na práci pracoval, chyba. Body bylo definované tak, že může dosahovat velikosti až 250 bajtů. To ale není možné, jelikož součet by přesáhl maximální povolenou velikost, tedy 255 bajtů pro *Vendor Specific Content*. Chybu jsem nahlásil a věřím, že bude co nejdříve opravena. <https://github.com/espressif/esp-now/issues/59>

Element ID	Length	Organization Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0~250 bytes

Obrázek 4.2. Vizualizace ESP-NOW Content Frame.

Protokol ESP-NOW má několik dalších odlišností oproti klasické komunikaci dle IEEE 802.11. Liší se kontrolním rámcem (*Frame Control*)¹¹ v tom, že bity FromDS a ToDS jsou nulové a v obecném rámci (*General frame format*),¹² pro který platí, že:

- v první adrese (*Address 1*) je uložena cílová destinace ve formě MAC adresy,
- druhá adresa (*Address 2*) ukládá adresu zdroje
- a třetí adresa (*Address 3*) je vždy nastavena na vysílání broadcastem, tedy na adresu 0xff:0xff:0xff:0xff:0xff:0xff.

Protokol ESP-NOW je zabezpečen pomocí ECDH a AES128-CCM.¹³ Více podrobností o bezpečnosti je možné dohledat v dokumentaci¹⁴ nebo přímo v oficiální repozitáři implementace protokolu.¹⁵ Jelikož šifrování nebude při implementaci použito, nebudu se mu v této práce věnovat více.

■ 4.3.3 Módy provozu wifi

Moduly podporující ESP-NOW¹⁶ umožňují dva módy provozu wifi technologie: Station mode a Access Point mode (Soft-AP mode).¹⁷

Station mode (STA) se používá v případě, kdy se modul připojuje k access pointu, jako je například router. Typickou aplikací je senzor, který například snímá teplotu v místnosti, je napojen na router, který poskytuje připojení k WLAN a v pravidelných intervalech odesílá data do cloudu.

Opakem je **Access Point mode** (Soft-AP mode někdy také označován pouze zkratkou AP mode), který funguje jako router v síti a může k němu být připojeno více zařízení. Typickou aplikací může být například spuštění wifi a webového serveru na modulu. V momentě, kdy se do sítě připojí nějaké zařízení, webový server vykreslí specifikovanou stránku.¹⁸ Tento mód je v této práci vhodný pro využití implementace stavu zařízení v módu *PRESETER*.

Protokol ESP-NOW může běžet ve všech módech.

■ 4.3.4 Popis fungování

Před využíváním ESP-NOW je třeba aktivovat wifi a **inicializovat** nutné struktury pomocí funkce `esp_now_init()`. Dokumentace doporučuje striktně dodržet zmiňované pořadí, tedy prvně aktivovat wifi a až poté protokol ESP-NOW.

¹¹ Definovaném v IEEE 802.11 kapitola 9.2.4.1 *Frame Control field*

¹² Definovaný v IEEE 802.11 kapitola 9.2.3 *General frame format*

¹³ Definovaných v IEEE 802.11-2012.

¹⁴ https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html#security

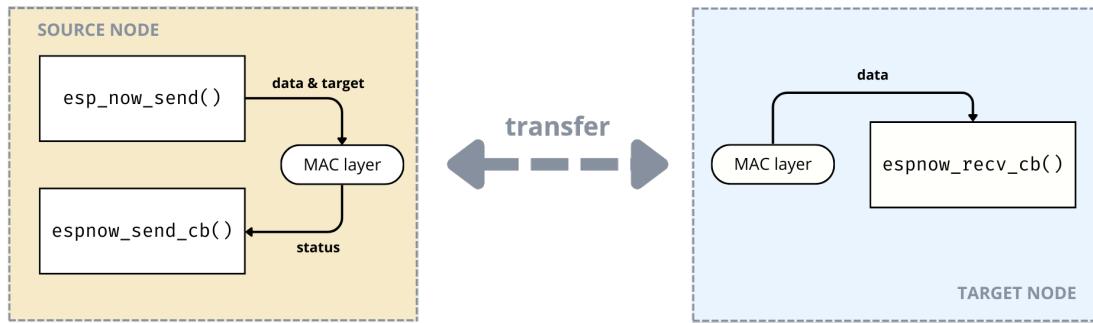
¹⁵ <https://github.com/espressif/esp-now/tree/master>

¹⁶ ESP32 a ESP8266

¹⁷ Ve skutečnosti existuje více módů provozu wifi na ESP modulech. Jedná se ale o hybridy již zmíněných módů, proto se jim nebudu více do detaily věnovat.

¹⁸ Poměrně pěkně a detailně je implementace Wifi technologie na ESP zařízeních popsána v hlavičkovém souboru pro framework *ESP-IDF*, který je dostupný na adrese https://raw.githubusercontent.com/espressif/esp-idf/4f0769d2ed074ef770c6432565d6e5610124e9ea/components/esp_wifi/include/esp_wifi.h

Pro odeslání dat je třeba mít informace o cílových zařízeních přidaná v tzv. `esp_now_peer_info`. Do této struktury je data možné přidávat pomocí funkce `esp_now_add_peer()`. Struktura uchovává MAC adresu zařízení, informace o zabezpečení komunikace a další parametry komunikace. Podle verze protokolu je struktura limitovaná počtem šifrovaných a nešifrovaných zařízení.¹⁹ Ve verzi 2.1.0 se jedná o maximálně 20 nešifrovaných případně 17 šifrovaných zařízení. Velikost jde uživatelsky v kódu změnit. To ale může mít negativní vliv na rychlosť fungování protokolu.



Obrázek 4.3. Schéma odesílání dat pomocí ESP-NOW.

Samotné **odesílání a přijímání dat** je ilustrované na obrázku 4.3 a funguje pomocí tzv. *call-back* funkcí. Funkce jsou v podstatě *Wifi procesy* s vysokou prioritou, které je třeba uživatelsky definovat. Protokol nedefinuje žádnou strukturu pro ukládání a pokročilé zpracovávání dat. Odesílání pak funguje tak, že uživatel zavolá funkci `esp_now_send()`, která data odešle na druhou vrstvu protokolu nazývanou MAC layer. Protokol pomocí *call-back* funkce `espnow_send_cb()` odešle status úspěšnosti operace. Ověření úspěšnosti odesílání se liší v závislosti na módu komunikace. V případě *broadcastu* je zaznamenané pouze úspěšné odesílání. V případě *unicastu* se čeká na potvrzení od adresáta.²⁰ Přijímání dat funguje na podobném principu. Jakmile *Wifi proces* zaregistruje příchozí data, zavolá *call-back* funkci `espnow_recv_cb()`, která data dle definice zpracuje.

Protokol podporuje další pokročilé funkce jako je například **změna rychlosti přenosu** či využívání **úsporného režimu vysílání** s metodou nastavení vysílacích intervalů.

4.3.5 Limity technologie

Protokol ESP-NOW je definován poměrně nově. Proto dochází k neustálému vývoji, drobným změnám a posouvání limitů.

První verze (1.0.0) vychází v roce 2016 a dle manuálu²¹ je limitována tím, že:

- nemá implementovaný broadcast,
- je možné šifrovat spojení pouze s 10 zařízeními
- a maximální payload zprávy je 250 bajtů.

Další verze rozšiřují počet šifrovaných zařízení, rychlosť přenosu a přidávají podporu pro broadcast. Tyto limity se dle mého názoru budou dále rozvíjet. Vývojáři mají v nejbližší době v plánu knihovnu rozšiřovať o možnost clusterizování sítě.

¹⁹ Přesné limity doporučuji sledovat přímo v oficiálním repozitáři protokolu ESP-NOW: <https://github.com/espressif/esp-now/tree/master>.

²⁰ Tato informace není nikde oficiálně dokumentována a byla zjištěna experimentálně.

²¹ https://www.espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf

Osobně také vidím problém v nedostatečné dokumentaci a vysvětlení, jak celý protokol funguje. Uživatel je nucen detailně zkoumat kódy a následně chování testovat, protože některé části nejsou veřejně dostupné.

Limit, který se dle mého názoru nezmění, je velikost zprávy. Ta je dána typem využívaného IEEE 802.11 rámce.

■ 4.3.6 ESP-NOW pro Arduino framework

Oficiální implementaci protokolu je možné aplikovat pouze s využitím **IoT frameworku ESP-IDF**. Pro využití **Arduino frameworku** je dostupná neoficiální verze od autora *Junxiao Shi* v repozitáři na GitHubu.²² Tato verze není oficiálně udržovaná a v době psaní této bakalářské práce má z mého pohledu dva zásadní limity.

- Knihovna **vychází ze zastaralé verze** (ESP-NOW 1.0.0). Tudíž neobsahuje opravy a nové funkce, které přináší až druhá verze protokolu.
- Knihovna implementuje **vlastní pseudo broadcast**. Ten funguje tak, že provede *BSSID*²³ skenování a na každé nalezené zařízení odešle zprávu pomocí *unicastu*.

Arduino Framework se nicméně těší vysoké oblibě. Z toho vyplývá, že většina veřejně dostupných návodů na internetu jsou nevhodné zdroje, protože popisují špatnou verzi.

■ 4.4 Architektura síťové infrastruktury

Komunikace mezi zařízeními je postavena na protokolu ESP-NOW. Ten hravě splňuje omezení počtu 10 zařízení, drobných zpráv, i komunikace pomocí wifi na nízké úrovni OSI modelu. Pro zpracování je nutné využít pomocné struktury. Konkrétně se jedná o frontu, která pomáhá zpracovávat příchozí zprávy. Celý proces putování dat během komunikace je ilustrován na obrázku 4.4 a probíhá tak, že:

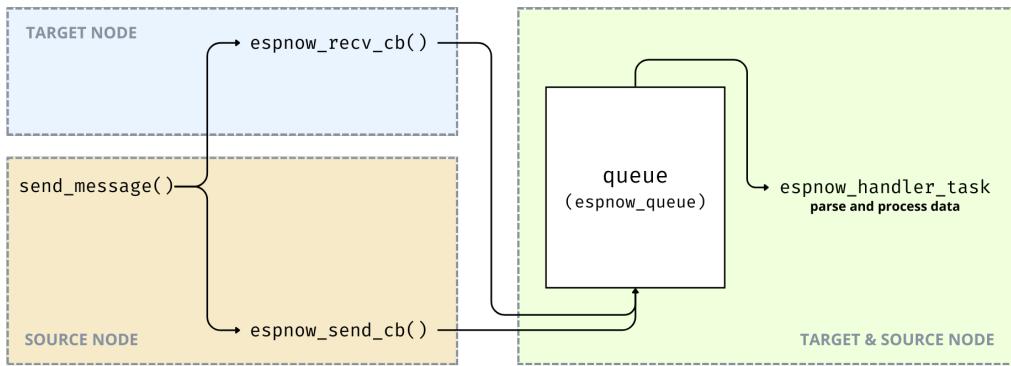
- před odesláním zprávy se připraví data, zkontroluje se, zdali je cílové zařízení přidáno mezi známými destinacemi a dojde k odeslání pomocí `send_message()`,
- odesílatel obdrží status, který zachytí *call-back* funkci `espnow_send_cb()` a odešle informace do fronty
- příjemce obdrží zprávu, kterou zachytí pomocí *call-back* funkce `espnow_recv_cb()` a data odešle do fronty.

Data z fronty jsou následně vytažena procesem `espnow_handler_task`, který má za úkol data zpracovat a zavolat následnou navazující akci. V podstatě funguje jako takový *ESP-NOW event hub*.²⁴

²² <https://github.com/yoursunny/WifiEspNow>

²³ Zkratka BSSID znamená **Basic Service Set Identifier** neboli 48-bitový identifikátor, pod kterým je zařízení identifikované v síti. [27]

²⁴ koncentrátor událostí

**Obrázek 4.4.** Workflow dat v sítové infrastruktuře.

Postup byl zvolen z důvodu efektivity fungování, především prioritizace procesů a jejich efektivního rozložení mezi více jader. Nejvyšší prioritu mají vždy *call-back* funkce, po nich následuje *handler task* a nejnižší má vždy funkce sloužící k odesílání zpráv.

Navíc struktura `QueueHandle_t` využívaná pro frontu je bezpečná. Díky ní není nutné pro zpracování událostí využívat více kritických sekcí.

4.5 Parametry sítové infrastruktury

Pro možnost abstrakce sítové infrastruktury v další práci je nutné změřit a otestovat parametry, kterých daná sítová infrastruktura dosahuje, především vlastnosti modulu a protokolu ESP-NOW. Testování bylo prováděno na zařízeních ESP32-S2 v různých prostředích a za různé konfigurace. Testování mělo simuloval reálný provoz. Parametr, který mne zajímal, byla velikost *Round Trip Time*, tedy doba přenosu informace na cílové zařízení a zpět.

Při měření jsem sledoval následující parametry:

- prostředí, v němž bylo měření prováděno,
- překážku mezi zařízeními,
- vzdálenost mezi zařízeními,
- velikost zprávy,²⁵ ²⁶
- počet odeslaných zpráv,
- typ odesílání (*broadcast/unicast*),
- počet chybných zpráv.²⁷

Z měření vyplývá, že vzdálenost zařízení od sebe ovlivňuje ztrátovost zpráv a velikost zprávy ovlivňuje dobu odesílání. Ve výsledcích je také vidět speciální chování pro vysílání v módu *broadcastu*. Zde se děje situace popsaná v kapitole 4.3.6.

Zařízení jsou schopna efektivně fungovat v prostředí s rušením. Na otevřeném prostřanství fungují i na vzdálenost více než 500 m bez nutnosti přídavné antény.

Podrobné výsledky i měření včetně komentáře lze nalézt v příloze B nebo v repozitáři projektu.²⁸

Závěrem měření mohu konstatovat, že výsledky prokazují, že sítová infrastruktura splňuje očekávané požadavky. Je vhodnější posílat zprávy pouze v módu *unicast* a infrastruktura je vyhovující pro použití v implementaci.

²⁵ Někdy označováno také jako *payload*.

²⁶ Maximální velikost zprávy je 250 bajtů.

²⁷ Měření tohoto parametru bylo přidáno od scénáře A, kde ovšem k žádné ztrátě nedocházelo.

²⁸ <https://github.com/petrkucera/rafting-button/>

Kapitola 5

Algoritmus

Kapitola popisuje známé algoritmy vhodné pro daný problém, specifikuje požadavky na algoritmus, které dává do kontextu a diskutuje jejich výhody a nevýhody. Následně detailně popisuje zvolený algoritmus pro implementaci.

5.1 Požadavky na algoritmus

Od algoritmu se očekává řešení **problému konsenzu**,¹ respektive shody celého systému na tom, v jakém pořadí byla tlačítka stisknuta.

Systém by měl být **distribuovaný**, tj. ve výchozím módu by měla být všechny zařízení² na stejně úrovni a žádné z nich by před spuštěním algoritmu nemělo vůči jinému být ve vztahu *master - slave*.

Systémem je v kontextu této práce označován soubor hlasovací zařízení spojených pomocí wifi technologie a komunikujících pomocí protokolu ESP-NOW.

Pojmem **distribuovaná síť**, obecně **distribuovaný systém** (DS)³ označujeme soubor autonomních nezávislých zařízení, která spolu komunikují skrze síť a jejich dorozumívacím prostředkem jsou zprávy. Dle Andrewa Tanenbauma se v ideálním případě celý distribuovaný systém jeví jako koherentní systém. Leslie Lamport je naopak kritičtější a říká, že pojmem distribuovaný systém je označován takový, ve kterém selhání počítače, o kterém jste nevěděli, že funguje, učiní užívaný počítač nepoužitelným. [28–30]

Literatura se v následující terminologii rozchází a také na každý z problémů v DS jsou kladený jiné požadavky, obecně ale můžeme říci, že rozlišujeme dva základní požadavky na DS:

- **živost** (*liveness*) - časem v DS bude dosažen žádoucí vztah,
- **bezpečnost** (*safety*) - nedojde k nežádoucímu stavu.

Dle **FLP teorému** v asynchronním distribuovaném systému nelze dosáhnout současně živosti a bezpečnosti distribuovaného výpočtu, pokud může docházet k selhání. [31] V praxi se proto spíše vyžaduje bezpečnost a díky částečné synchronicitě zařízení v DS lze předpokládat, že algoritmus doběhne, tj. dosáhne výsledků v konečném čase.⁴ [32]

5.1.1 Obecné požadavky na distribuovaný systém

- **Bezpečnost** - systém musí garantovat, že nedojde k nežádoucímu stavu, tj. stavu s kterým se nepočítá.
- **Konečnou živost** - není nutné, aby byl distribuovaný výpočet dokončen do určitého času. Nýbrž je požadováno dokončení výpočtu v konečném čase.

¹ Pojem problém konsenzu je myšlený problém nutnosti najít shody dvou až n procesů na specifické hodnotě. Ve vztahu k našemu tématu se jedná o shodě na logu určujícím pořadí.

² Literatura používá pojmenování proces a to z důvodu, že často může jít o DS v rámci jednoho zařízení. V této práci z důvodu smyslu na základě kontextu budu často místo pojmu *proces* používat pojednání *zařízení*.

³ Pojem **distribuovaný systém** je dále označován i zkratkou **DS**.

⁴ Takovýto požadavek označujeme jako tzv. *konečnou živost* (*eventual liveness*).

- **Uspořádání** - musí být dodrženo kauzální uspořádání v závislosti na čase, tj. musí být dodrženo pořadí.

5.2 Představení známých algoritmů

Distribuované systémy jsou s námi už několik desítek let. Přesto se stále jedná o moderní obor, který se neustále rozvíjí, a to například i s masivním příchodem *cloud computingu*. Pro horizontální škálování je třeba umět rozložit výpočetní sílu mezi více serverů. Proto existují algoritmy, které základní úlohy DS řeší. Jedná se především o problémy: detekce selhání, kauzalita a času, globální snapshot, vyloučení procesů, volba lídra a konsenzus. Ve vztahu k mé práci nejsou všechny problémy relevantní, proto se zaměřím pouze na **kauzalitu a čas a konsenzus**, které budu ve svém projektu řešit.

5.2.1 Kauzalita a čas

Problematiku **kauzality a času** je třeba v DS řešit především z toho důvodu, že ve většině případů nemáme garanci toho, že fyzické hodiny zařízení mají stejnou rychlosť, že aktuální hodnota hodin je v různých zařízeních rozdílná, že přenos zprávy ze zařízení A do zařízení C bude trvat stejně dlouho jako ze zařízení B do zařízení C nebo že zprávy jsou přijaté ve stejném pořadí jako byly odeslány.

První tři zmínované nejistoty, konkrétně **mimoběžnost hodin** (*clock skew*) a **drift hodin**, se řeší synchronizací času. Podle zdroje referenčního času rozlišujeme synchronizaci na interní a externí. **Interní** se snaží o minimalizaci časového rozdílu δ mezi dvěma zařízeními (d_i, d_j), tedy

$$|T_i - T_j| \leq \delta,$$

kde T_i je aktuální čas v zařízení d_i a T_j je aktuální čas v zařízení d_j . Druhou možností je užití **externí synchronizace**, kdy dochází minimalizaci časového rozdílu δ mezi externím zdrojem E a lokálními hodinami T_i , tedy

$$|T_i - E| \leq \delta.$$

Příkladem interní synchronizace je například Berkeley algoritmus a externí NTP nebo Cristianův. [33–34]

Důležitým pravidlem je, že nikdy nesmíme nastavit čas dozadu. Mohlo by se stát, že nastane víckrát situace se stejným časem. Proto je vhodnější čas zpomalit.

Pro implementaci je poměrně jednoduchý a v DS, kde se čas synchronizuje napřímo mezi zařízeními, je z pohledu latence vysoce efektivní **Cristianův algoritmus**. Algoritmus předpokládá, že je doba přenosu symetrická. Přesně funguje tak, že pokud zařízení A chce synchronizovat čas od zařízení B,

- pošle mu zprávu s žádostí o referenční čas a uloží si aktuální čas T_0 při odeslání zprávy.
- Zařízení B pošle zařízení A zpět svůj čas.
- Zařízení A nastaví čas jako přijatou hodnotu s dobou přenosu, tedy

$$T_A = T_B + \frac{RTT}{2} = T_B + \frac{T_B - T_0}{2}.$$

Pokud není komunikace plně symetrická, lze přesnost zlepšit odesláním více hodnot a spočítáním průměru *RTT*. Autorem algoritmu je Flaviu Cristian. [35]

Druhým algoritmem, který by šlo použít k synchronizaci času je v průmyslu často využívaný **protokol PTP** (*Precision Time Protocol*), který je definován standardem IEEE 1588. Algoritmus si klade především za cíl:

- minimální požadavky na hardware zařízení,
- použitelnost s minimální zprávou, a to i v malých sítích,
- podporu i v levných a rozšířených sítích
- a přesnost v řádu maximálně mikrosekund.⁵

Samotný princip PTP je založený na **pravidelném zasílání časových značek**, ze kterých se následně určí časový posun hodin a zpoždění dané vlivem přenosu mezi jednotlivými zařízeními. V první verzi protokol nefunguje efektivně v hierarchických sítích z důvodu kumulace chyby. Pozdější verze chybu opravuje.

V krátkosti PTP funguje tak, že si v první fázi zvolí zařízení, které bude v módu *MASTER* a to pomocí algoritmu BMC.⁶ Ostatní jsou v módu *SLAVE*. Po volbách každé ze *SLAVE* zařízení zjistí posun vlastních hodin, a to pomocí zpráv, které jsou pravidelně odesílány z nadřazeného zařízení. V druhém kroku zjistí chybu danou dobou přenosu a to tak, že nadřazenému zařízení odešle speciální zprávu *Delay_Req*. Následně za splnění předpokladů, že zpoždění způsobené dobou přenosu je symetrické v obou směrech, posun hodin je po dobu výměny zpráv konstantní a že *MASTER* a *SLAVE* mohou přesně určit čas přijetí a odeslání, jsme schopni určit offset, tedy čas, o který se mají hodiny posunout, aby byly synchronizovány. [36–37]

Kromě synchronizace času můžeme v DS řešit i **kauzalitu**. Tedy aby bylo zachováno pořadí událostí. K této problematice můžeme vyžít fyzických hodin se synchronizovaným časem nebo logických hodin. **Logické hodiny** jsou takové, které se inkrementují v případě nějaké události jako je přijmutí či odeslání zprávy. Jednotlivé události pak jsou mezi sebou ve vztahu *stalo se před*.

Jejich využití v DS vypadá tak, že každé zařízení má svoje logické lokální hodiny. V případě přijmutí zprávy se lokální hodiny inkrementují o jedna. Pokud je hodnota lokálních hodin nižší než hodnota obsažená ve zprávě, nechá se nastavena lokální hodnota. Pokud je vyšší hodnota ve zprávě, bude nastavena jako lokální logický čas.

Typů logických hodin je více. Pokud mají jednu dimenzi mluvíme o **Lamportových hodinách**. Ty respektují kauzalitu, ale nerozlišují současnou událost. Proto pokud přidáme do logických hodin dimenzi pro každé zařízení, můžeme mluvit o **vektorových hodinách**. Ty implikují kauzalitu a zajišťují i rozlišení současných událostí. Existují také **maticové hodiny**. Fungují jako vektorové, navíc kromě lokálních hodin si udržují i informace ostatních známých logických hodin. [34, 38]

5.2.2 Konsenzus

Problém konsenzu je v DS označovaná situace, když se N různých zařízení snaží **shodnout na výstupní hodnotě**. V kontextu zadání této práce se jedná především o shodě na pořadí hlasování.

Aby byl algoritmus řešící konsenzus platný, musí splňovat následující vlastnosti.

⁵ Neplatí vždy. Záleží na vrstvě, ve které je protokol implementován. Typicky se jedná o druhou (v lokální síti), pak je možné dosáhnout uvedené přesnosti. Pokud se ale PTP implementuje ve vyšší vrstvě, například je součástí UDP, pak je téměř nemožné dosáhnout požadované rychlosti.

⁶ Best Master Clock Algorithm

- Všechna zařízení, která nehavarovala, se musejí **shodnout na výstupu**.
- Na všech zařízeních po skončení algoritmu musí být **výstup stejný**.
- Každé zařízení musí skončit běh algoritmu konsenzu v **konečném čase**.

Problém konsenzu je v DS jedním z nejdůležitějších, a to proto, že **je ekvivalentem** k řešení dalších problémů DS, jako je například *vyloučení procesů*, *volba lídra* či *totálně usporádaný multicast*. Tuto záležitost naznačuje i četnost vědeckých článků právě na toto téma. Jeden z nich například řeší problematiku toho, že v posledních letech narůstá množství dat, která sbírají IoT zařízení. Data nám pomáhají pochopit procesy, jako bylo například chování a šíření COVID-19. Při sběru velikého množství dat, občas i privátních, je důležitá anonymita. Proto systémy využívají *privátního blockchainu* společně s *raft algoritmem*, který jim dopomáhá řešit problematiku konsenzu, respektive slučování informací dohromady. Pokud se ale jedná o moc veliké množství dat, vedoucí uzel je pak při běhu algoritmu enormně zatížen. [39]

Obecně můžeme tvrdit, že konsenzus v synchronních i asynchronních systémech bez selhání a v synchronních systémech se selháním je řešitelný. Konsenzus ale nemůžeme vyřešit s garancí všech požadovaných vlastností pro platnost v případě, že je DS asynchronní a dochází k selhání. Tato skutečnost implikuje **FLP teorém**, kterému se detailněji věnuji v kapitole 5.1. [32]

Jedním z nejznámějších algoritmů řešící právě konsenzus je poměrně nový⁷ **RAFT**. Jedná se o paralelní algoritmus konsenzu pro správu a replikaci logů. Poskytuje stejný výsledek jako protokol *Paxos*. Jeho efektivita je stejná, ale **je srozumitelnější**. Proto poskytuje lepší funkcionality a snadnější implementaci v různých systémech. Pro zvýšení srozumitelnosti, Raft odděluje jednotlivé klíčové prvky konsenzu.⁸

Běh raftu se skládá z fází:

- volby lídra,
- běžného provozu (základní replikace logu),
- bezpečnosti a konzistence po změně lídra,
- neutralizace starých lídrů,
- a interakce s klienty.

První fází je volba lídra. Ta probíhá tak, že každé zařízení může být právě v jednom z následujících stavů.

- *LÍDR* obsahuje požadavky klientů a replikuje log.
- *NÁSLEDOVNÍK* je pasivní zařízení, které pouze reaguje na zprávy od jiných zařízení.
- *KANDIDÁT* je přechodná role v průběhu volby lídra.

Platí, že za běžného provozu existuje pouze 1 lídr a N následovníků.

Celkový běh raftu se rozděluje do jednotlivých časových⁹ období, tzv. **epoch**. Každá epocha má svoje ID číslo, které se vždy inkrementálně zvyšuje. Každé zařízení uchovává číslo epochy. Díky tomu se Raft umí vyporádat i s např. vypnutím. Epochu se skládá ze dvou částí: volby lídra a běžného chodu. Může nastat epocha bez lídra, jelikož se nepovede lídra zvolit.

Na začátku volby lídra jsou všechna zařízení *následovníci*. Ti očekávají zprávu od *lídra* nebo od *kandidáta* na lídra. Lídři v průběhu své vlády posílají *heartbeats*, aby si udrželi autoritu. Jakmile *následník* neobdrží zprávu do určitého timeoutu, předpokládá že *lídr* havaroval a iniciuje volbu nového lídra. Zařízení, které spustí volbu nového lídra:

⁷ 2014

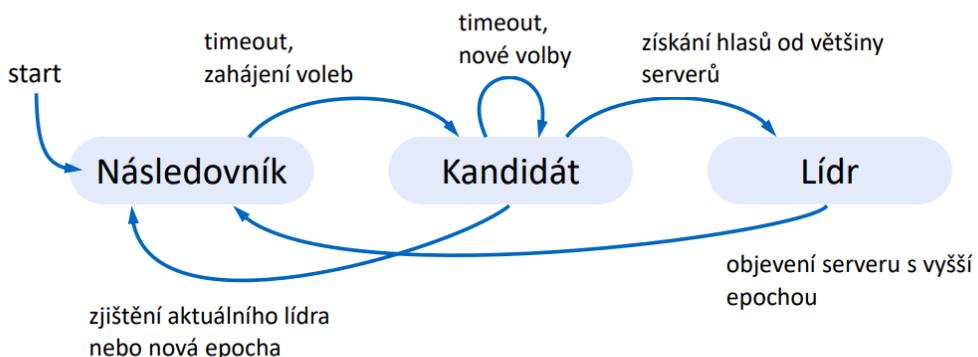
⁸ Například volbu lídra, replikaci logů a bezpečnost.

⁹ Jedná se o *logický čas*.

zvýší číslo epochy, změní svůj stav na *kandidát*, zahlasuje pro sebe, pošle `RequestVote` všem ostatním zařízením a čeká na jeden z následujících stavů:

- obdrží hlasy od většiny zařízení -> změní se na *lídr* a pošle *heartbeat* ostatní procesům,
- přijme zprávu od validního lídra (tedy jiný lídr byl zvolen dříve), vrátí se tedy do stavu *následovník*
- nikdo nevyhraje volby, tj. vyprší *timeout* -> zvýší se ID epochy.

Klíčová vlastnost *bezpečnosti* je splněna, jelikož může být pouze jeden lídr v každé epoše. Byla také splněna *živost*, protože jeden z kandidátů musí časem vyhrát.¹⁰



Obrázek 5.1. Stavový diagram změny rolí při volbě nového lídra v algoritmu RAFT. [40]

Druhou fází běhu raftu je běžný provoz. Ten se stará o distribuci a konsenzus logu. Logy jsou seznamem, který v každé položce obsahuje:

- index dané položky,
- epochu, ve které daný příkaz vznikl,
- a samotný příkaz.

Logy jsou perzistentní, tedy přežijí havárii.¹¹ Dále existuje tzv. potvrzený záznam (*committed*). To je takový záznam, který je již potvrzený, tedy uložený již na většině zařízení.

Provoz pak probíhá tak, že *klient* pošle *lídrovy* příkaz, *lídr* přidá příkaz na konec svého logu. Následně *lídr* pošle zprávu `AppendEntries` *následovníkům* a čeká na odpověď. Jakmile přijde nadpoloviční většina odpovědí, záznam je potvrzen (*committed*). Příkaz se vykoná a odpověď je předána *klientovi*. *Lídr* pošle informaci o potvrzení svým *následovníkům*, kteří vykonají daný příkaz.

Havárie *následovníka* se neřeší. *Lídr* posílá opakováně zprávu `AppendEntries`, dokud doručení neuspěje.

Algoritmus raft je efektivní především díky konzistenci logů. Tedy pokud se záznamy shodují ve všech pozicích s indexy 1 až n obsahujících záznam se shodným číslem epochy a se shodným příkazem.

Díky návrhu Raftu platí tzv. **invarianty raftu**.

¹⁰ Pro minimalizaci kolapsu při volbách -> timeouty jsou nastavovány individuálně v určitém intervalu.

¹¹ Pokud se zařízení vypnou.

- Mají-li záznamy logů uložené na různých zařízeních stejný index a epochu, pak obsahují stejný příkaz a logy jsou identické ve všech předešlých záznamech.
- Je-li daný příkaz potvrzený, pak jsou potvrzeny i všechny předchozí logy.

Při odeslání zprávy `AppendEntries` zpráva obsahuje i index a příkaz předchozího záznamu, který slouží k validaci. A pokud se záznam neshoduje, je zápis odmítnut.

Třetí fází je problematika bezpečnosti a změny lídra. Během normálního fungování je zajištěna konzistence. Jakmile ale lídr havaruje, dochází k nové volbě. Je třeba ale efektivně ošetřit danou situaci, aby se předešlo nekonzistenci. Raft neimplementuje žádnou speciální úklidovou fázi. Nýbrž neustále posílá zprávy, přičemž zpráva je zpráva od lídra. Tedy lídr má vždy pravdu.

Otázka tedy zní, jak ale tedy ošetřit vyskytlé havárie? Prvně si je třeba obecně stanovit, co to znamená **bezpečnost**. Obecně nutná bezpečnost pro garanci replikace je to, že jakmile je příkaz ze záznamu logu vykonán některým zařízením, nesmí žádne jiné zařízení vykonávat jiný příkaz pro stejný záznam. *Tedy než něco vykoná dané zařízení, musíme si být jisti, že jiné zařízení nevykoná místo tohoto příkazu něco jiného. Tohoto bezpečnostního požadavku dosáhneme pomocí tzv. Bezpečnostního invariantu Raftu.* Jakmile lídr prohlásí záznam v logu za potvrzený, jakýkoliv budoucí lídr bude mít tento záznam ve svém logu. Díky tomu platí, že:

- lídři nikdy nepřepisují záznamy ve svých lozích, pouze je přidávají,
- pouze záznamy v logu lídra mohou být potvrzeny,
- a záznamy musí být v logu potvrzeny předtím, než jsou vykonány na daných zařízeních.

Proto platí první podmínka říkající, že jelikož dosavadní logika fungování Raftu bezpečnostní invariant negarantuje, snaží se Raft zvolit lídra, který má nejúplnější log. Tj. epochu nahradí nejvyšší epochou případně pokud mají stejnou epochu, tak s nejvyšším indexem. A druhá vyžaduje, že aby lídr požadoval záznam za potvrzený musí být:

- záznam uložený na většině serverů,¹²
- také alespoň jeden nový záznam z lídrovy aktuální epochy na většině serverů.¹³

Jakmile je tato i předchozí podmínka splněna, již lze prohlásit invariant a bezpečnost za splněnou.

K opravě logu následovníků dochází tak, že nový lídr musí udělat logy následovníků konzistentních se svým logem. Tj. smazat přebytečné záznamy a doplnit chybějící záznamy. V praxi se oprava implementuje tak, že lídr určuje proměnnou `nextIndex` pro každého následovníka: index dalšího záznamu logu, který by měl být odeslán následovníkovi a je inicializována na $1 + \text{index posledního záznamu lídra}$. Pokud kontrola konzistence `AppendEntries` selže, sníží `nextIndex` o jednu a zkusi znova. *V podstatě prvně zjistí, jak moc do historie se musí vrátit. Tam vše ustříhne a naplní zásobník novými logy.*

Čtvrtá fáze popisuje neutralizace starého lídra, kterou je třeba řešit situaci, kdy zhavaruje lídr. Např. se od něho opozdí zprávy. Později opět ožije. Jak ale zjistí, že již dále není lídrem? Řešením je to, že zařízení nepřijme zprávu s nižší epochou. Přesněji pokud zařízení zachytí zprávu s nižší epochou od starého lídra, pošle mu zpátky zprávu s informací, že si má zvýšit epochu a zrušit status lídra.

Pátá poslední fáze řeší interakci s klientem. V ní je definován **protokol klienta**, kde klienti posílají příkaz lídrovi. Pokud zařízení není lídrem, pošle zařízení klientovi současného lídra a ten již ví na koho směřovat zprávu.

¹² známe podmítku

¹³ podmínka navíc

Může se stát, že *lídř* havaruje poté, co už ale vykonal příkaz před odesláním odpovědi. Existuje tedy riziko opakovaného vykonání příkazu. Řešením je, že každý příkaz má jedinečné ID příkazu. Při přijmutí příkazu, pak vždy *lídř* zkонтroluje, zdali již ID není na zařízení uloženo. [40–42]

Nevýhodou Raftu může být například vysoká výpočetní náročnost na lídra. To lze ale řešit například drobnou úpravou algoritmu, jak popisují Yang, Doh a Chae ve svém článku. [39]

Algoritmus Raft je jeden z nejmodernějších (2014) a vysoce využívaných paralelních algoritmů. Je implementován např. v distribuovaných databázích, které dokážou běžet nad clusterem nebo se využívá v již zmiňované technologii blockchainu.¹⁴

5.3 Rozbor problému

Abstrahujme síťovou vrstvu DS a předpokládejme, že jsou všechna zařízení DS propojena, komunikace mezi nimi je spolehlivá, tj. po odeslání každé zprávy může odesílatel obdržet od adresáta potvrzení, zdali byl přenos úspěšný či nikoliv. Úkolem je při stisku periferie (*tlačítka*) rozdistribuovat informaci do ostatních zařízení DS a určit pořadí.

Pro **určení pořadí** je třeba buďto znát *kauzalitu událostí*¹⁵ anebo umět vytvořit časovou značku pomocí hodin, které budou synchronizované v celém DS. Kauzalita událostí nám v tomto případě stačit nebude. Důvodem je to, že systém není schopen garantovat rychlosť odeslání. Tedy například v momentě využití *Lamportových hodin*, které pracují s vektorovými nebo logickými hodinami, nejsme schopni sítí zajistit, že v momentě vzniku události (*stisku tlačítka*) se data rozdistribuují okamžitě. Proto při použití logického času nejsme schopni garantovat pořadí.

Systém tedy musí umět **synchronizovat hodiny** na každém zařízení. K tomuto problému by šlo využít známých protokolů, jako je například *Precision Time Protocol (PTP)*. V případě vhodné¹⁶ implementace broadcastu na síťové vrstvě by šlo synchronizaci značně zjednodušit. Mohla by fungovat tak, že by se v síti zvolil *MASTER OF TIME*, který by jednou za n sekund rozeslal broadcastem čas všem zařízením v síti.

Kromě distribuce a synchronizovaného času DS musí řešit problematiku **distribuce logů** s časovými značkami. Logy je myšlena struktura, které obsahuje časovou značku, typ události v DS a popis události. Pro distribuci těchto logů je možné využít část raft algoritmu diskutovaného v kapitole 5.2.

Pro efektivní fungování síťové vrstvy by bylo vhodné, aby DS implementoval i mechanismus, který by byl schopen **distribuovat** neustále aktualizovaný **seznam zařízení** v DS.

Z krátké úvodní analýzy problému vyplývá, že algoritmus musí v DS řešit tři základní problémy:

- synchronizaci času,
- distribuci logů,
- distribuci seznamu zařízení DS.

¹⁴ Pro hezké otestování a lepší pochopení algoritmu existuje hezká stránka s přehledem a vizualizací raft.github.io.

¹⁵ Vědět jaké událost té druhé předcházela.

¹⁶ Vhodnou implementací se myslí taková, která funguje jako pravý broadcast popisovaný v IEEE 8002.11, nikoli *pseudo broadcast* popisovaný v kapitole 4.3.6.

5.4 Problematika synchronizace času

Požadavek na přesnost synchronizace času je 1 ms. Tato hodnota vychází z reakční doby člověka, která se u špičkových atletů při startu pohybuje okolo 0,1 až 0,25 s. [43] Pokud tedy stanovíme hodnotu o dva řády nižší, nemělo by dojít ke kolizím. V případě kolize, která je vysoce nepravděpodobná, rozhoduje náhoda.

5.4.1 Detailní popis zvoleného algoritmu

Algoritmus aplikuje Cristianův algoritmus, navíc k němu přidává prvky z PTP. Doba přenosu zprávy z uzlu N_A do uzlu N_B trvá dobu D_n s chybou O_n , přičemž platí, že T_A je hodnota hodin v uzlu N_A a T_B je hodnota hodin v uzlu N_B . Pak o přenosu zprávy platí, že:

$$T_A = T_B + D_n + O_n.$$

Při fungování systému nejsme schopni zjistit absolutní hodnotu D_n a O_n , proto počítáme s průměrovanou hodnotou (*aritmetický průměr*) zpoždění \bar{D} a chybou počítanou klouzavým průměrem \bar{O} . Toto zjednodušení způsobuje nepřesnosti při synchronizaci času. Po přidání průměrných hodnot tedy platí, že:

$$T_A = T_B + \bar{D} + \bar{O},$$

kde doba přenosu D_n je počítána jako¹⁷

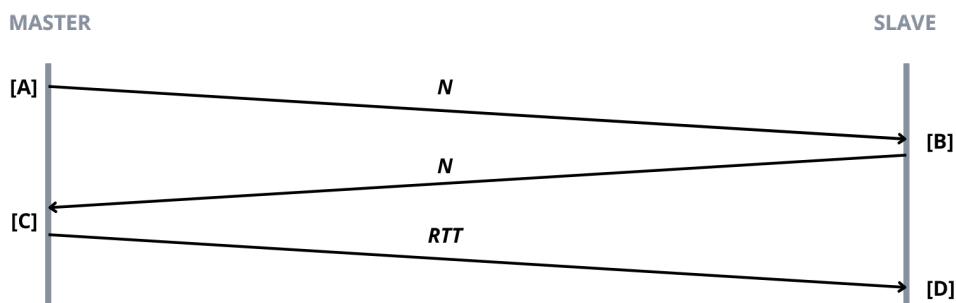
$$D_n = \frac{RTT}{2}$$

a velikost chyby O_n jako

$$O_n = T_B - T_A - \bar{D}.$$

Celkově algoritmus funguje tak, že z uzlu *MASTER* je jednou za 100 ms odešle do všech *SLAVE* uzel zpráva inicializující synchronizaci **doby přenosu** a jednou za 500 ms zprávu inicializující synchronizaci **času**.

Průběh zprávy pro synchronizaci **doby přenosu** je takový, že:



Obrázek 5.2. Schéma synchronizace doby přenosu.

¹⁷ RTT znamená *round trip time*

- [A] Z uzlu *MASTER* se odešle zpráva s lokálním časem do uzlu *SLAVE*.
- [B] *SLAVE* uzel přijme zprávu a se stejným obsahem ji okamžitě odešle zpět do uzlu, z kterého zprávu obdržel. Tedy do uzlu typy *MASTER*.
- [C] Po obdržení času se v uzlu *MASTER* z přijaté hodnoty a aktuálního lokálního času vypočítá doba přenosu jako

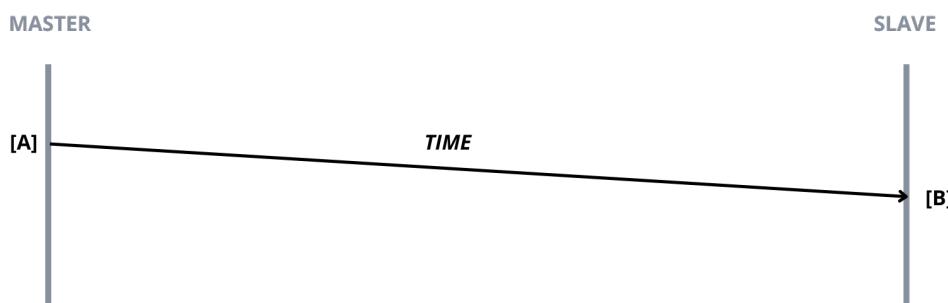
$$D_n = \frac{T_B - T_A}{2},$$

kde T_A je čas odeslaný v situaci [A], tedy na začátku celého procesu a T_B je čas v situaci [C], tedy po přijetí zprávy od uzlu *SLAVE*. Po vypočítání se hodnota odešle do uzlu *SLAVE*.

- [D] Uzel *SLAVE* přijatou hodnotu zapíše do pole, aby mohla být později použita k výpočtu průměrné doby přenosu \bar{D} .

Výpočet doby zpoždění předpokládá, že je doba přenosu symetrická. Tedy, že průměrné odesílání z *MASTER* do *SLAVE* trvá stejnou dobu jako odesílání z *SLAVE* do *MASTER*.

Průběh zprávy pro synchronizaci **času** je takový, že:



Obrázek 5.3. Schéma synchronizace času.

- [A] Z uzlu *MASTER* se odešle zpráva s lokálním časem do uzlu *SLAVE*.
- [B] Po přijetí zprávy dojde k výpočtu průměrné chyby \bar{O} a nastavení lokálního času na uzlu *SLAVE* podle reference z uzlu *MASTER*. Průměrná chyba se počítá s užitím klouzavého průměru jako

$$\bar{O} = \bar{O}O_n = \bar{O}(T_S - T_M - \bar{D}),$$

kde T_M je čas odeslaný z uzlu *MASTER* a T_S je aktuální hodnota času v uzlu *SLAVE*.

Čas se následně nastaví v závislosti na velikosti odchylky. Pokud je větší než konstanta $\bar{O} > |k|$,¹⁸ hodnota hodin je nastavena jako:

$$T_S = T_M + \bar{D} \pm k.$$

V případě, že $\bar{O} \leq |k|$, pak je čas nastaví jako:

$$T_S = T_M + \bar{D} + \bar{O}.$$

¹⁸ Chyba je způsobena především rozdílem rychlosti běhu oscilátorů v zařízeních. Podle výrobce může odchylka dosahovat až ± 10 ppm. Pokud je tedy hodnota chyby větší než cca 100 μ s, jedná se o chybu v době přenosu. Proto v takové situaci nastavíme maximální hodnotu rovnou na velikost konstanty. V našem případě tedy 100 μ s nebo -100 μ s, podle orientace.

5.4.2 Simulace fungování algoritmu

Pro ověření toho, zdali navržený algoritmus splňuje definovanou podmínu přesnosti synchronizace (1 ms), jsem připravil simulaci. Simulace je naprogramovaná v jazyce C. Implementaci si je možné prohlédnout v repozitáři projektu.¹⁹ Simulace se skládá z hlavních třech částí:

- inicializace prostředku a konfigurace parametrů simulace,
- smyčka realizující samotný běh simulace
- a rutina pro vyčištění alokovaných prostředků.

V simulaci je možné **měnit parametry** nastavující:

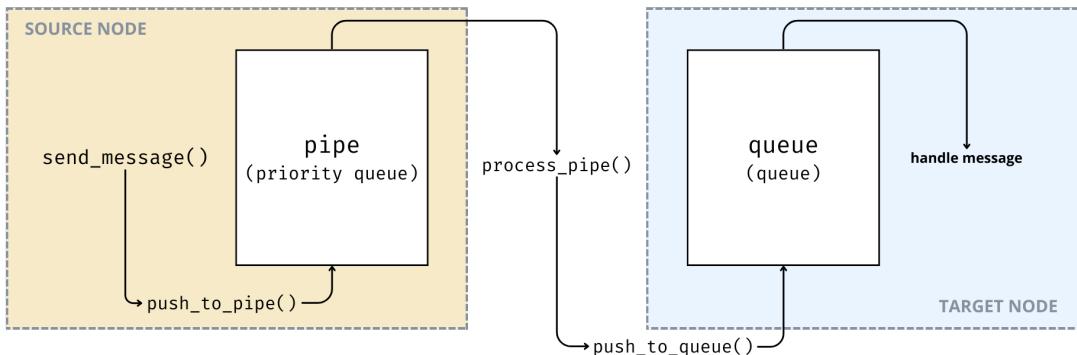
- dobu simulace (udávaná v μ s),
- počet uzlů (maximální počet je 255),
- počáteční čas simulace,
- status uzlu,²⁰
- chybu oscilátoru na každém z uzlů,
- počáteční čas času uzelů, (udávaný v μ s),
- dobu přenosu zprávy mezi uzly,
- velikost pole pro výpočet průměrného zpoždění,
- konstantu pro určení maximální odchylky.

Běh **simulace pak probíhá** jako smyčka omezená dobou. Běh jednoho cyklu představuje dobu 1 μ s. V prvním kroku se nastaví náhodné zpoždění pro daný běh na všech uzlech. Druhý inkrementuje lokální hodiny na všech uzlech a realizuje případnou chybu oscilátoru. Třetí krok na všech uzlech spustí funkci `process_pipe()`, která realizuje zpoždění při odesílání zpráv, resp. zkonzoluje jestli nějaká ze zpráv nedosáhla požadované doby odesílání. Pokud ano, odešle ji do fronty na cílový uzel. Čtvrtý krok tvoří stavový automat, který se stará o odbavení příchozích zpráv na každém z uzlů. Tento proces funguje podle algoritmu pro synchronizaci času popsaného v předchozí kapitole 5.4.1. Pátý krok odesílá inicializační zprávy pro konkrétní procesy, tedy synchronizaci doby přenosu a času. Synchronizace času se spouští jednou za 500 ms a synchronizace doby zpoždění jednou za 100 ms. V případě definování makra `BUILD_REPORT` se v předposledním kroku vypíší hodnoty do log souboru, ze kterého jsou následně generovány grafy. Pro efektivitu velikosti logovacího souboru jsou vypisovány pouze cykly, které obsahují změnu. Pro každý uzel krom uzelu *MASTER* se vypisují tři hodnoty, konkrétně průměrné zpoždění, velikost chyby a rozdíl času v uzelu *MASTER* a daného *SLAVE* uzlu. Na závěr každého běhu cyklu je inkrementovaný čas běhu simulace.

Odesílání zpráv se zpožděním je realizováno pomocí prioritní fronty nazývané *pipe* a fronty obsahující příchozí zprávy. Každý uzel má své tyto dvě struktury. Celý proces je ilustrován v schématu 5.4 a probíhá tak, že dojde k odeslání zprávy pomocí funkce `send_message()`. Tato funkce alokuje paměť pro novou zprávu a vloží zprávu do prioritní fronty *pipe*. Prioritu určuje doba odesílání, konkrétně čím nižší hodnota, tím dříve bude zpráva zpracována. Pokud už uplynula doba odesílání, zpráva je vyjmuta z *pipe* a vložena do *queue* na cílovém uzlu, na které jsou uchovávány příchozí zprávy.

¹⁹ Tato simulace je uložena konkrétně na adrese <https://github.com/petrkucera/rafting-button/tree/main/code/simulation>.

²⁰ Standartě je jeden z uzlů v režimu *MASTER* a ostatní v režimu *SLAVE*.



Obrázek 5.4. Schéma odesílání zpráv v simulaci.

Výsledky simulace je možné vizualizovat pomocí třech python skriptů, které se nacházejí v repozitáři simulace.

První `visualize0.py` zobrazuje velikost chyby \bar{O} v závislosti na čase pro 3 *SLAVE* uzly, umožňuje zobrazit maximální a minimální akceptovatelnou chybu a střední hodnotu všech vypsaných hodnot.

Druhý skript `visualizeRTT.py` zobrazuje průměrnou hodnotu doby přenosu \bar{D} v závislosti na čase pro 3 *SLAVE* uzly a umožňuje zobrazit střední hodnotu všech vypsaných hodnot.

Třetí soubor se souborem `visualizeTIME.py` zobrazuje rozdíl uzlu *MASTER* a *SLAVE* v závislosti na čase simulace pro 3 *SLAVE* uzly a umožňuje zobrazit střední hodnotu vypsaných hodnot.

5.4.3 Získání parametrů pro simulaci

Pro spuštění simulace je třeba znát latenci, resp. reálnou dobu odesílání zpráv mezi zařízeními. K získání této hodnoty jsem připravil zapojení pro měření, které funguje tak, že propojuje ESP32 zařízení vodičem, který synchronizuje jejich vnitřní hodiny a je ho možné dohledat v repozitáři projektu.²¹

Měřící sestava obsahuje 3 ESP32 zařízení a jedno STM32.²² STM32 generuje pomocí PWM pulz, který slouží k synchronizaci hodin v DS. ESP32 mezi sebou vysílájí zprávy, pomocí nichž se spočítá doba latence.

Měření latence probíhá přesně následujícím způsobem.

- Zařízení A odešle do zařízení B a C zprávu s časem odeslání.
- Zařízení B a C zprávu příjemou a spočítají latenci, tedy dobu od odeslání (přesněji zapsání synchronizovaného času do zprávy) do přijmutí zprávy (spuštění callback funkce a provedení výpočtu latence). Latence se počítá jako:

$$t_l = T_{B,C} - T_A,$$

kde T_l je velikost latence, T_A čas ve zprávě, resp. čas odeslání a $T_{B,C}$ je čas přijmutí zprávy.

Čas je v ESP32 synchronizován pomocí pulzů z PWM generovaného STM32. To tak, že při registraci náběžné hrany na vstupním pinu (nastaveno na `GPIO_NUM_21`)

²¹ <https://github.com/petrkucera/rafting-button/tree/main/code/esp-now-parameters>

²² Konkrétně se jedná o STM32G431KB, které se na ČVUT FEL používá k výuce předmětu LPE. Schéma a realizace zapojení je dostupné v příloze D.14 a D.13.

se generuje přerušení (ISR), které uloží aktuální hodnotu globálního času od spuštění a prohlásí ji za výchozí čas synchronizace (T_s). Čas DS se následně počítá jako

$$T_{DS} = T_g - T_s,$$

kde T_{DS} je čas synchronizovaný v DS, T_g je doba běhu daného zařízení a T_s je čas synchronizace času.

Čas je synchronizován s průměrnou odchylkou okolo 173 μ s. Tato hodnota byla určena měřením, kdy je generován pulz o frekvenci 1 Hz, perioda je tedy 1 s. A *ISR handler* funkce vždy spočítá čas od předchozího běhu a vynuluje T_s . Zpoždění je dánou režijními náklady běhu procesoru. Maximální odchylka byla určena také měřením a při běhu 24 min a 18 s dosáhla maximální hodnoty 810 μ s.²³

K měření jsem využil obou velikostí ESP32 modulů a jejich doba odesílání se nelišila.

Měření jsem spustil ve dvou scénářích. Lišily se pouze časem běhu. Scénář A trval přibližně 6,5 min a scénář B přibližně 37 min. Výsledné hodnoty jsou totožné, pouze se liší množstvím odeslaných a přijatých zpráv.

měření	zařízení	mean	max	min
A	COM6	1308.59 μ s	14944 μ s	899 μ s
A	COM7	1256.00 μ s	19471 μ s	883 μ s
B	COM6	1152.03 μ s	13239 μ s	893 μ s
B	COM7	1134.67 μ s	25143 μ s	871 μ s

Tabulka 5.1. Parametry měření latence.

Pokud se na naměřená data chceme podívat z pohledu četnosti pro jednotlivé zpoždění, bude nejlepší využít histogramů v příloze D.15 a D.16. Histogram zobrazuje oblast od 900 μ s až po 1200 μ s. Počet zpráv pro vyšší zpoždění je oproti tomuto intervalu zanedbatelný, proto vyšší zpoždění není zobrazeno.

Zajímavé jsou vždy 2 oblasti vyšší četnosti. Ty si i na základě detailního studování logů²⁴ vysvětlují tak, že vždy první odeslání trvá delší dobu. Testovací vysílání totiž funguje tak, že z uzlu *MASTER* vždy odešle 3x2 zprávy, resp. 3x odešle zprávu do uzlu COM6 a COM7. První odesílání je vždy do uzlu COM6. Poté se počká na synchronizaci času a měření se opakuje.

Z měření vyplývá, že **průměrná doba zpoždění unicast** vysílání se pohybuje mezi 900 μ s a 1050 μ s.²⁵

5.4.4 Simulace s reálnými parametry

Simulace byla prováděna s konfigurací, kdy

- doba simulace byla 10 min (bylo provedeno 6000 synchronizací doby odesílání a 2000 synchronizací času),

²³ Původně jsem se domníval, že overhead je veliký především kvůli funkci `esp_timer_get_time()` z důvodu dohledané diskuse <https://esp32.com/viewtopic.php?t=16228>. Experimentem jsem ale ověřil, že tato funkce není hlavní brzdou při měření. Prodlevu se mi ale povedlo snížit přetaktováním na 240 MHz a optimalizací kódu.

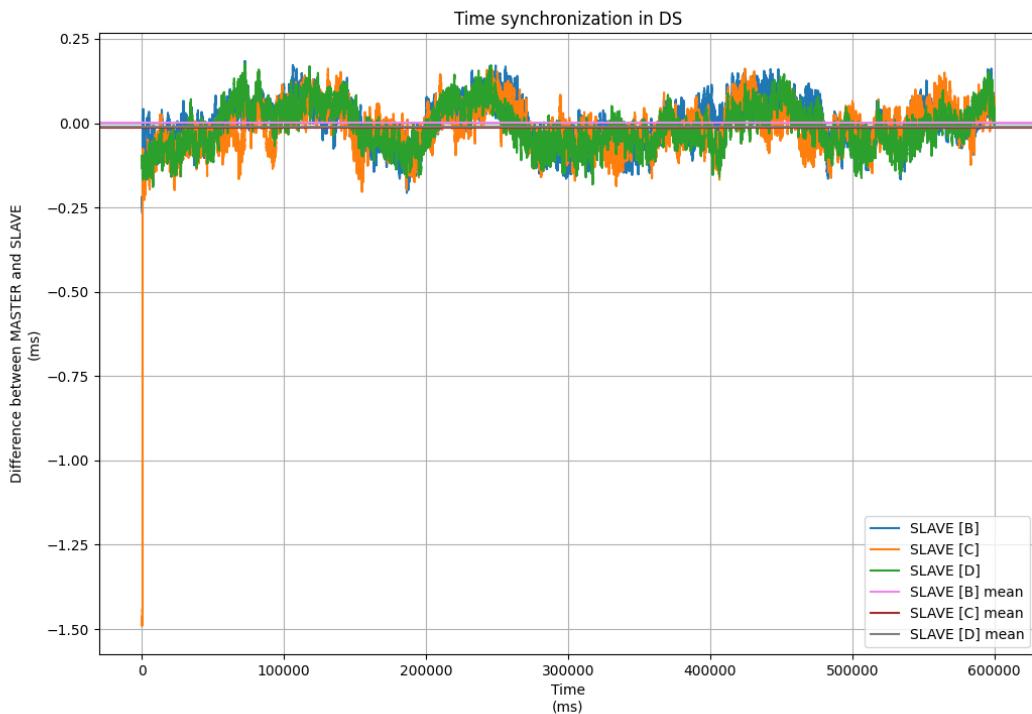
Nepřesnost v tomto měření může způsobit i skutečnost, že nevím, s jakou přesností je generován referenční signál.

²⁴ Možné dohledat v repozitáři projektu.

²⁵ Dle skriptu `distribution.py` víme, že přesně 86,7% zpráv byla odeslána v tomto časovém intervalu.

- pro simulaci byly užity 4 uzly,
- *MASTER* uzel začínal v čase nula, ostatní uzly na náhodném čase v rozmezí 100 µs až 1500 µs,
- doba zpoždění zpráv byla náhodně zvolena pro každé odesílání, a to v rozsahu od 900 µs až po 1050 µs
- a chyba oscilátorů se pohybovala náhodně mezi maximální chybou ± 1 ppm až ± 10 ppm.

Jsem si vědom, že simulace nezohledňuje veškerou problematiku. Rozdělení odesílání zprávy není náhodné, nýbrž se chová dle výsledků kapitoly 5.4.3. Teoreticky při běhu může také dojít k tomu, že jedna situace nastane vícekrát. To v případě, kdy během synchronizace bude čas nastaven dozadu. Jsem si této chyby vědom. Nicméně se v mém případě nejedná o kritickou aplikaci a synchronizace dosahuje takových výsledků přesnosti, že je vysoce nepravděpodobné, že by k této chybě došlo.



Obrázek 5.5. Simulace fungování algoritmu pro synchronizaci času - rozdíl času.

Výsledky chování simulace zobrazují 3 grafy. První z nich v příloze E.17 ukazuje průměrnou chybu \bar{O} v závislosti na čase a střední hodnotu chyb. Ty se pohybují v blízkém okolí nuly, jak bychom očekávali. Druhý graf, také dostupný v příloze E.18, zobrazuje průměrnou hodnotu doby přenosu \bar{D} v závislosti na čase. Vidíme, že k synchronizaci dojde poměrně rychle. Třetí a pro splnění stanovených parametrů nejvíce důležitý graf 5.5 zobrazuje rozdíl času mezi *MASTER* uzlem a *SLAVE* uzlem. Střední hodnota tohoto ukazatele je okolo nuly a simulace dosahuje maximální odchylky do $\pm 0,25$ ms, což hraje splňuje náš limit 1 ms.

Ze simulace vyplývá, že navržený algoritmus dle simulace **splňuje stanovený požadavek přesnosti synchronizace času**, tedy 1 ms.

■ 5.4.5 Měření přesnosti synchronizace času

Při ověřování modelu algoritmu na reálném hardwaru jsem narazil na nepřesnost, která byla způsobena režimem FreeRTOS, kterou jsem v simulaci nezohlednil.²⁶ Proto jsem provedl následující **modifikaci algoritmu** a způsobu zpracování ESP-NOW událostí s cílem dosáhnout požadované přesnosti.

- Chyba se **nepočítá klouzavým průměrem** ale pouze z aktuální hodnoty.
- Čas se nastavuje stejně do momentu, než je chyba menší než konstanta E . Jakmile je menší, **čas se opravuje o velikost konstanty K** a to tak, že pokud je chyba O větší než K tak

$$T = T - K$$

a pokud je chyba O menší než $-K$ tak

$$T = T + K,$$

kde T je čas zařízení *SLAVE*. Ve skutečnosti je implementace na reálném zařízení trochu jiná, protože čas v DS se na zařízení počítá jako

$$T_{DS} = t_{local} - c,$$

kde T_{DS} je čas DS na daném zařízení, t_{local} je doba běhu zařízení od jeho spuštění a c je proměnná korekce, kterou se snažíme natavit tak, aby byl čas v celém DS stejný. Korekce o konstantu K ve výsledku probíhá tak že, pokud platí $O \notin (-K, +K)$, pak $c = c \pm K$ dle orientace.

- Při přijmutí ESP-NOW události je událost označena **časovou značkou (*timestamp*)**, která se následně využívá k výpočtu. Toto minimalizuje chyby způsobenou během FreeRTOS.

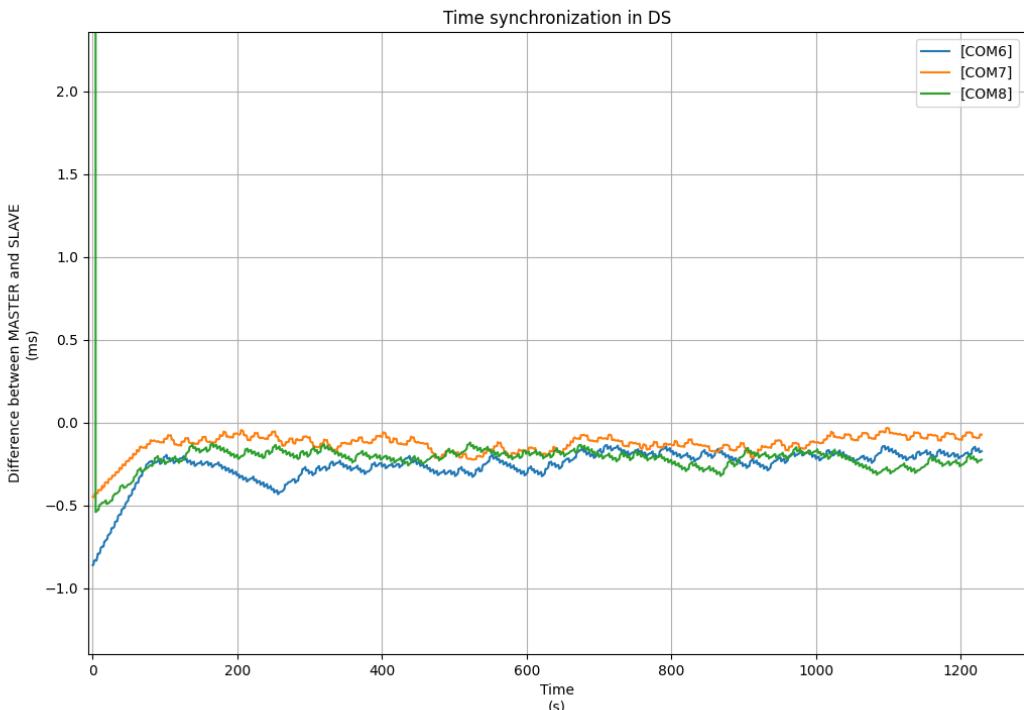
Tyto úpravy mi pomohly zajistit dostatečnou přesnost měření, jak vyplývá z naměřených dat, které si je možné prohlédnout na obrázku 5.6 a v příloze E.2. **Algoritmus pro synchronizaci času splňuje požadované parametry je vhodný pro využití v implementaci.**

Měření bylo prováděno s téměř identickým zapojením²⁷ jako měření latence popsané v kapitole 5.4.3. Přidal jsem pouze moduly a změnil velikost odporů na vstupu referenčního signálu. Reference generovaná ze STM32 byla použita pro sjednocení času výpisů. Zdrojové kódy využité pro měření včetně skriptů pro vizualizace jsou dostupné v repozitáři projektu.²⁸

²⁶ Výsledky měření bez úpravy algoritmu si je možné prohlédnout v příloze E.20.

²⁷ Schéma zapojení je dostupné v příloze E.22.

²⁸ <https://github.com/petrkucera/rafting-button>



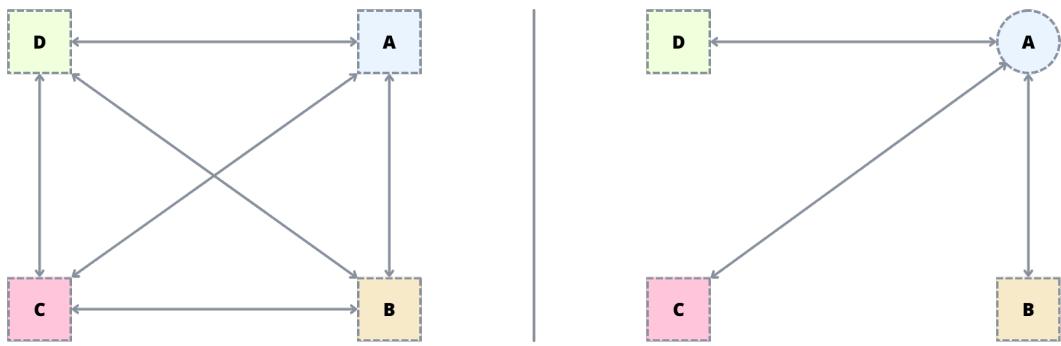
Obrázek 5.6. Měření synchronizace času - rozdíl času.

5.5 Distribuce logů a seznamu zařízení DS

Z rozboru problému vyplývá, že vhodným způsobem pro určení pořadí stisku tlačítka je ke každé události přiřadit časovou značku (*timestamp*) a pomocí ní určit kauzalitu jednotlivých událostí. **Časová značka** bude vždy v celém DS systému **jedinečná**. To vyplývá z předpokladu z kapitoly 5.4, že rozlišovací schopnost systémů je mnohem vyšší než reakční doba člověka. Pokud by přeci jenom ke konfliktu došlo, rozhodne v daném případě náhoda.

Pokud se událost stisku na daném zařízení podepíše časovou značkou, zbývá ji ze zařízení rozdistribuovat do celého DS. Tento problém je **ekvivalentní** problému distribuce seznamu sousedů v celé síti, proto je budu řešit v rámci jednoho algoritmu.

Zásadní otázkou, kterou si je třeba položit a zdůvodnit, je, zdali **má komunikace probíhat tak, že každé ze zařízení rozdistribuuje událost do celé sítě samo nebo k tomu využije zvoleného lídra**, tedy zařízení typu *MASTER*.



Obrázek 5.7. Srovnání komunikační náročnosti při distribuci logů.

Komunikační zátěž L pro jedno kolo, tedy počet zpráv, které je nutné odeslat pro hlasování všech zařízení, bude pro scénář bez lídra vyžadovat větší počet zpráv, konkrétně

$$L = N(N - 1),$$

kde N je počet zařízení. Oproti tomu využitím řídícího zařízení, bude třeba menší počet zpráv, konkrétně

$$L = 2(N - 1).$$

Mohlo by se zdát, že je vhodnější využít model s lídrem. To neplatí vždy. V případě vyššího počtu zařízení by mohlo dojít k přetížení lídra. Se stanovenými podmínkami v této práci **k přetížení nedojde**. Pro deset zařízení je třeba při jednom kole odeslat 18 zpráv a na základě výsledků měření síťové infrastruktury v kapitole 4.5 je zařízení schopné odeslat až 20 zpráv za 1 s, a to i s drobnými pauzami. Další otázkou vyžadující komentář je, za jak dlouho dojde k přijmutí zprávy na cílovém zařízení. V našem případě po algoritmu požadujeme pouze **konečnou živost**, tedy nepotřebujeme omezit dobu, do které informace přijde. Potřebujeme mít pouze jistotu, že informace přišla, a to je garantováno vlastností komunikace a algoritmu. Využití *MASTER* zařízení je tedy vhodným způsobem.

Každé ze zařízení bude uchovávat lokální **strukturu s logy**, které budou řazeny podle časové značky. Struktura bude omezena velikostí, tj. staré události budou přepsány novými. Samotný proces distribuce logů bude probíhat následujícím způsobem.

- Na zařízení dojde k vzniku události. Událost se přidá do lokálního logu a odešle se do *MASTER* zařízení.
- *MASTER* zařízení si zprávu přidá do svého lokální logu a odešle ji na všechny *SLAVE* uzly.
- *SLAVE* uzel si přijatou zprávu uloží do lokálního logu.

Obdobně bude probíhat i distribuce seznamu sousedů. Bude k ní využíváno řídící zařízení z důvodu ušetření zpráv. V momentě, kdy *MASTER* obdrží zprávu se sousedy, rozešle ji na všechny známé zařízení, kteří si svůj seznam zaktualizují.

5.6 Návrh celkového algoritmu

V předchozích kapitolách byly diskutovány dílčí problémy běhu celého systému a známé algoritmy, které lze aplikovat. Výsledný algoritmus spojuje dohromady Raft a již vyřešené úlohy týkající se synchronizace času a distribuce logů.

Hlavní algoritmus tvoří tři **základní komponenty**:

- registrace zařízení do DS,
- běžný chod
- a terminace zařízení registrovaného v DS.

5.6.1 Registrace zařízení

Jakmile se zařízení spustí, pošle *broadcast* zprávu typu `HELLO_DS`. Zařízení v síti, která zprávu přijmou, si ho přidají na svůj seznam sousedů a zpět odešlou zprávou typu `NEIGHBOURS`. Do ní také přiloží základní informace o DS, jako je *ID epochy*. Jakmile zařízení obdrží více jak polovinu odpovědí. Přechází do další fáze – běžného chodu DS.

5.6.2 Běžný chod

Běžný chod se skládá z **epoch**. Každá epocha má svoje ID a tvoří ji dvě fáze – *volba lídra* a *běžný provoz*.

Zařízení bude označovat stejně jako v Raftu.

- *Lídř* (*MASTER*) funguje jako centrální prvek, který rozesílá logy událostí DS, seznamy sousedů a synchronizuje čas.
- *Následovník* (*SLAVE*) je pasivní zařízení, které se chová běžným způsobem.
- *Kandidát* je přechodná role v průběhu volby lídra.

První fází každé epochy je vždy **volba lídra**. Volba lídra se koná v každé epizodě, ovšem nemusí skončit úspěšně. Probíhá velice podobně jako volba lídra v algoritmu Raft. *Lídř* si udržuje svoji autoritu pomocí rozesílání zprávy pro synchronizaci času `TIME`. Pokud zařízení neobdrží zprávu do *timeoutu* t_{sync} , zařízení zvýší číslo epochy a spustí nové volby. Ty probíhají tak, že rozešle na všechny známé aktivní sousedy zprávu `REQUEST_VOTE`, tedy žádost o to, že chce být *lídrem*. Poté může nastat jedna ze těchto tří situací:

- zařízení dostane **potvrzení od většiny** `GIVE_VOTE` aktivních sousedů a stane se novým *lídrem*
- nebo **přijme zprávu synchronizující čas** `TIME`, novým *lídrem* se stalo nějaké zařízení rychleji
- nebo budou **volby neúspěšné** do *timeoutu* $t_{election}$, volby skončí neúspěchem a začne nová epocha.

Po úspěšných volbách probíhá fáze **běžného provozu**. Pokud nejsou volby úspěšné, nemusí k ní dojít. Během ní DS distribuuje logy událostí pomocí mechanismu popsáного v kapitole 5.5 a zprávy typu `LOG`. Při registraci zařízení do DS nebo terminaci rozesílá seznam sousedů. A lídr posílá synchronizační zprávy `TIME` pro udržení autority *lídra* a synchronizaci času v DS.

Pokud v této aplikaci dojde k selhání zařízení, neexistuje žádná opravná rutina, a to z podstaty systému a navrženého algoritmu. Respektive neexistuje stav, který by algoritmus neřešil a šlo ho softwarově opravit. Pokud dojde k selhání zařízení, uživatel bude informován pomocí stavu zařízení.

Chyba ožití starého *lídra* je ošetřena mechanismem *ID epochy*. Zařízení akceptuje pouze zprávy s aktuálním nebo vyšším *ID epochy*.

5.6.3 Terminace zařízení

Jako **neaktivní zařízení** je označeno každé, které nepřijme více jak tři zprávy úspěšně. Pokud neodpovídá, je terminováno z DS. Respektive je v seznamu sousedů označeno jako neaktivní a tato informace je pak rozeslána přes *lídra* do celého DS.

Kapitola 6

Zařízení z pohledu UX

Ve své práci jsem se okrajově věnoval i návrhu zařízení z pohledu UX.¹ Na základě pravidelných diskuzí s kolegy z univerzity, přáteli a vedoucím mojí práce vyplynuly v podstatě dva možné koncepty.

V prvním případě se jednalo o zařízení **menšího** rozměru, které by si mohl uživatel vzít do ruky a ovládat ho stiskem tlačítka. Druhou možností bylo vyrobit **robustnější** krabičku, kterou by šlo položit na stůl. Kdokoliv z uživatelů by mohl hlasovat například i pomocí silného úderu.

Původně jsem chtěl připravit dva prototypy, každý pro jednu verzi. Pomocí nich bych následně provedl výzkum na základě uživatelského zážitku. Z časových důvodů a také důvodu toho, že se daná práce orientuje na jinou oblast, jsem se po doporučení vedoucího práce rozhodl připravit pouze krátký formulář s cílem zjistit mínění lidí.

Formulář jsem koncipoval jednoduše. Na začátku popisoval motivaci a dva koncepty, které jsem doplnil o ilustrační obrázky vytvořené modelem AI.² Následně jsem položil otázku, jakou z možností by korespondenti volili. Odpověď bylo možné doplnit komentářem.



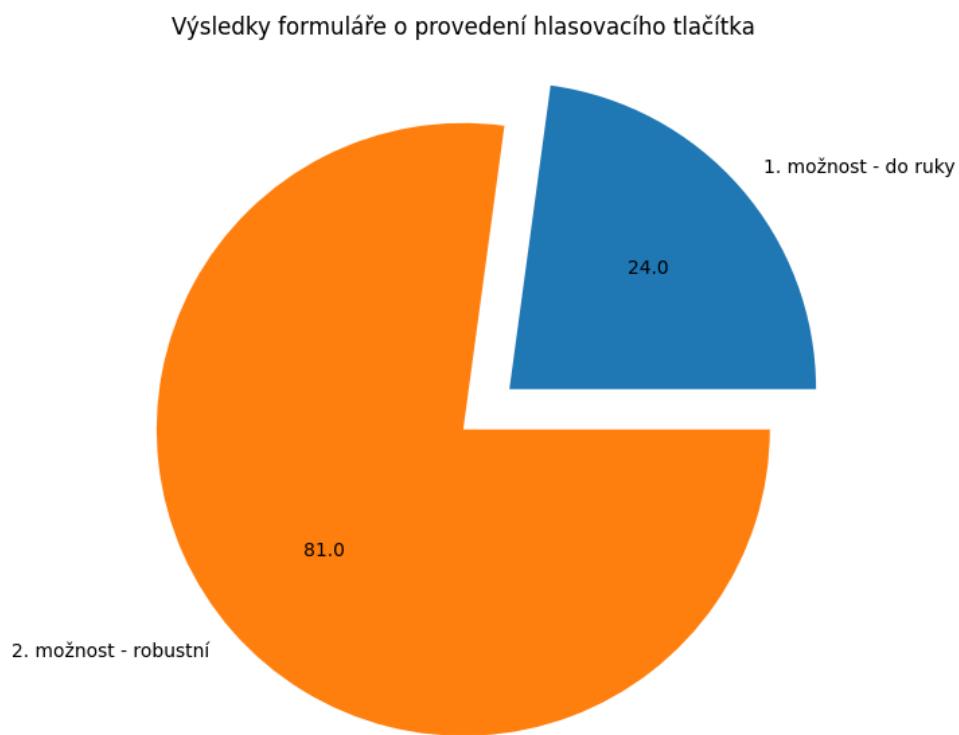
Obrázek 6.1. Ilustrační obrázky obalu vygenerované modelem AI.

Formulář jsem sdílel ve story na svém instagramovém profilu. Za 24 hodin mi na formulář odpovědělo 105 lidí, přičemž 44 doplnilo svoji volbu o slovní komentář. Jsem si vědom, že lidé, kteří odpovídali, jsou z mého okolí. Tato skupina je z většiny tvořena mladými lidmi ve věku od 16 do 30 let, kteří studují nebo čerstvě dokončili vysokou školu. Jsou mezi nimi jak lidé z technických oborů, tak ale i lidé

¹ User Experience - uživatelského zážitku

² Použil jsem model *DALL-E* napojený na aplikaci *Microsoft Designer*.

z oblasti školství či zdravotnictví. Jelikož nepracuji s reprezentativním vzorkem, rozhodně nelze data považovat za zcela empirická. Domnívám se, že se ale mohu opřít o slovní komentáře, které jsou přiloženy v příloze F.



Obrázek 6.2. Graf výsledků z průzkumu podoby zařízení.

Z výsledků můžeme vyvodit, že lidé spíše preferují jedno větší zařízení pro celý tým.

Kapitola 7

Implementace a testování

V této kapitole nastíním implementaci a shrnu výsledky testování.

7.1 Implementace

Kód jsem implementoval v jazyce C s použitím **ESP-IDF**.¹ Jedná se o oficiální vývojový framework pro procesory ze sérií ESP32, ESP32-S, ESP32-C a ESP32-H. Framework se skládá z jednotlivých komponent, jako je například RTOS kernel, který využívá *FreeRTOS*, ovlaďače periferií, wifi či komponenta pro optimalizaci spotřeby energie. Většina komponent je vyvíjena jako *open-source*. ESP-IDF také podporuje balíčkový systém,² který umožňuje pokročilé verzování a efektivní zacházení s jednotlivými komponenty.

Jedním z požadavků na zařízení bylo, aby se jednalo o **autonomní systém**. Výsledný software je tedy na všech zařízeních z počátku ve stejné konfiguraci.

Z důvodu více jader procesoru a využití *callback* funkcí v implementaci využívám **multitaskové infrastruktury**, kterou zajišťuje *FreeRTOS*. Běh systému je složen z následujících procesů.

- Rodičovský proces `app_main` inicializuje nutné komponenty a stará se o správu celého systému. Také se využívá pro vypisování pravidelných hlášení o stavu systému.
- Proces `espnow_handler_task` slouží k zpracování *ESP-NOW udláostí*. To jsou události vyvolané *callbacky*, které signalizují odeslání nebo přijmutí zprávy, o nich více v kapitole 4.3.4. Proces funguje jako stavový automat a je detailněji popsán v další kapitole.
- Pokud je zařízení *lidrem*, pak proces `send_rtt_cal_master_task` rozesílá zprávu inicializující výpočet doby přenosu D . Pokud lídrem není, nic neprovádí.
- Proces `send_time_task` funguje podobně jako předchozí proces. Tedy pokud je zařízení *lidrem*, pak rozesílá zprávy synchronizující čas. Tím si *lidr* udržuje svoji autoritu.
- Pokud zařízení po dobu t_{sync} neobdrželo zprávu s časem od *lidra*, proces `send_request_vote_task` inicializuje nové volby a následně je obslouží.
- Posledním uživatelsky zavedeným procesem je `handle_ds_event_task`. Ten se stará o distribuci logů mezi zařízeními.
- V systému probíhají i jiné systémové obslužné rutiny, například zajišťující chod wifi.

Software implementuje 3 **přerušení**, respektive dvě obsluhy přerušení a jedno přerušení. Konkrétně se jedná o *callback* funkce `espnow_send_cb` a `espnow_recv_cb` obsluhující přerušení ESP-NOW a přerušení `gpio_handler_isr` pro obsluhu hardwarového tlačítka připojeného na bránu `GPIO_NUM_23`.

¹ Espressif IoT Development Framework

² Balíčkový systém se nazývá *IDF Component Registry*.

Priorita jednotlivých procesů je popsána tabulkou 7.1. Nejvyšší prioritu mají přerušení a rutiny obsluhující přerušení. Po nich následuje `espnow_handler_task`, protože se stará o časově kritické procesy jako je synchronizace času. Střední prioritu mají všechny ostatní procesy krom rodičovského, který má prioritu nejnižší.

proces	priorita
<code>app_main</code>	1
<code>espnow_handler_task</code>	3
<code>send_rtt_cal_master_task</code>	2
<code>send_time_task</code>	2
<code>send_request_vote_task</code>	2
<code>handle_ds_event_task</code>	2

Tabulka 7.1. Priority procesů.

7.1.1 Struktura rámce zprávy

Z měření síťové infrastruktury vyplývá,³ že **velikost zprávy má vliv na dobu odeslání**. Algoritmus v ideálním případě očekává symetrickou dobu odesílání zpráv. Proto jsem při implementaci využil maximální velikost rámce a nevyužité místo jsem zaplnil náhodnými daty. Díky tomu nebude mít rozdílná velikost vliv na případnou asymetričnost.

Z důvodu zjednodušení využívám pouze **jediný typ rámce** pro všechny typy zpráv. Rámce se skládají z následujících částí, které jsou vizualizovány obrázky 7.1 a 7.2.

- **Typ zprávy** (`message_type_t`) ovlivňuje stavový automat v procesu, který zpracovává příchozí zprávy.
- **ID epochy** (`uint32_t`) říká o jakou epochu se jedná, a tak zajišťuje bezpečnost.
- **Číselný obsah** (`uint64_t`) používá se k přenášení různých typů zpráv. Ve většině případů obsahuje časovou značku.
- **Typ události DS** (`ds_event_t`) specifikuje, zdali se jednalo o stisk nebo reset. Využívá se k distribuování logu.
- **Mac adresa události** (`uint8_t [ESP_NOW_ETH_ALEN]`) se využívá k distribuování logu, konkrétně ke zdrojové adrese dané události.
- **Úkol události** (`ds_task_t`) se využívá k distribuci logů.
- **Sousedé** (`neighbour_t [NEIGHBOURS_COUNT]`) je pole obsahující seznam sousedů. Požadavek na systém definuje maximálně 10 připojených zařízení. Do pole se neukládá informace o zařízení, kterému seznam náleží. Proto je velikost pole 9. Schéma struktury souseda je zobrazeno obrázkem 7.2.
 - **Titul** (`device_title_t`) určuje, zdali je zařízení *lídr* nebo *následovník*.
 - **Status** (`device_status_t`) je informace o stavu zařízení, tj. zdali je aktivní či nikoli.
 - **Mac adresa** (`uint8_t [ESP_NOW_ETH_ALEN]`) identifikuje souseda pomocí jeho mac adresy.
 - **Výplň** (`uint8_t`) neboli payload vyplňuj nevyužitý prostor náhodnými daty.

³ Možné dohledat v kapitole 4.5.

Typ zprávy	ID epochy	Číselný obsah	Typ události	Mac adresa události	Úkol události	Sousedé	Výplň
4 bytes	4 bytes	8 bytes	4 bytes	8 bytes	4 bytes	9 x 16 bytes	250 - 174 bytes

Obrázek 7.1. Struktura rámce zprávy.

Titul	Status	Mac adresa
4 bytes	4 bytes	8 bytes

Obrázek 7.2. Struktura rámce zprávy - sousedé.

7.1.2 Proces zpracování zpráv

Jak již bylo uvedeno v kapitole 7.1.1, zprávy zpracovává speciální mechanismus. To tak, že *callback* funkce zaznamená novou zprávu, pošle ji jako událost do fronty a z ní proces `espnow_handler_task` vyjme událost a tu následně odbaví.

Tento **proces je klíčovým** pro fungování celého systému. Funguje jako stavový automat. Přijatou událost, respektive zprávu zpracuje podle jejího typu. Pokud se jedná o zprávy typu:

- **HELLO_DS** zařízení ověří, zdali se jedná o neznámé zařízení. Pokud ano, přidá ho do svého seznamu. Pokud již zařízení zná, pouze nastaví jeho status jako aktivní. Jako odpověď mu pošle seznam všech zařízení v síti.
- **NEIGHBOURS** zařízení si přidá všechny neznámé sousedy do svého listu a aktualizuje status a titul u jednotlivých zařízení.
- **RTT_CAL_MASTER** zařízení odešle zprávy zpět odesílateli se stejným obsahem. Tato zpráva se využívá k výpočtu doby přenosu D .⁴
- **RTT_CAL_SLAVE** zařízení vypočítá dobu přenosu a odešle ji odesílateli zprávy.⁵
- **RTT** zařízení uloží hodnotu přenosu do pole, aby bylo možné vypočítat průměrnou dobu přenosu. Pokud se jedná o první takovou informaci, pole se touto hodnotou vyplní celé.
- **TIME** zařízení spočítá chybu synchronizace času O a nastaví proměnnou konstantu c pro výpočet času T_{DS} podle algoritmu popsaného v kapitole 5.4.5.

Zpráva tohoto typu se využívá i k tomu, aby si *lídř* udržel svoji autoritu. Pokud je tedy zařízení ve stavu *kandidáta* a přijme daný typ zprávy, ukončí svoji kandidaturu a přepne se do stavu *následovník*.

Při každém zpracování této zprávy se ukládá časová značka. Ta se následně využívá k výpočtu *timeoutu* t_{sync} . Pokud je delší než stanovená konstanta, zařízení přechází do stavu *kandidát*, zvyšuje číslo epochy (*epoch ID*) a inicializuje volby.

- **REQUEST_VOTE** zařízení odešle odesílateli zprávu s hlasem.
- **GIVE_VOTE** zařízení zprávu uloží. Pokud má více odpovědí, než je polovina aktivních zařízení v síti, prohlásí se za *lídrem* a ostatní zařízení za *následovníky*.
- **LOG2MASTER** zařízení pošle log do fronty, která předává tento typ událostí procesu `handle_ds_event_task`. Log je uložen a rozesán všem *následovníkům*.
- **LOG2SLAVES** zařízení pošle log do fronty, která předává tento typ událostí procesu `handle_ds_event_task`. Log je uložen.

⁴ Více informací o dané problematice je dostupné v kapitole 5.4.1.

⁵ Více informací o dané problematice je dostupné v kapitole 5.4.1.

V případě, že proces obdrží události s informací o **neúspěchu odeslání zprávy**, se inkrementuje součet neodeslaných zpráv v daném zařízení. Pokud jich je více než počet povolený konstantou `COUNT_ERROR_MESSAGE_TO_INACTIVE`, proces je prohlášen za neaktivní a tato informace je rozdistribuoována na všechny aktivní sousedy.

7.1.3 Poznámky k implementaci

Kód je doplněn o ladící výpisy definované v ESP-IDF. Konkrétně se jedná o aplikaci funkcí `ESP_LOGI`, `ESP_LOGW` a `ESP_LOGE`.

Veškerý kód je dostupný v repozitáři projektu.⁶ Kód je částečně komentován. V nejbližší době také doplním podporu pro generování Doxygen dokumentace.

7.2 Testování

Základní komponenty algoritmu pro synchronizaci času, doby přenosu a velikosti chyby byly otestovány v kapitole 5.4.5. Systém je schopen synchronizovat čas s minimální přesností 1 ms.

Ve **výsledné implementaci** jsem prováděl testování pro dílčí problémy, jako je nalezení zařízení v okolí, distribuce seznamu sousedů, volby lídra a synchronizace času. K testování jsem využil ladících výpisů a zkoušel všechny možné scénáře. Testování bylo prováděno maximálně s 5 zařízeními.⁷ Z testování vyplývá, že **systém je úspěšně schopný**:

- najít zařízení v okolí,
- distribuovat seznam sousedů, který je možné plně aktualizovat,
- zvolit lídra,
- řešit problematiku pozdního připojení zařízení do sítě,
- vyřešit problém, když zařízení přestane odpovídat, respektive vypne se,
- synchronizovat čas v celém DS s přesností 1 ms.

Kompletní otestování systému, včetně distribuce logu při reálné hře, jsem z časových důvodů v této práci nestihl. V nejbližší době tento test provedu a jeho výsledky představím při obhajobě práce a přidám je do repozitáře projektu.

Testování plánuji provádět pomocí zařízení STM32G431KB, používaného již v předchozích měřeních. Pomocí něj budu simulovat stisky tlačítek. Testování bude obsahovat několik scénářů. První bude mít za úkol otestovat normální průběh hry. V druhém se budu snažit stanovit časový limit, při kterém je zařízení stále schopno správně rozeznat kauzalitu dvou událostí, které nastanou v bezprostřední blízkosti. Třetí scénář bude testovat potenciálně problematické situace. To především vliv odpojení zařízení ze sítě a rušení sítě jiným signálem.

⁶ <https://github.com/petrkucera/rafting-button/>

⁷ Historie testování i s komentáři je dostupná v repozitáři projektu.

Kapitola 8

Závěr

Cílem práce bylo vytvořit koncept zařízení, které bude schopné uspořádat události distribuovaného systému tak, aby byla zachována jejich kauzalita. Důraz byl kladen především na algoritmickou část projektu.

Po prostudování teoretické problematiky možností bezdrátových sítí pro IoT, jsem se věnoval volbě vhodného hardwaru, definici požadavků na síťovou infrastrukturu, jejímu vybudování a ověření požadovaných parametrů. Dále jsem studoval známé algoritmy distribuovaných systémů a navrhl vlastní algoritmus, který jsem následně implementoval na zařízení a částečně prováděl jeho testování.

Systém je schopen uspořádat události s minimální rozlišovací schopností 1 ms 5.4.5. Tato přesnost byla testována při měření na reálných zařízení. Pro tuto aplikaci je přesnost dostačující. V případě využití algoritmu v časově kritickém systému by bylo nutné dosáhnout mnohem větší přesnosti. Schopnost funkčnosti celého systému v reálném provozu (hře) testována nebyla 7.2.

Výsledné zařízení by kromě zamýšleného využití jako hlasovací zařízení pro společenské hry mohlo najít i využití ve vzdělávání nebo zdravotnictví.

Literatura

- [1] *Internet of Things Global Standards Initiative.*
<https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>.
- [2] Alexander S. Gillis. *What is the internet of things (IoT)?* 2022.
<https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>.
- [3] *Internet of things.* 2001-.
https://en.wikipedia.org/wiki/Internet_of_things.
- [4] *Machine to Machine Communications.* 2022.
<https://dot.gov.in/machine-machine-communications>.
- [5] *Průmysl 4.0.* 2022.
<https://www.plm.automation.siemens.com/global/cz/our-story/glossary/industry-4-0/29278>.
- [6] *Co je průmysl 4.0?*
<https://www.sap.com/cz/insights/what-is-industry-4-0.html>.
- [7] *Wireless.* 2001-.
<https://en.wikipedia.org/wiki/Wireless>.
- [8] *LoRa.* 2001-.
<https://en.wikipedia.org/wiki/LoRa>.
- [9] *LoRa Alliance®.* 2023.
<https://lora-alliance.org/>.
- [10] *Sigfox.* 2001-.
<https://cs.wikipedia.org/wiki/Sigfox>.
- [11] *Sigfox.* 2023.
<https://www.sigfox.com/>.
- [12] *Narrowband IoT.* 2001-.
https://en.wikipedia.org/wiki/Narrowband_IoT.
- [13] *Vodafone.* 2023.
<https://www.vodafone.cz/podnikatele/internet-veci/nb-iot1/>.
- [14] Kais Mekki, Eddy Bajic, Frédéric Chaxel a Fernand Meyer. A comparative study of LPWAN technologies for large-scale IoT deployment. 2019, 5 1-7. DOI 10.1016/j.icte.2017.12.005.
- [15] *IEEE 802.* 2001-.
https://en.wikipedia.org/wiki/IEEE_802.
- [16] *ZigBee.* 2001-.
<https://cs.wikipedia.org/wiki/ZigBee>.
- [17] *Bluetooth.* 2001-.
<https://cs.wikipedia.org/wiki/Bluetooth>.

- [18] *Wi-Fi*. 2001-.
<https://en.wikipedia.org/wiki/Wi-Fi>.
- [19] *802.11 versions*. 1981- 2023.
<https://www.pc当地.com/encyclopedia/term/80211-versions>.
- [20] *Využívání informačních a komunikačních technologií v domácnostech a mezi osobami - 2022*. 2022.
<https://www.czso.cz/documents/10180/164606768/0620042201.pdf/5699654d-a722-44c9-a5e8-80443c89be18?version=1.1>.
- [21] Win Hlaing, Somchai Thepphaeng, Varunyou Nontaboot, Natthanan Tangsunantham, Tanayoot Sangsuwan a Chaiyod Pira. Implementation of WiFi-based single phase smart meter for Internet of Things (IoT). *2017 International Electrical Engineering Congress (iEECON)*. 2017, 1-4. DOI 10.1109/IEECON.2017.8075793.
- [22] *Low-Power Wi-Fi Ideal for Energy Efficient IoT Devices*. 2023.
<https://www.silabs.com/blog/low-power-wi-fi-Ideal-for-energy-efficient-iot-devices>.
- [23] Benny Har-Even. *Wi-Fi 6 For Low-Power IoT*. 2013-2023.
<https://semiengineering.com/wi-fi-6-for-low-power-iot/>.
- [24] Tomáš Přeučil a Martin Novotný. *Evaluation of power saving methods for low-power WiFi environment sensors*. In: *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. 2022. 1-5.
- [25] *ESP-NOW*. c2016 - 2023.
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html.
- [26] IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*. 2021, 1-4379. DOI 10.1109/IEEESTD.2021.9363693.
- [27] *Service set (802.11 network)*. 2001-.
[https://en.wikipedia.org/wiki/Service_set_\(802.11_network\)](https://en.wikipedia.org/wiki/Service_set_(802.11_network)).
- [28] CSc. Doc. Ing. Cyril Klimeš. *Distribuované systémy*. In: Ostrava: Ostravská univerzita v Ostravě, Přírodovědecká fakulta, 2014. 7-32.
<https://web.archive.org/web/20140714151456/http://www1.osu.cz/~prochazka/ds/SkriptaKlimes.pdf>.
- [29] *Distribuovaný systém*. 2001-.
https://cs.wikipedia.org/wiki/Distribuovan%C3%BD_syst%C3%A9m.
- [30] Michal Jakob. *Úvod do distribuovaných systémů*. In: Praha: 16-37.
https://cw.fel.cvut.cz/wiki/_media/courses/b4b36pdv/lectures/pdv_ds_01_uvod_modely_2022.pdf.
- [31] Michael J. Fischer, Nancy A. Lynch a Michael S. Paterson. *Impossibility of Distributed Consensus with One Faulty Process*. In: *Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985.*.. Association for Computing Machinery, 1985. 1-9.
<https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf>.

- [32] Michal Jakob. *Závěr a shrnutí*. In: Praha: 1-11.
https://cw.fel.cvut.cz/wiki/_media/courses/b4b36pdv/lectures/06_zaver_a_shrnuti_2021.pdf.
- [33] *Berkeley algorithm*. 2001-.
https://en.wikipedia.org/wiki/Berkeley_algorithm.
- [34] Michal Jakob. *Čas a kauzalita v DS*. In: Praha:
https://cw.fel.cvut.cz/b202/_media/courses/b4b36pdv/lectures/02_cas_a_usporadani_2021.pdf.
- [35] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*. 1989, 3 (3), 146-158. DOI 10.1007/BF01784024.
- [36] František Zezulka a Ondřej Hynčica. Synchronizace v distribuovaných řídicích systémech: Precision Time Protocol (PTP) podle IEEE 1588. *AUTOMA*. 2010, 2010 (2), 17-19.
- [37] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*. 2020, 1-499. DOI 10.1109/IEEESTD.2020.9120376.
- [38] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*. 1978, 21 (7), 558-565. DOI 10.1145/359545.359563.
- [39] Dana Yang, Inshil Doh a Kijoon Chae. Cell Based Raft Algorithm for Optimized Consensus Process on Blockchain in Smart Data Market. *IEEE Access*. 2022, 10 85199-85212. DOI 10.1109/ACCESS.2022.3197758.
- [40] Michal Jokob. *Algoritmus Raft*. In: Praha: 1-40.
https://cw.fel.cvut.cz/wiki/_media/courses/b4b36pdv/lectures/05_raft_2021.pdf.
- [41] Diego Ongaro a John Ousterhout. In Search of an Understandable Consensus Algorithm.
- [42] Petr Kučera. *Raft algoritmus*. In: *Kučův blog*. Spojil:
<https://blog.petrkucera.cz/post/raft-algoritmus>.
- [43] *Reakční doba*. 2001-.
https://cs.wikipedia.org/wiki/Reak%C4%8Dn%C3%AD_doba.
- [44] *ESP32-S2-Pico*.
<https://www.waveshare.com/wiki/ESP32-S2-Pico>.

Příloha A

Slovniček pojmu

AI	■ Artificial Intelligence, umělá inteligence
AP	■ access point
CSS	■ Chirp Spread Spectrum
DS	■ distribuovaný systém
frame	■ rámec
IEEE	■ Institute of Electrical and Electronics Engineers
IoT	■ Internet of Things, Internet věcí
ISM	■ industrial, scientific and medical
LPPAN	■ Low-power privat-area network
LPWAN	■ Low-power wide-area network
MAC adress	■ media access control address
MCU	■ mikrokontroler
SoC	■ System on Chip
UNB	■ Ultra Narrow-Band

Příloha B

Detailní výsledky měření sítové infrastruktury

B.1 Scénář A

Scénáře Axy



Obrázek B.1. Vizualizace parametrů pro měření scénářů A.

Ve scénářích typu A se snažím otestovat to, jak změna parametru typu odesílání, tj. přepínání mezi *broadcastem* a *unicastem*, ovlivní *round-time trip* v závislosti na velikosti zprávy. Měření se odehrává v prostředí bytu v činžovním domě, kde dochází k rušení několika okolními Wifi sítěmi.

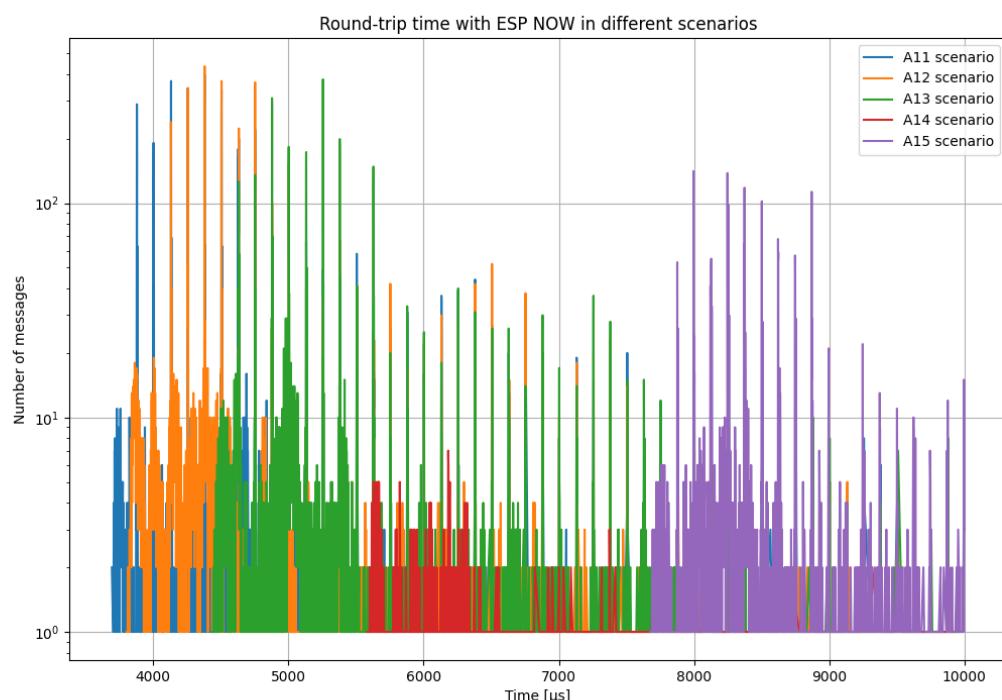
SCÉNÁŘE	A11	A12	A13	A14	A15
prostředí	byt	byt	byt	byt	byt
překážka	vzduch	vzduch	vzduch	vzduch	vzduch
vzdálenost	50 cm				
velikost	1 B	10 B	50 B	120 B	250 B
počet zpráv	10 000	10 000	10 000	1 000	5 000
typ vysílání	broadcast	broadcast	broadcast	broadcast	broadcast

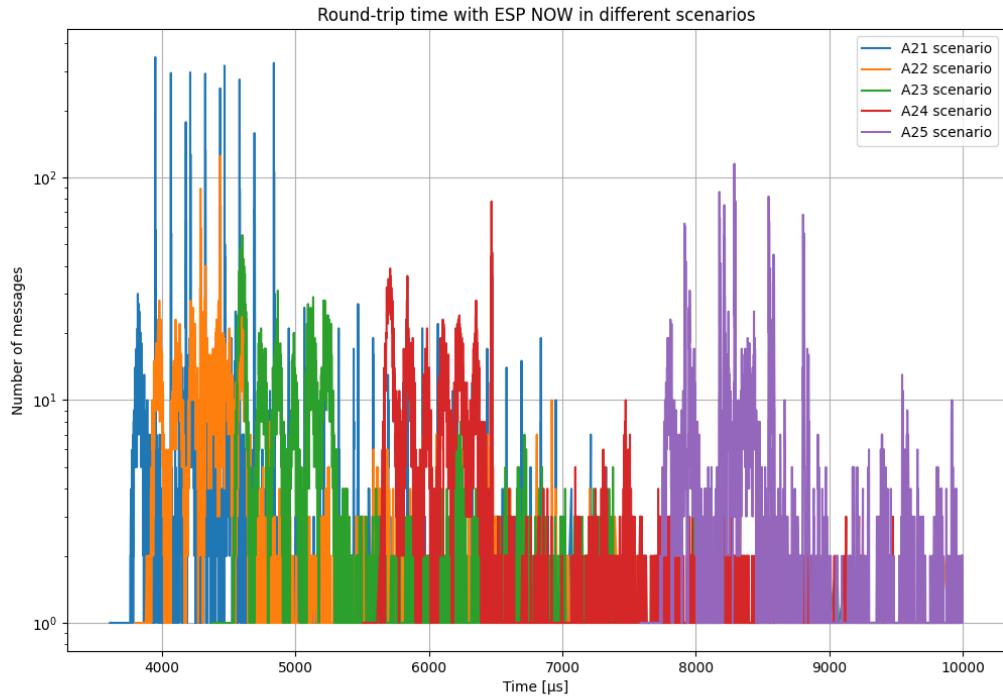
Tabulka B.1. Přehled parametrů pro A1x scénáře.

Výsledky měření těchto scénářů si je možné prohlédnout v grafech B.2 a B.3. Odesílání se chová dle očekávání. Zprávy větší velikosti trvají déle než zprávy té menší. Zajímavé je srovnání *broadcastu* a *unicastu*. *Broadcast* je nepatrňě pomalejší.

Také je zajímavé si povšimnout nakumulovaných odpovědí v jeden čas. Tuto skutečnost si vysvětluji implementací *broadcastu* v protokolu ESP-NOW.

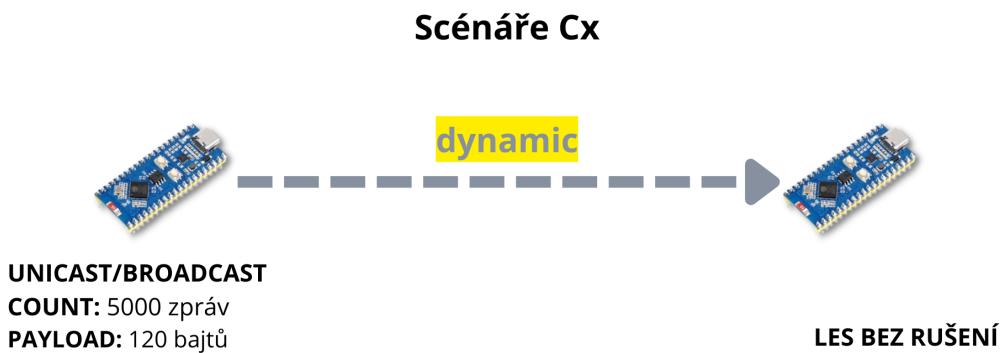
SCÉNÁŘE	A21	A22	A23	A24	A25
prostředí	byt	byt	byt	byt	byt
překážka	vzduch	vzduch	vzduch	vzduch	vzduch
vzdálenost	50 cm				
velikost	1 B	10 B	50 B	120 B	250 B
počet zpráv	10 000	10 000	10 000	10 000	10 000
typ vysílání	unicast	unicast	unicast	unicast	unicast

Tabulka B.2. Přehled parametrů pro A2x scénáře.**Obrázek B.2.** Graf výsledků měření scénářů A1.



Obrázek B.3. Graf výsledků měření scénářů A2.

B.2 Scénář C



Obrázek B.4. Vizualizace parametrů pro měření scénářů C.

Sadou scénářů Cx se snažím pozorovat vlastnosti v signálově čistém prostředí¹ v závislosti na vzdálenosti a typu vysílání.

Výsledky měření těchto scénářů si je možné prohlédnout v grafu B.5. Zde se opět protokol chová dle očekávání. Nedochází k takovému zpoždění jako například při měření scénářů typu A. Také si je možné povšimnout toho, že se jednou za čas nějaká zpráva opozdí.

¹ Nejedná se o laboratorně čisté prostředí. Měření bylo prováděno v prostředí lesa, který je od nejbližší obce vzdálen asi 5 km a v okolí mého bydliště nejvíce čisté od 2,4 GHz rušení.

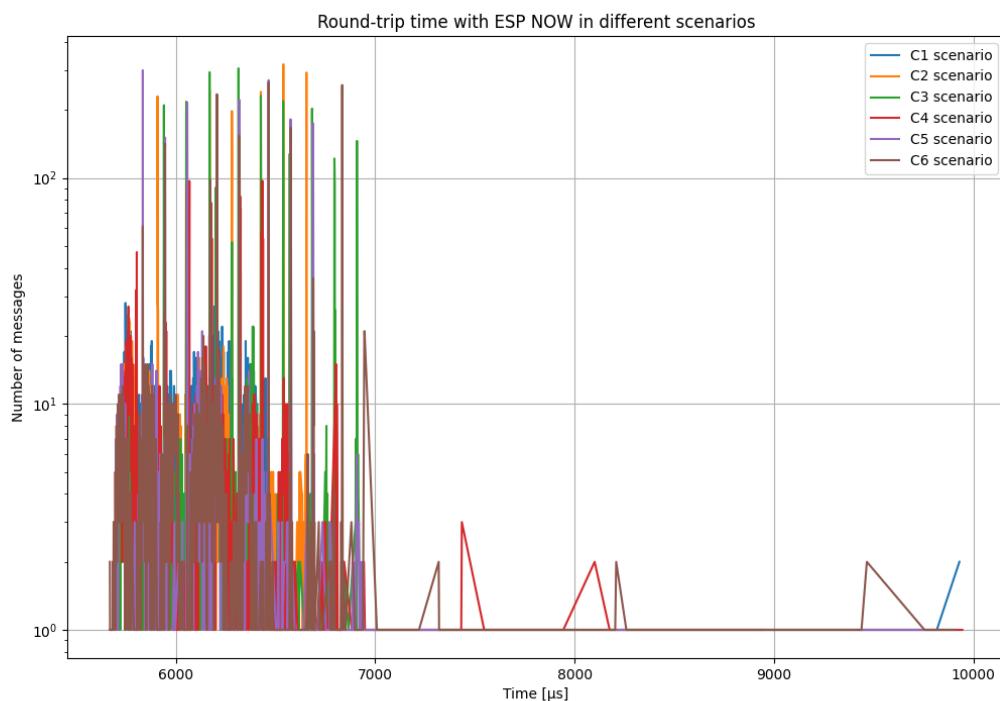
SCÉNÁŘE	C1	C2	C3	C4	C5	C6
prostředí	les	les	les	les	les	les
překážka	vzduch	vzduch	vzduch	vzduch	vzduch	vzduch
vzdálenost	0,5 m	25 m	50 m	100 m	100 m	50 m
velikost	125 B	125 B				
počet zpráv	5 000	5 000	5 000	5 000	5 000	5 000
počet chybných zpráv	0	3	15	35	15	6
typ vysílání	unicast	unicast	unicast	unicast	broadcast	broadcast

Tabulka B.3. Přehled parametrů pro Cx scénáře.

Při tomto měření jsem zaznamenával také chybovost počet chybných zpráv.² Jejich četnost si ji možné prohlédnout v tabulce B.3.

Během tohoto měření jsem zjistil, že je důležité, aby na větší vzdálenosti³ nestála signálu v cestě žádná překážka.

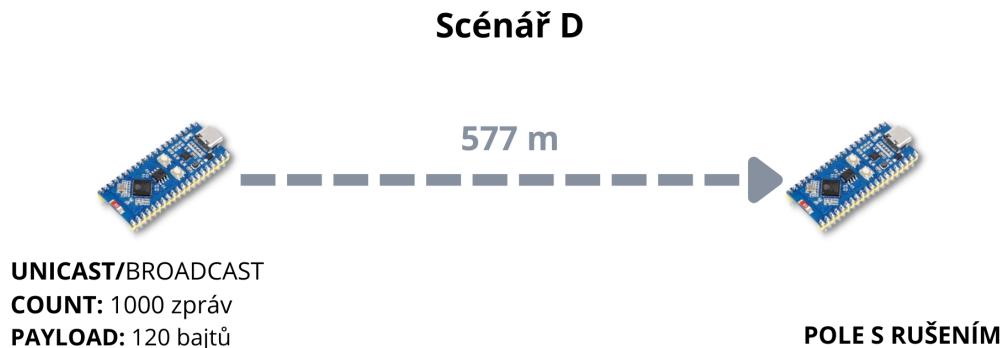
Zajímavé je také srovnání scénáře A a C. Můžeme pozorovat, že **vliv vzdálenosti ovlivňuje především ztrátovost paketů. Naopak velikost ovlivňuje rychlosť přenosu.**

**Obrázek B.5.** Graf výsledků měření scénářů C.

² Chybnou zprávou se myslí taková zpráva, která nedorazí do specifikovaného deadlinu, tedy mimo graf.

³ 25 m a více

B.3 Scénář D



Obrázek B.6. Vizualizace parametrů pro měření scénářů D.

Scénář D byl oproti ostatním měřením odlišný v tom, že jsem se nejprve snažil stanovit hranici, kdy je zařízení ještě schopno přijímat zprávy a kdy už ne. Experimentálně jsem dospěl k hranici 580 m. Následně jsem odesal 1000 zpráv s cílem zjistit, jak je veliká ztrátovost. Měření bylo realizováno na poli, přes které může procházet signál na 2,4 GHz.

SCÉNÁŘE	D1
prostředí	pole (s 2,4 GHz)
překážka	vzduch
vzdálenost	577 m
velikost	125 B
počet zpráv	1 000
počet chybných zpráv	50
typ vysílání	unicast

Tabulka B.4. Přehled parametrů pro D scénář.

Při měření jsem zjistil, že při odesílání na velikou vzdálenost je třeba dbát na orientaci čipu. Pokud nebylo zařízení správně natočeno, nešlo odeslat žádné zprávy.

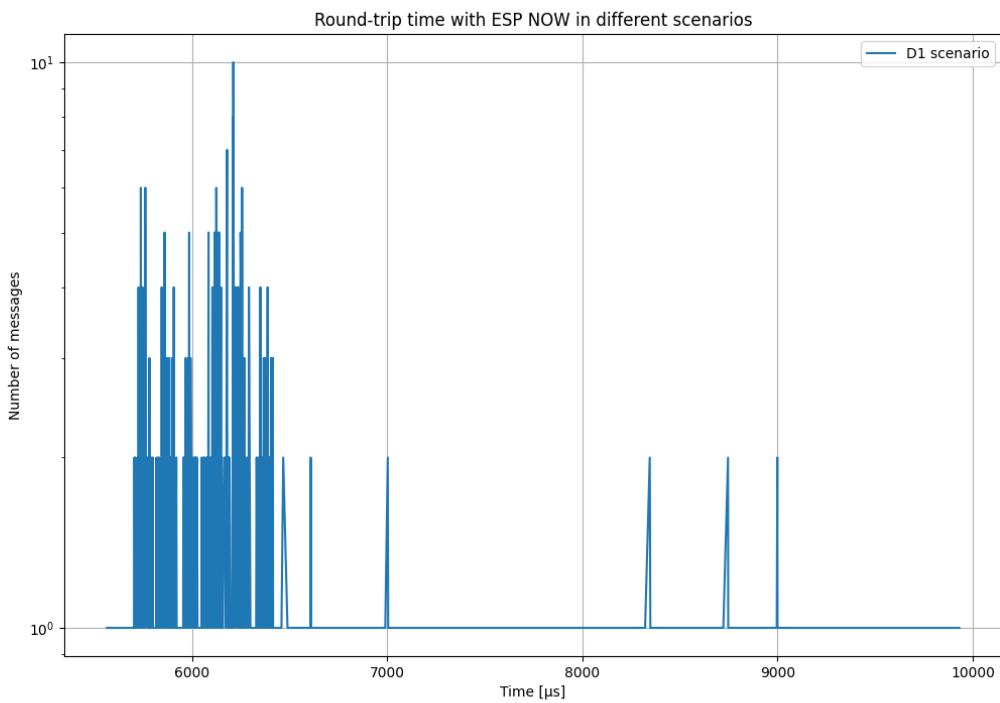
Výsledek měření je vizualizován grafem B.7. Při tomto scénáři bylo ovšem mnohem zajímavější pozorovat četnost úplné ztráty dat.

Při odeslání 1000 zpráv, se ztratilo 50. Můžeme tedy jednoduchým výpočtem zjistit, jaká je procentuální ztrátovost na dlouhé vzdálenosti.

$$\text{loss} = \frac{\text{error}}{\text{sent}}$$

V našem případě se při odeslání 1000 zpráv objevilo 50 chyb. Chybovost je tedy 5 %.⁴.

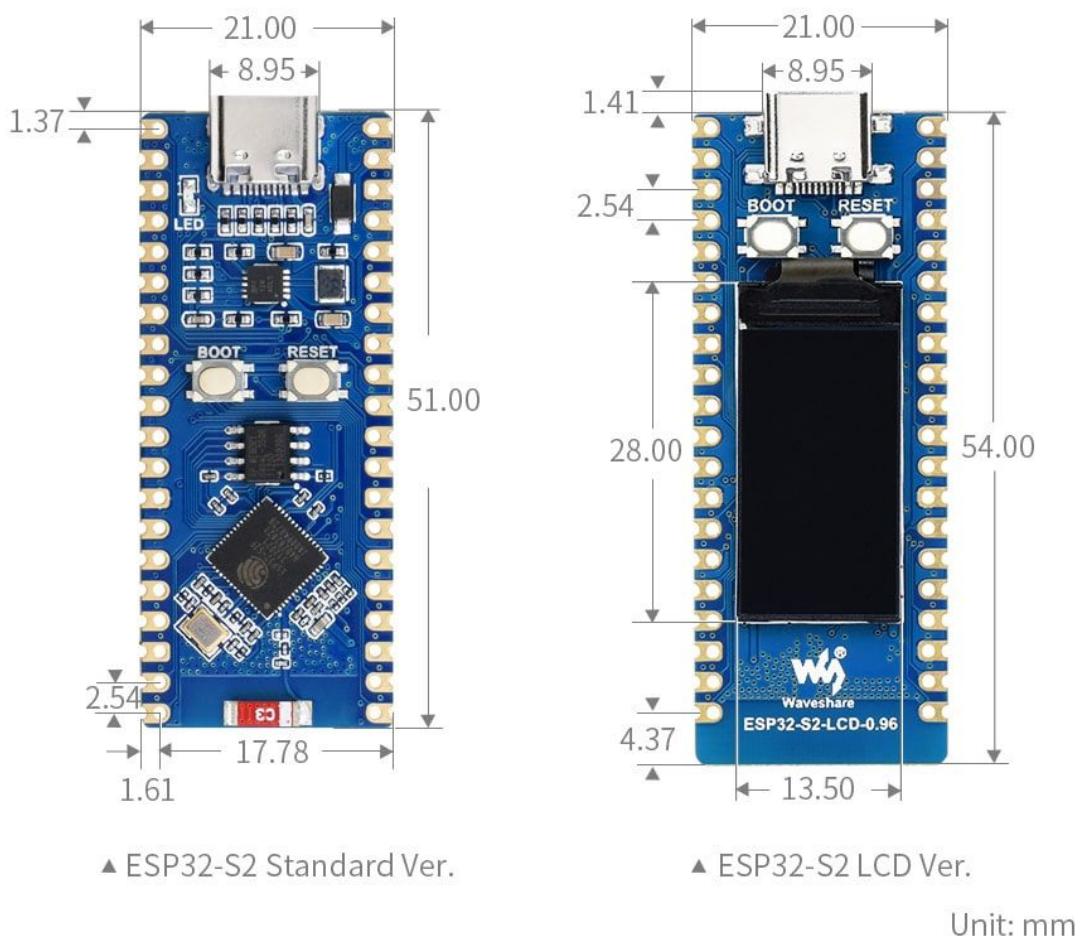
⁴ $\frac{50}{1000} = 0,05$



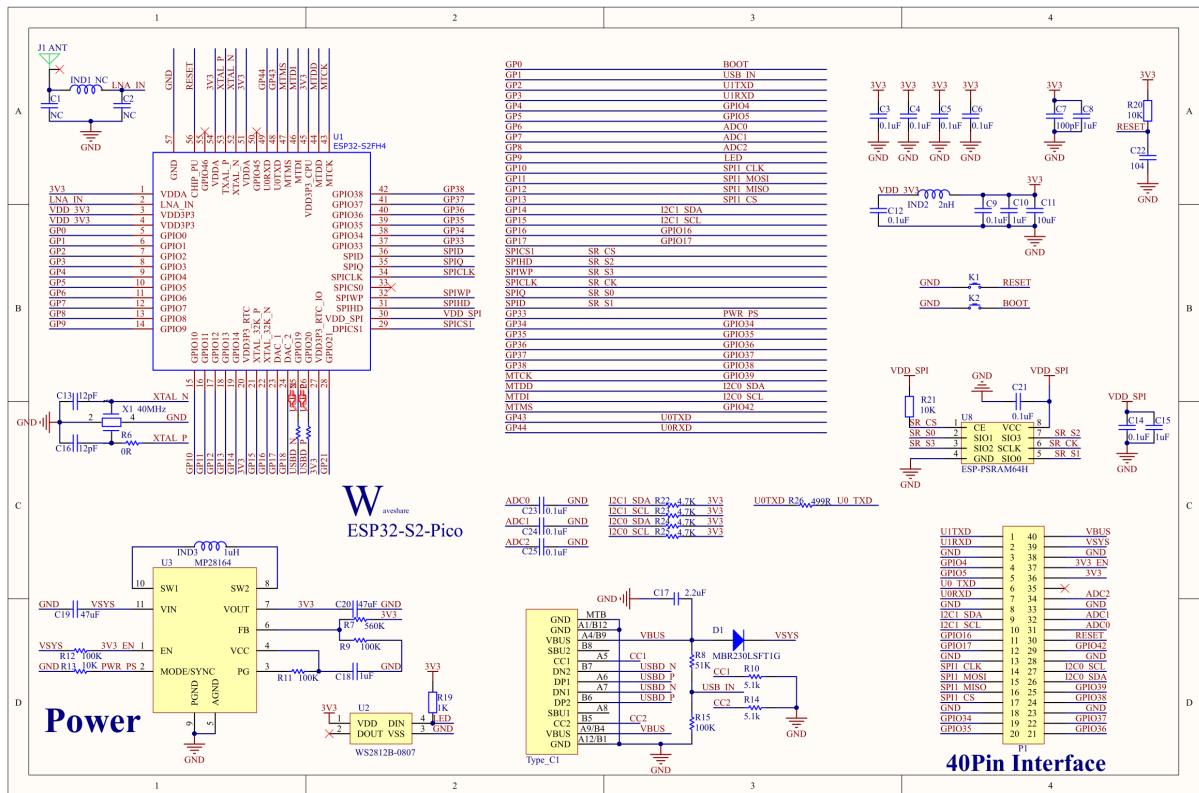
Obrázek B.7. Graf výsledků měření scénáře D.

Příloha C

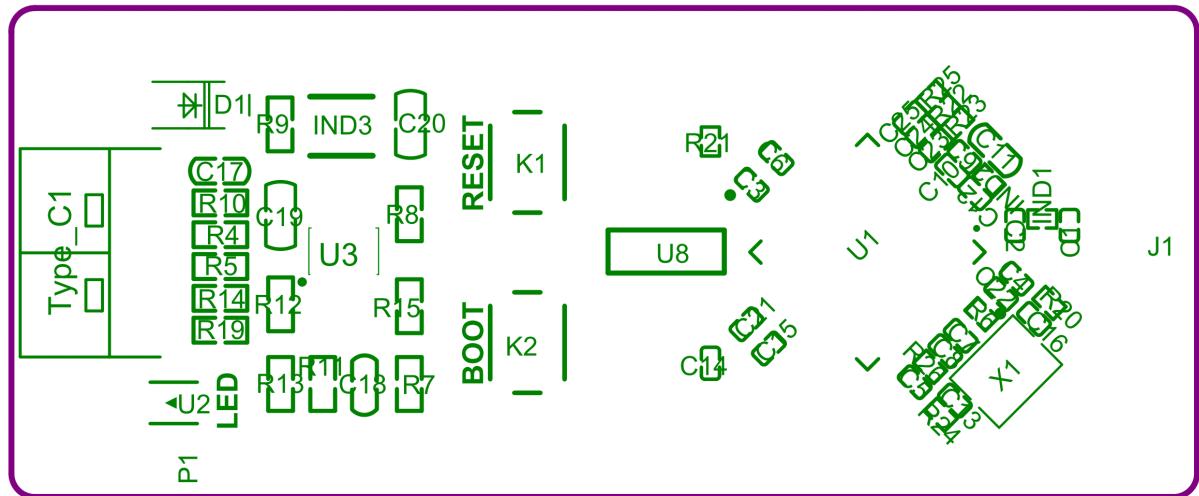
Schéma modulů



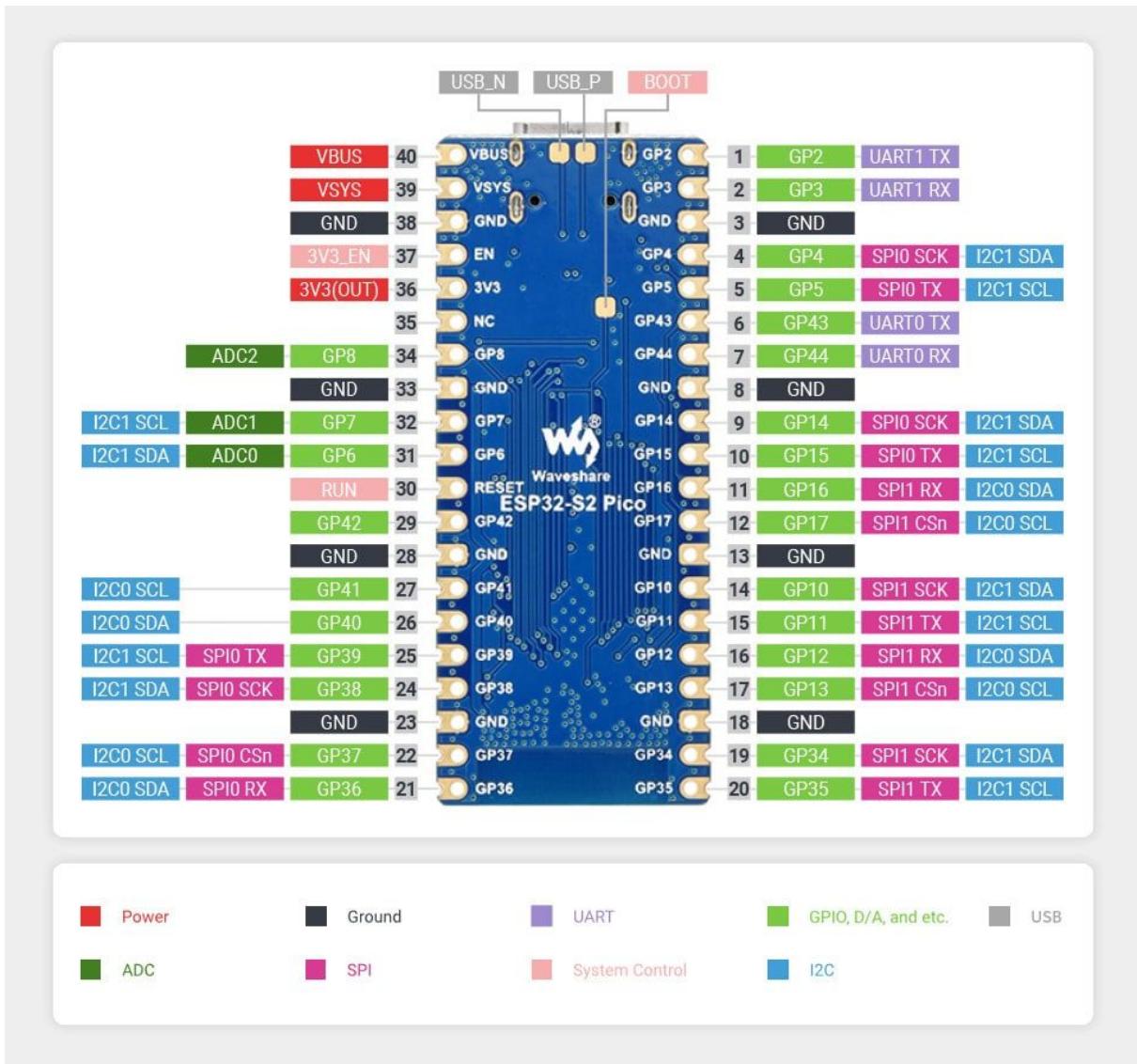
Obrázek C.8. Rozměry ESP32-S2-Pico [44].



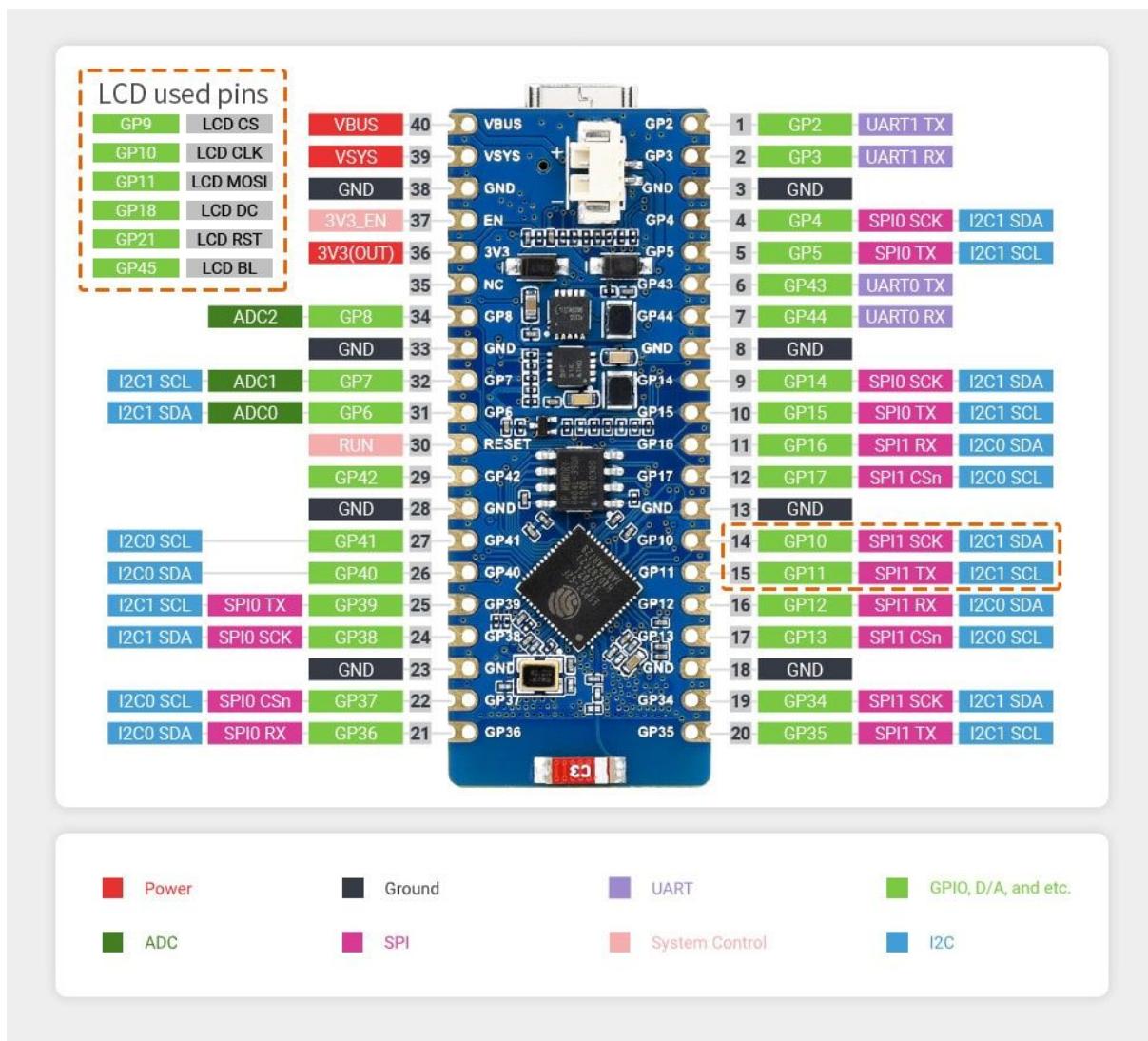
Obrázek C.9. Schéma ESP32-S2-Pico [44].



Obrázek C.10. Schéma rozložení ESP32-S2-Pico [44].



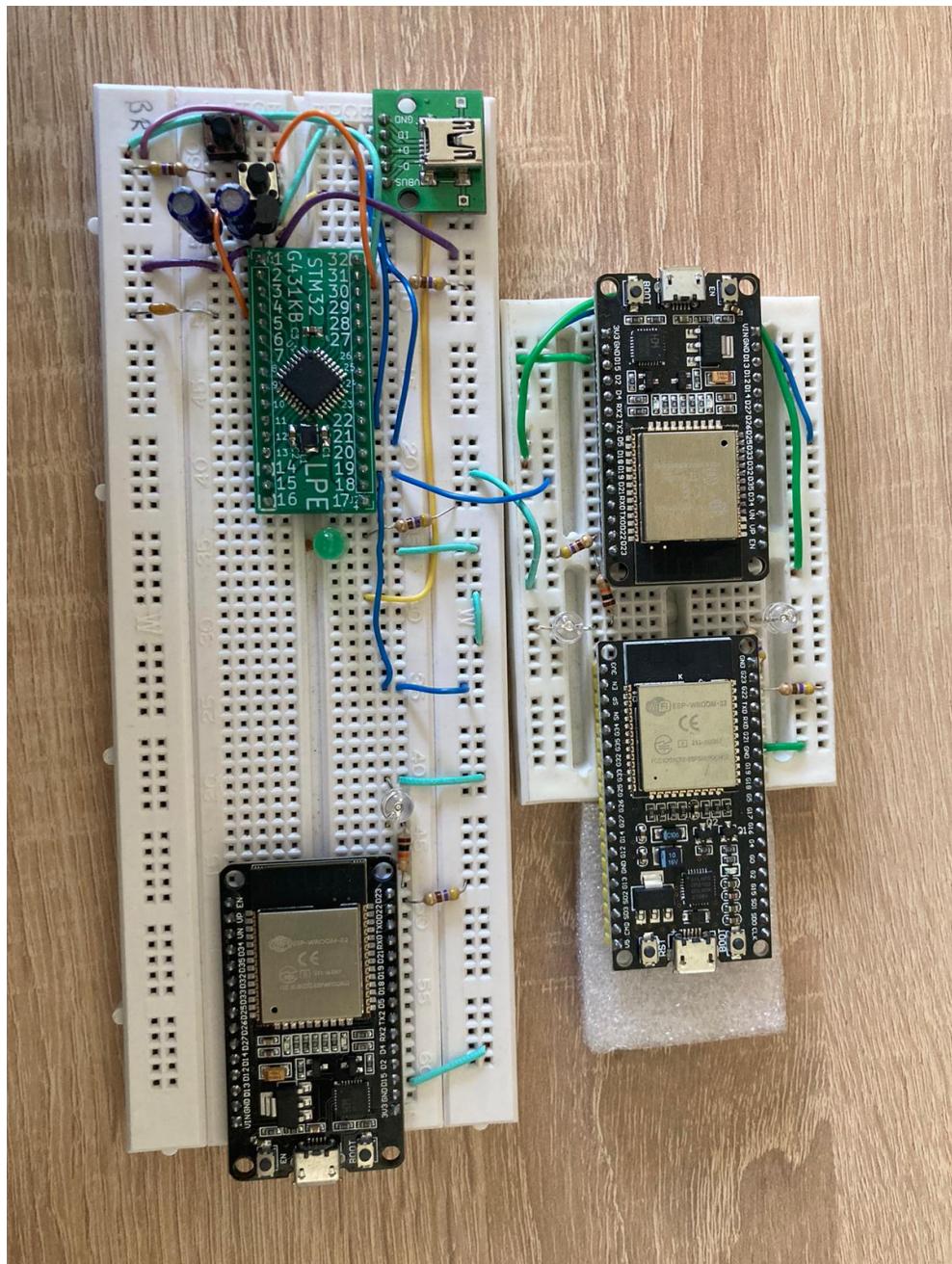
Obrázek C.11. Schéma pinu ESP32-S2-Pico [44].



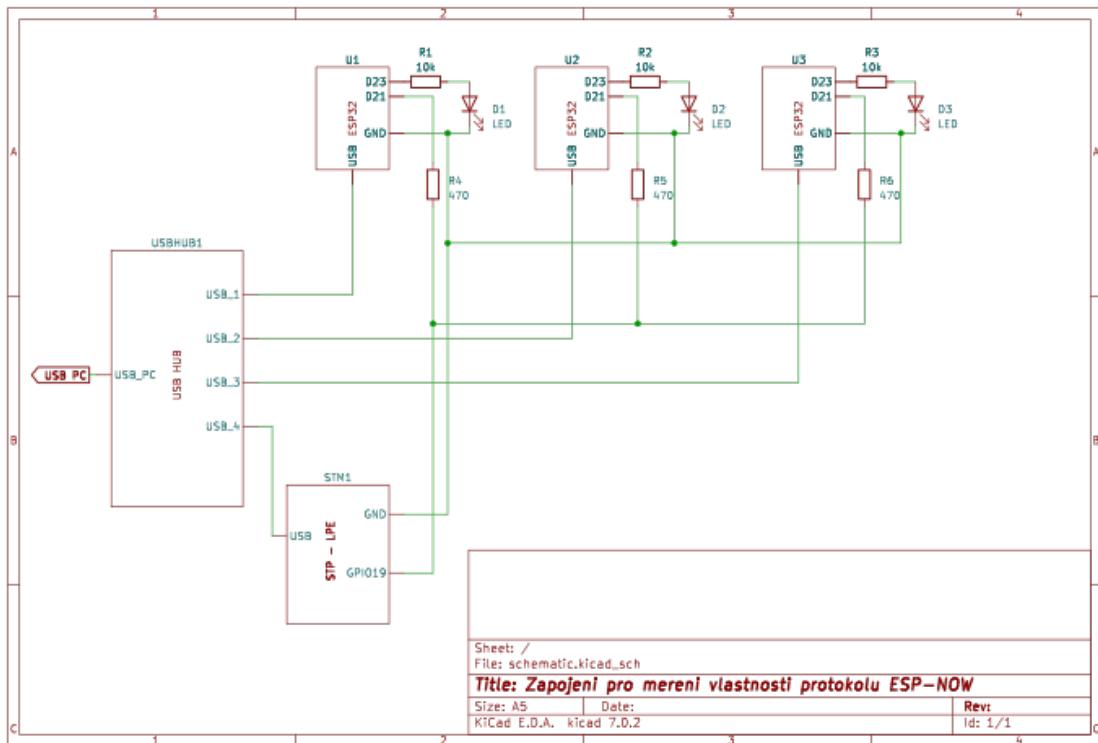
Obrázek C.12. Schéma pinu ESP32-S2-LCD [44].

Příloha D

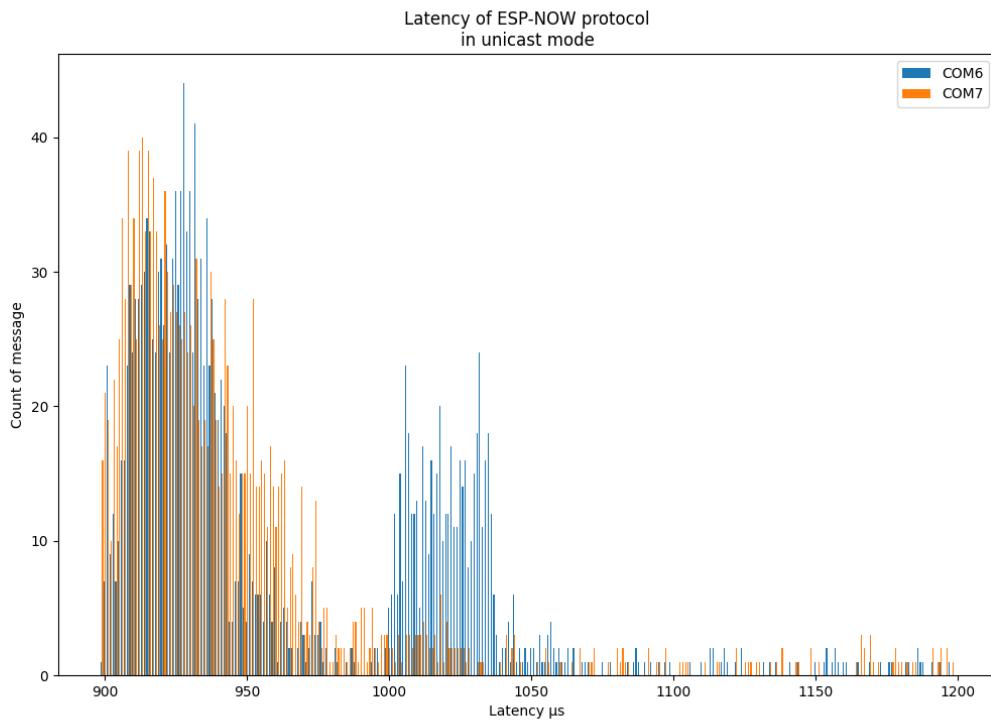
Měření latence



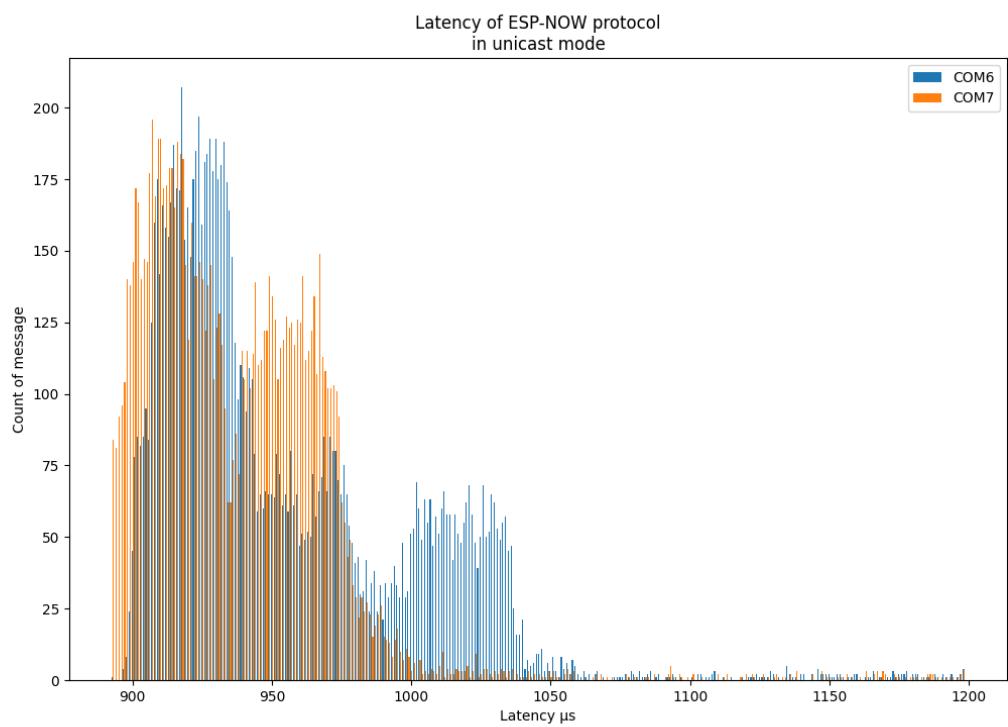
Obrázek D.13. Zapojení obvodu pro měření latence.



Obrázek D.14. Schéma zapojení obvodu pro měření latence.



Obrázek D.15. Výsledky měření latence při scénáři A.

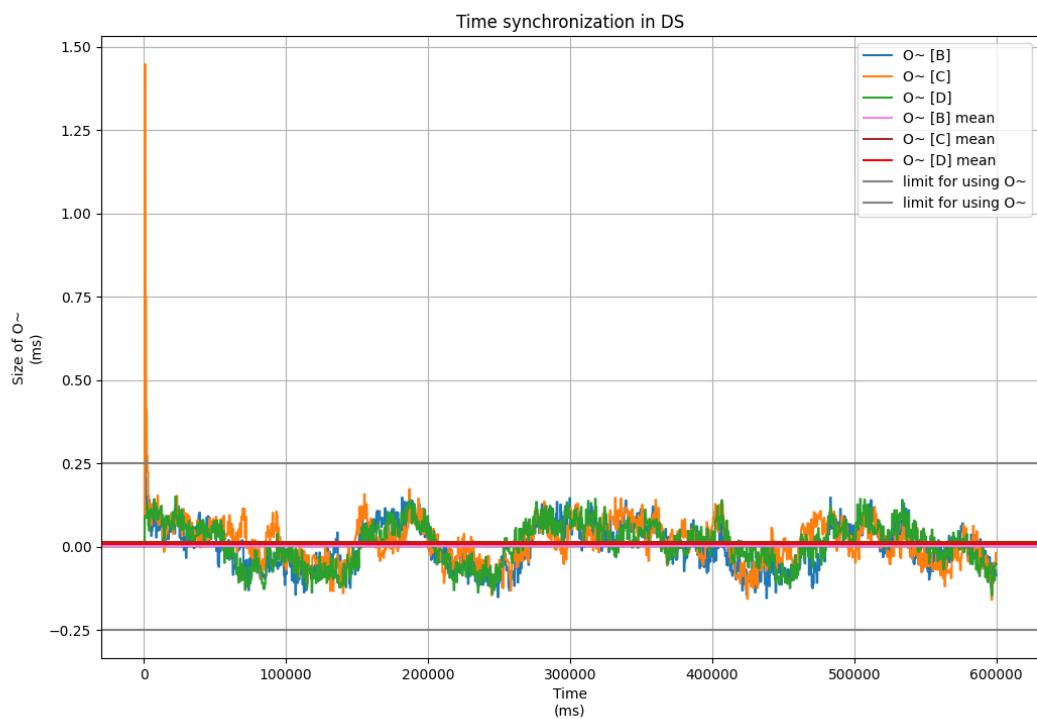


Obrázek D.16. Výsledky měření latence při scénáři B.

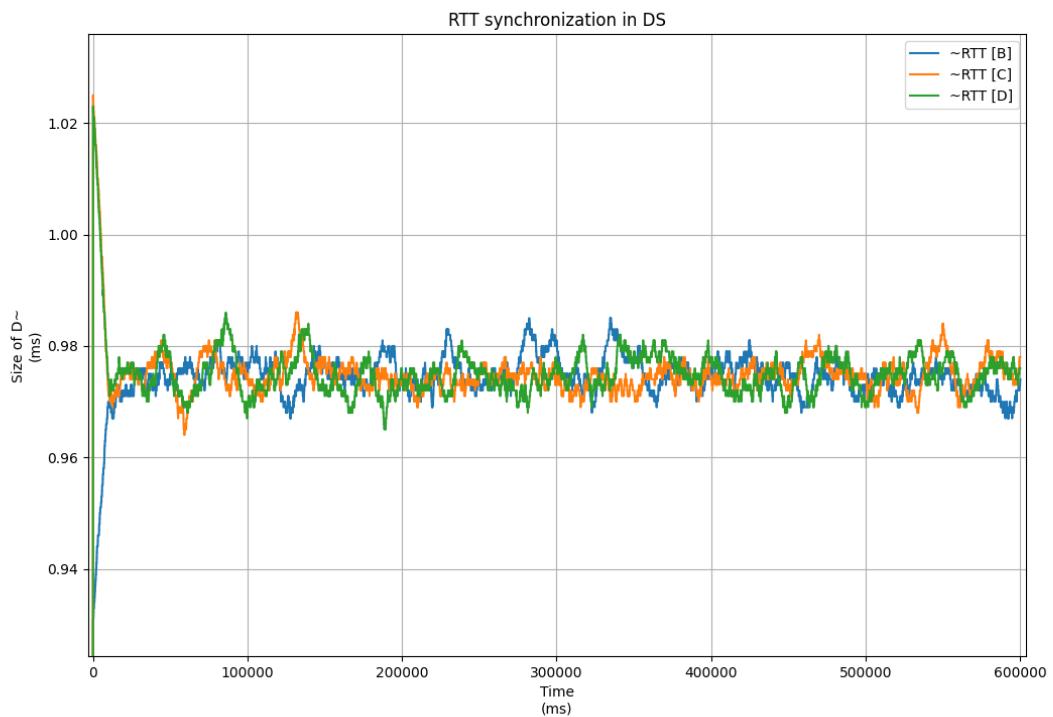
Příloha E

Synchronizace času

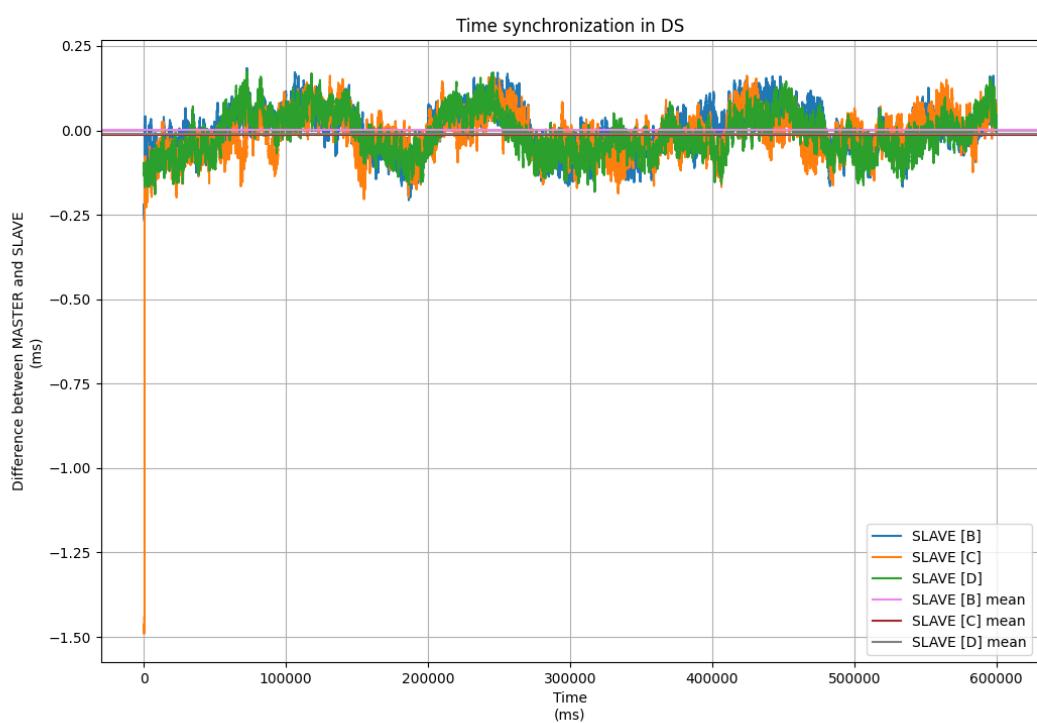
E.1 Simulace



Obrázek E.17. Simulace fungování algoritmu pro synchronizaci času - chyba.

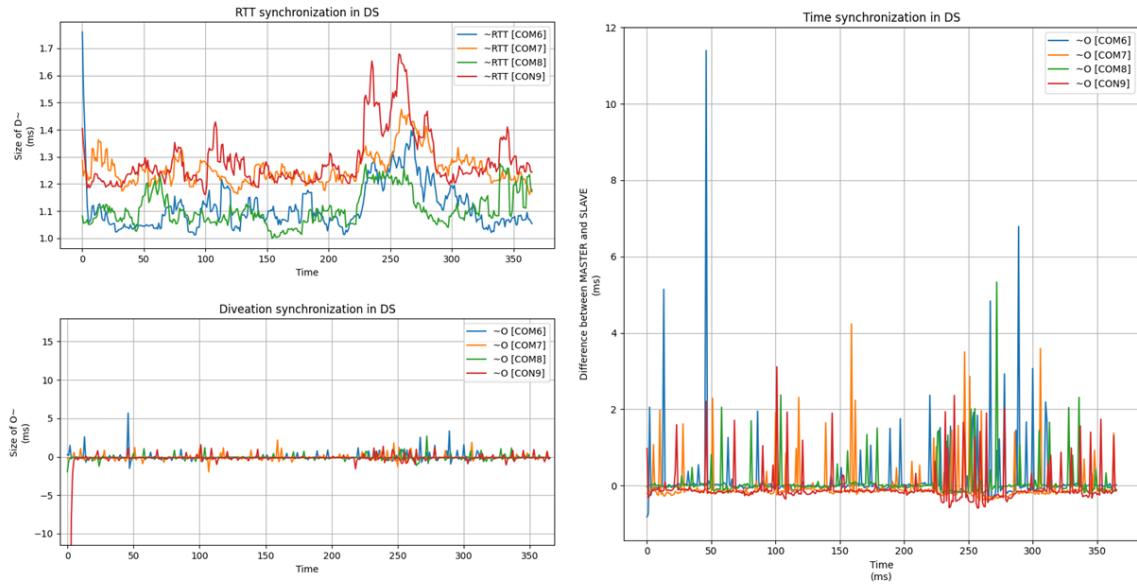


Obrázek E.18. Simulace fungování algoritmu pro synchronizaci času - doba přenosu.

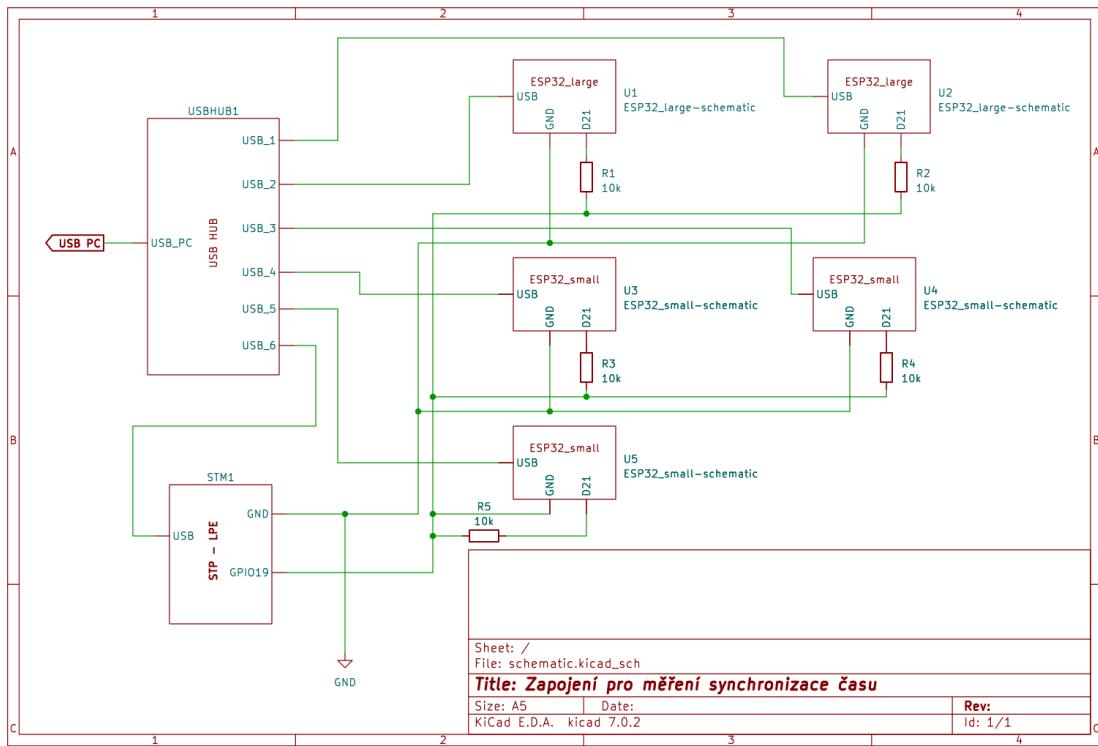


Obrázek E.19. Simulace fungování algoritmu pro synchronizaci času - rozdíl času.

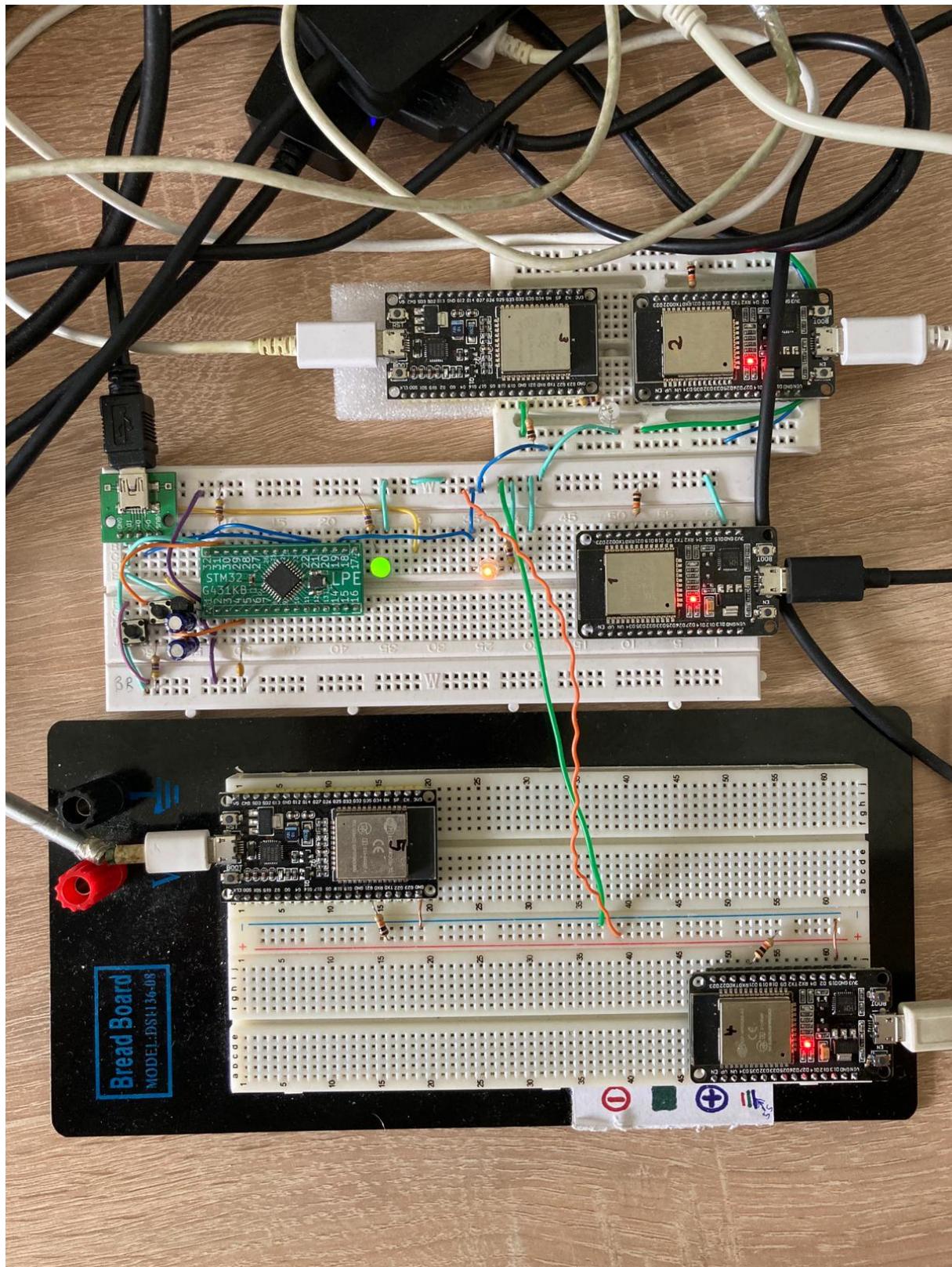
E.2 Měření



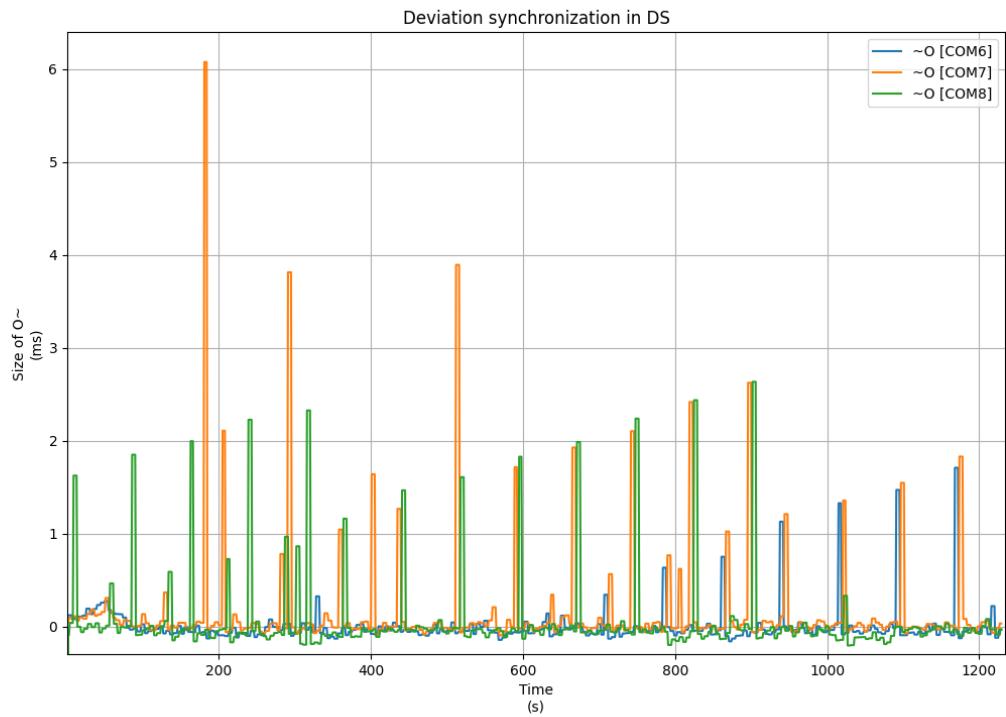
Obrázek E.20. Průběžné výsledky měření synchronizace času na reálném zařízení.



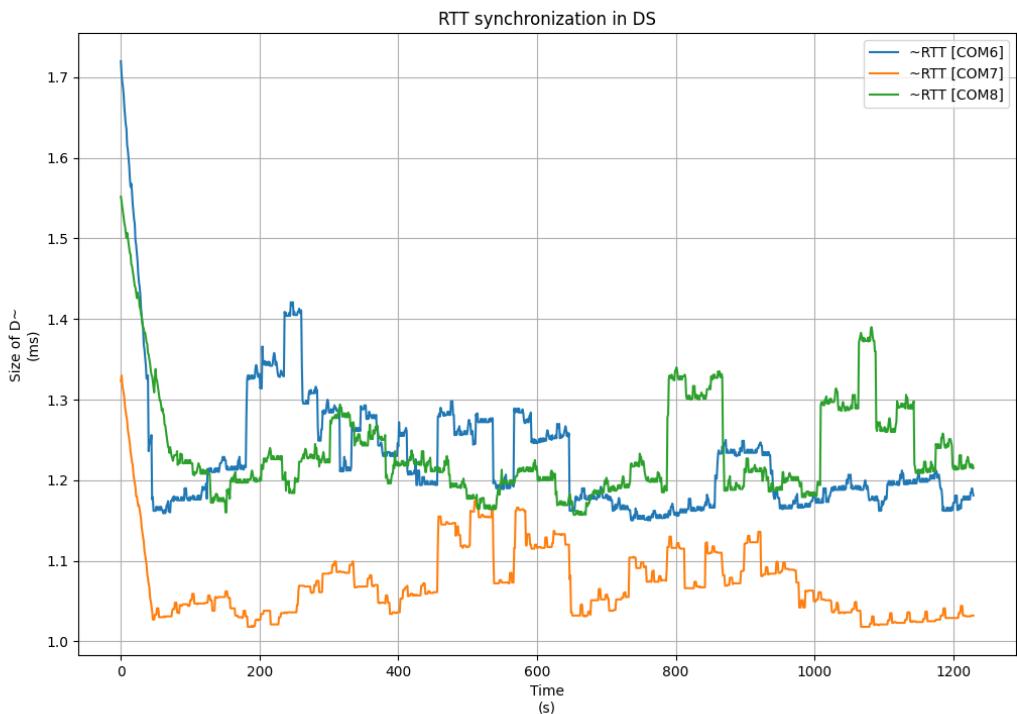
Obrázek E.21. Měření synchronizace času - schéma.



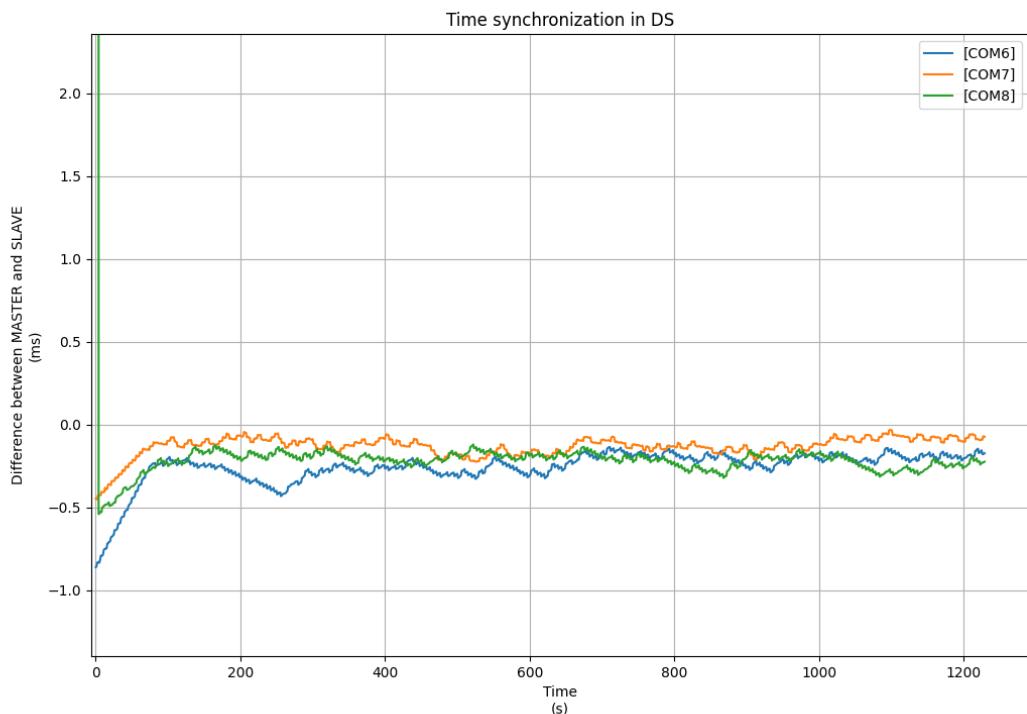
Obrázek E.22. Měření synchronizace času - zapojení.



Obrázek E.23. Měření synchronizace času - chyba.



Obrázek E.24. Měření synchronizace času - doba přenosu.



Obrázek E.25. Měření synchronizace času - rozdíl času.

Příloha F

Slovní komentáře průzkumu podoby zařízení

Níže přikládám okopírované odpovědi, které nejsou modifikovány mimo vymazání jmen respondentů kvůli ochraně osobních údajů.

flákací verze dává možnost vyniknout introvertnějším ale chytřejším členům týmu, proto vybírám ji

Hele, v týmu si rozhodně radší praštím.
Kdyby to mělo být třeba na jednotlivce,
nebo nějaký spešl důvod, tak si dovedu představit
ten malej klikač, ale ve většině aplikací bych
upřednostnil velký bouchadlo.
Čusákos.
Lobumbus.

Variantu A bych zvolil pokud by každý hlasoval za sebe,
varianta B mi připadá spíše jako jedno tlačítko
pro celý tým, který stojí kolem.

V návalu adrenalinu, stejně tak po požití alkoholu stoupá soupeřivost a určitým způsobem třeba i živelnost či mírná agrese. Proto si myslím, že odolnější zařízení je pro tento model lepší variantou.

Za prvé, nesnáším tyhle hry, ale když si představím, že by mě to bavilo, tak by bylo víc vzrušující si pořádně fláknout i za cenu toho, že to může být pomalejší způsob

Myslím, že při hrách vzniká taková atmosféra,
že odolná buch krabička bude pamparádní.

Brala bych verzi B - menší šance ztráty a kdo by chtěl vyměňovat baterky u 30 mini kontrolérů?
Taky na mě možnost B působí víc týmově, nejsi tam sám za sebe, máš prostě jeden čudlík o který se dělíš se svým týmem.
Obojí se mi zdá v něčem špatné. Malá verze bude jednoduše rozbitná, ale lepší na ovládání. U druhé zase hrozí nějaké sražení rukou a možné hádky. Myslím, že by to chtělo kompromis mezi oběma.
Myslím asi malé zařízení, které bude zároveň odolnější z tvrdsiho plastu a jednoduššího tvaru.

Odolnější mi připadá jako v týmu větší zábava, raději bych teda byl pro tuto variantu s tím, když budou v týmu hezky slečny tak se můžeme mačkat.

Pokud to má být zařízení je do hospody a ne na tajné hlasování, tak určitě odolná krabice na stole uprostřed. Už ten střet rukou v rámci týmu pak může být zajímavý

Kromě toho, že bych se musela na flaknuti zvednout, tak by jste docházelo k trapasum typu "omlouvám se, vy první" "ne, vy první, máte přednost" apod.

Hodně hustý! Za mě tlačítko uprostřed stolu ideál, můžeš do toho mlátit jak chceš a nestane se že se třeba omylem uklikneš na nějaký malý blbosti v ruce, že ti to upadne nebo Bůh ví co

Pokud je to pro tým, přijde mi efektivnější zařízení, ke kterému mají všichni stejnou možnost přístupu + pocitově to vypadá, že to bude fungovat rychleji než klikání.

"Ahoj, raději budu mít svoje tlačítko, než kdybych se měl dělit popřípadě se nějakým způsobem ""prát"" s ostatními, kdo to zvládne první. Snad to chápou dobře. :D

Pro mě to prostě byla první možnost, která mně osobně přišla pohodlnější... jinak ke svému výběru žádný větší důvod nemám.

Druhá varianta je asi zajímavější a i na zuřivost po odpovědi efektivnější, u té první není moc vysvětleno, jestli by to měl každý hráč v týmu nebo by bylo pouze jedno klikátko :) z dlouhodobějšího hlediska výdrže jsem pro kliknutí :D

Maly do ruky je praktictejsi

Rozhodně odolnější verze..pak se někdo zapomene, flákne a ejhle, je po klikací verzi

Podle mě záleží na cilovce, pro menší děti určitě kravička a pro dospělé asi ten clicker

Kliknutí se zdá o dost víc praktické, avšak dle mého názoru fláknutí k těmto typům her prostě patří a je to větší zábava. Navíc u klikru by mi vadilo, že nemám volnou ruku a ještě bych si ho někam položila a ztratila :D Flákací, to může být uprostřed stolu a lépe se s tím pracuje v týmu, než neustále ztrácat čas zjišťováním, u koho zůstal ovladač

Myslím, že nejdokonalejší verze by byla, kdyby měl každý své odolné tlačítko. Ze začátku hry by jistě stačilo jen tak lehce klikat, ale postupem času, kdy se do soutěže pořádně vžijete, je potřeba si řádně fláknout, co nejrychleji a nejlépe tak, aby ostatní viděli, že jste byl první. Proto by se hodilo, aby měl každý své odolné

tlačítko a nemusel se "strachovat", že se nestihne včas natáhnout do prostřed stolu nebo že mu bude někdo kolem překážet. Ale kdyby taková možnost nebyla, tak aspoň to jedno odolné tlačítko uprostřed, aby se člověk nebál, že to malé tlačítko v zápalu hry rozbije :) V zapálu boje bych si vybral druhou možnost, protože ta déle vydrží.

Rozhodně fláknu! To musí být nejzábavnější část toho procesu, jinak nehraju. :D Zdravím z Brna a krásný den:)

Proste jako Cink, to je vždycky adrenalin a hlavně by jsi do toho mohl vstupovat s nějakejma pravidlama - např. všichni musí mít ruce za zády, dokud se neřekne teď.. atd. což by u toho clickeru nešlo. Takže ja jsem proc řečo jako cink, idealně i se sound effectem Ať se daří!

Emoce přidávají na síle. Čím robustnější, tím lepší.

Pokud by měl každý své zařízení pak jsem pro kliknutí
Pokud by bylo jedno, tak fláknutí.

- 1) Volím flákanec, protože když jsem ve hře, tak mám extra adrenalin a určitě je uspokojující flagnout do tlačítka celou dlaní než mačkat tlačítko.
- 2) Přijde mi to lepší, protože je jedno, zda sedíš nebo stojíš, prostě je tato varianta přístupná krásně všem a může hádat více lidí najednou.
- 3) Ve většině her musíš mít ruce za zádama a tak je tato varianta lepší.
- 4) Je to rozhodně lepší varianta i s přihlédnutím k věku lidí. Malé děti mají menší ruku a proto můžou mít problém se stisknutím tlačítka, které bude ovládat pouze palec.
- 5) S tímto tlačítkem se bude pracovat nejen malým dětem, ale také seniorům (kteří mají ruce, které se klepají a ochablý palec - mohli by tlačítko zmáčknout nechtěně) a postiženým (kteří nemají takovou citlivost v prstech a spíš ovládají hrubou motoriku)

Záleží na co to bude použité - pokud pro hospodský kvíz, tak máš více lidí v týmu a kdo z nich bude první vědět tak do toho flákne. A je třeba, aby to bylo ideálně velké a dobře viditelné, protože nějakýho malého švába by taky mohli ty lidi v hospodě I pár sekund hledat.
Když to bude něco individuálního, třeba i běhacího, tak tam se hodí, že je to malé, můžu to mít permanentně v ruce a kliknout kdy budu potřebovat.
Takže obě dvě verze jsou možné a každá řeší něco jiného a to si musíš říct ty, co preferuješ, jaký přístup. V bakalářce bych ale zmínil oba a uvedl, že u tohodle mám ale prototyp. A kdybys potřeboval 3D tiskárnu na prototypování, tak neváhej

napsat, moje je ti kdykoliv k dispozici!
Pokud chceš dost dobrý informace, tak si napiš Zdeňku Mikovcovi
email: xmikovec@fel.cvut.cz je to garant celého HCI oboru
a je totálně povídavej, takže myslím, že kdyby sis mu napsal,
tak se s ním můžeš v pohodě sejít a dá ti taky know-how.

Pokud půjde o rychlost, je lepší mít něco v ruce. Vzhledem
k mé práci s dětmi vím, že u "flákacího" zařízení si hodně
ublíží (i fyzicky i slovně), tak bych spíše volila
pro každého jedno.

odolnost je v hospodě důležitá

Při hraní her bych rozhodně raději flákla, protože při
hospodském kvízu, ale i jiných stolních hrách, které se
hrají povětšinou s přáteli je žádoucí dát průchod svým emocím,
což mi prezentér absolutně neumožní. Zároveň by musel mít
v ruce každý svůj (více práce i nákladů), protože do prezentétu
nelze mlátit (skákal by po stole a tlačítko by se možná ani
nezmáčklo). Prezentér je tedy nutné vzít do ruky a pak až
mačkat (za éče více času), což je dle mého názoru mnohem
méně atraktivní možnost (nelze se při ní vyblbnout).
Doporučila bych ji třeba na nějaké konference či jednání,
kde se chce člověk přihlásit o slovo.

Zároveň mám poznámku k "flákacímu" zařízení. Udělala bych určitě
větší tlačítko a vypouklé, aby hezky sedělo do dlaně. Ale chápu,
že může být obtížnější jej sehnat. Možná by se dalo odmontovat
z nějaké hračky

Vzhledem k adrenalinu, napětí a vzrušení, který panuje v týmech
bych zvolil robustnější krabičku.
PS: radši si fláknu"

Myslím, že by bylo lepší odolnější zařízení, možná trochu
nižší než na obrázku, aby na něj lidi lépe dosáhli ale může
to dodat tu "pravou" soutěživost, a zároveň dlouho vydrží

Jelikož se bude hlásit o slovo celý tým, který má více lidí,
tak by asi chtělo, aby bylo jedno velké tlačítko uprostřed stolu.

Idk jestli to dobře chápu, ale to B vypadá víc týmově

Když klikneš jen sám u sebe, je to méně kolektivní. Když
se lidi perou o tlačítko, které je uprostřed, přijde mi
to jako větší zábava. Ale samozřejmě záleží dle druhu
a cíle hry.

Většinou jsou lidé do takových her tak zapálení, že vy
se mohlo stát, že by klikací verzi rozbili.

Párkrát jsem na hospodském kvízu byla a byla to taková přetlačovaná oslava kdo křiknul první dobrou odpověď byl vítěz takže pro mě kliknutí by bylo dobré na té středové krabičce by byla asi rvačka

B funguje pro všechny členy týmu. Pro určitě skupiny lidí nebezpečné, že se lidé v týmu platí navzájem. Záleží, jaká je cílovka...
A by musel mačkat jeden, takže když někdo z týmu ví, musí nejdřív předat pokyn k přihlášení... Pomalejší, kultivovanější.

Možnost B. Větší věc, menší možnost ztráty. Všichni uvidí, jaký člen z týmu se přihlásil. Navíc by se jich nemuselo vyrábět tolik, sice je to větší zařízení, ale tu malou by musel mít každý člen týmu. Velká krabice stačí jedna doprostřed stolu. Bylo by možné, že je jen jedno tlačítko do týmu. Mohl by se tak hlásit každý a ne jen ten, kdo by ji držel. Navíc při vypjatější atmosféře to bouchnutí k tomu prostě tak nějak patří.

Pokud je účel pobavení a nezáleží tolik na presnosti.
Tak B, záleží na přesném použití za mě

Cítím, že tvůj favorit je flákadlo, přesto bych uživatelsky volil klik. Uznávám, že flák je divácky a adrenalinově zajímavější. Kdybych se ale měl rychle přihlásit, už na tom chci držet palec. Škoda že nepřidal aspoň další otázky pohlaví a věk. Mohlo by se to vyprofilovat a pro různé druhy příležitostí mít více variant hlasovacích zařízení. Předpokládám, že chlapci budou raději flákat a s věkem bude stoupat klikavost.