VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Michal Molent

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SIMULACE PRŮMYSLOVÉHO PROTOKOLU CIP
CIP INDUSTRIAL PROTOCOL SIMULATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Michal Molent

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Blažek

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Michal Molent ***ID:*** 174360

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Simulace průmyslového protokolu CIP

POKYNY PRO VYPRACOVÁNÍ:

Práce je zaměřena na realizování pracoviště, které bude simulovat síťovou komunikaci průmyslových automatizačních protokolů vycházejících z protokolu CIP (Common Industrial Protocol). Student v rámci práce provede analýzu průmyslových protokolů, které vycházejí z protokolu CIP (doporučeny – ControlNET, DeviceNET, EtherNET). Dále bude zprovozněna průmyslová komunikační síť s PLC a vybranými protokoly dle jejich standardu. Výstupem práce bude pracoviště, které bude schopno simulovat reálnou komunikaci vybraných protokolů mezi dvěma a více simulovanými zařízeními. Dílčím cílem práce bude návrh a realizace scénářů komunikace (alespoň dva) vyskytujících se v průmyslovém odvětví.

DOPORUČENÁ LITERATURA:

[1] R. Štohl and K. Stibor, "Safety through Common Industrial Protocol," Proceedings of the 14th International Carpathian Control Conference (ICCC), Rytro, 2013, pp. 442-445.doi: 10.1109/CarpathianCC.2013.6560585

[2] R. Piggin, "Ethernet/IP - Control in real time," in Computing & Control Engineering Journal, vol. 17, no. 6, pp. 28-31, Dec.-Jan. 2006.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Petr Blažek

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

ABSTRAKT

Práce se věnuje rodině průmyslových automatizačních protokolů CIP, která spadá do systému SCADA. První část je zaměřena na rozbor protokolu CIP a analýzu dvou protokolů (EtherNet/IP a DeviceNet), které z protokolu CIP vychází. Druhá část se zabývá návrhem scénářů pro simulaci, dále simulací jednosměrné komunikace, obousměrné komunikace za pomocí vyčítání z konzole a obousměrné komunikaci v reálném čase mezi dvěma zařízeními Raspberry Pi 3B+. Ve třetí části je rozebrána realizace samotné simulace, spuštění, funkce předdefinovaných scénářů a grafické rozhraní pro ovládání simulace. Čtvrtá část je zaměřena na analýzu síťové komunikace v situacích, které při simulaci průmyslového protokolu EtherNet/IP nastávají.

KLÍČOVÁ SLOVA

SCADA, CIP, EtherNet/IP, DeviceNet, TCP, UDP, Raspberry Pi 3B+, python, PLC, simulace, master a slave, CPPPO, Wireshark

ABSTRACT

The thesis is about industrial automatic CIP protocol, which belongs to SCADA systems. The first part is focused on basic principles of CIP protocol and on analysis of two protocols (EtherNET/IP and DeviceNet), which are based on CIP protocol. The second part deals with designing scenarios for a simulation. The simulation of one-way communication, two-way communication with help of reads from the console and two-way real time communication between Raspberries PI 3B+. The third part deals with a realization of the simulation, a start-up and a function of predefined scenarios and graphic interface. The fourth part deals with analysis network communication in situations, which occur during a protocol EtherNet/IP simulation.

KEYWORDS

SCADA, CIP, EtherNet/IP, DeviceNet, TCP, UDP, Raspberry Pi 3B+, python, PLC, simulation, master and slave, CPPPO, Wireshark

MOLENT, Michal. *Simulace průmyslového protokolu CIP*. Brno, 2019, 148 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Blažek

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Simulace průmyslového protokolu CIP“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Petru Blažkovi za odborné vedení, konzultace a hlavně velkou trpělivost. Také bych rád poděkoval svému dlouholetému kamarádovi Ing. Jaroslavu Hájkovi za věcné rady a konstruktivní kritiku. Musím poděkovat i zaměstnancům firem Rockwell Automation s.r.o. a Siemens, s.r.o, za poskytnutí informací z praxe.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Diplomová práce se zabývá průmyslovým protokolem CIP (Common Industrial Protocol), který je v poslední době trendem v odvětví průmyslové automatizace, díky svému univerzálnímu použití. Tradiční podnikové sítě byly tvořeny pro konkrétní aplikace, se zaměřením na kontrolu, sběr informací nebo podnikovou bezpečnost. Toto řešení je na první pohled ideální, avšak z dlouhodobého hlediska zjistíme, že pokud má firma různé výrobní procesy, více odvětví nebo jiné technologie, je nutné budovat další sítě pro jednotlivé účely. V konečném důsledku to znamenalo, že jedna firma má spoustu nejednotných sítí, které nejsou spolu kompatibilní a přináší zvýšené režie na výrobu. Díky této skutečnosti začal vznikat požadavek na propojení celé podnikové infrastruktury jednou firemní sítí, pomocí které by mohl mít jakýkoliv zaměstnanec přístup k potřebným datům a byla možná bezproblémová integrace nových zařízení. Protokol CIP dokáže všechny tyto požadavky zajistit.

První část této diplomové práce se zabývá analýzou průmyslových protokolů, které vychází z protokolu CIP. Protokoly vybrané pro analýzu jsou DeviceNet, který je prvním vzniklým protokolem a EtherNet/IP, který je naopak nejmodernější. Analýza těchto dvou protokolů je zaměřena na základní parametry, struktur jednotek, využití a provedení zpráv.

Druhá část je zaměřena na návrh simulace vybraného průmyslového protokolu. V tomto případě se jedná o protokol EtherNet/IP, který je simulován za pomoci dvou jednodeskových miniaturních počítačích Raspberry Pi 3B+. Ty dokáží nahradit zařízení v reálném provozu. Po návrhu simulace jsou navrženy scénáře z reálného provozu, rozebrána hodnota zpoždění a jsou popsány typy konfigurací pro různé druhy komunikace.

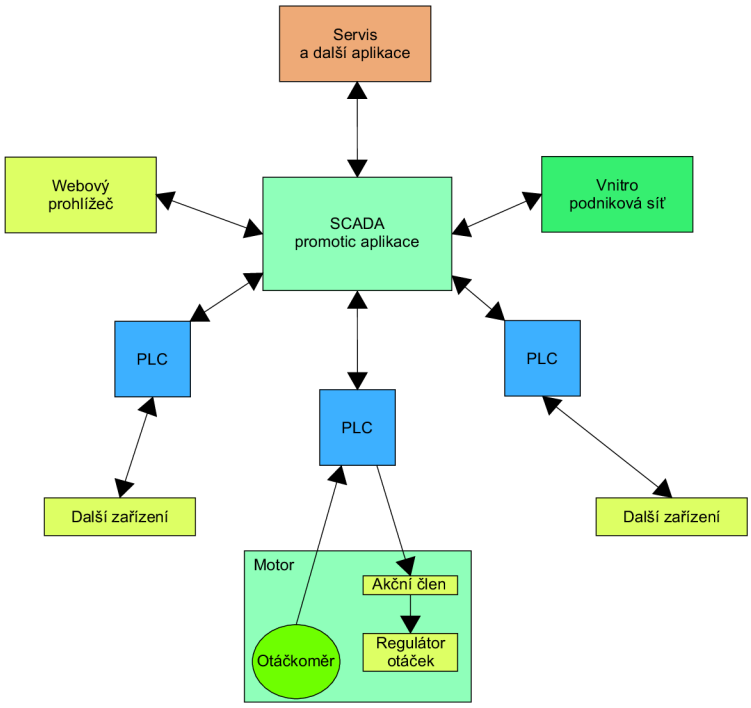
Třetí část popisuje simulaci průmyslového protokolu EtherNet/IP. Jsou zde rozebrány vytvořené knihovny v programovacím jazyce python a jejich funkce, které jsou nezbytné pro správné fungování simulace. Také jsou zde popsány funkce jednotlivých scénářů a grafické rozhraní, které bylo vytvořeno z důvodu, aby uživatel měl snazší manipulaci se samotnou simulací.

Čtvrtá část se zabývá analýzou síťové komunikace, výše zmíněného protokolu Ethernet/IP na realizovaných simulacích. Dále jsou zde popsány funkční mechanismy jednotlivých situací, které při simulaci nastávají. Analýza nejprve představuje, jak by situace teoreticky měla vypadat a následně je tato teorie potvrzena pomocí programu Wireshark.

1 SCADA

Dohledová kontrola a sběr dat v anglickém znění „Supervisory Control And Data Acquisition (SCADA)“, zajišťuje sběr informací a na základě těchto informací řídí dění v provozu. Zařízení jako jsou čidla a měřidla, informace prezentuje aplikacím. Řídící aplikace vyhodnocují stav sledovaných zařízení a vysílají řídící instrukce akčním členům, které udržují zařízení ve stavu nominálního provozu. Akční člen je jednotka sloužící k přenosu výstupního signálu z regulátoru do regulované soustavy. Zařízení v systému SCADA dělíme na nadřízené stanice (master) a podřízené stanice (slave). Nadřízené stanice v praxi bývají nejčastěji programovatelný logický automat neboli PLC (Programmable Logic Controller), ke kterému může být připojeno více podřízených zařízení pro sběr dat a zároveň více podřízených zařízení, kterým jsou zasílány řídící instrukce. PLC je průmyslový počítač pro automatizaci výrobních procesů v reálném čase. Systém SCADA je čistě softwarové řešení, které je využíváno v mnoha různých odvětvích jako je výrobní průmysl, energetický průmysl, chemický průmysl, distribuce, experimentální zařízení a další. Systém SCADA je nezávislý na operačním systému, velice adaptivní a lze jej použít jak pro menší výrobní firmy, tak gigantické závody. [1]

Na obrázku 1.1 je příklad implementace SCADA, kde je zobrazeno ovládané zařízení motor, ve kterém je umístěno zařízení pro sběr informací (otáčkoměr). Cyklicky je odesílán počet otáček v motoru za minutu na nadřazené zařízení. PLC vyhodnocuje stav zařízení a zasílá řídící instrukce pro akční člen, který reguluje otáčky v motoru. Dále také zasílá data do bloku SCADA promotic aplikací, kde probíhá vizualizace, tvoří se hlášení, vyhodnocují se trendy a události. Na blok SCADA promotic aplikace navazuje podniková síť, ve které se nachází dohledové centrum, kde jsou informace z provozu prezentovány lidské obsluze a databáze nejčastěji typu SQL, kde jsou data archivovány. Také jsou data prezentována na webový prohlížeč, aby vedoucí pracovníci mohli sledovat trendy ve výrobě. V případě poruchy nebo blížící se revize motoru jsou prostřednictví emailu na sms zprávou kontaktováni pracovníci servisu popřípadě další zaměstnanci.

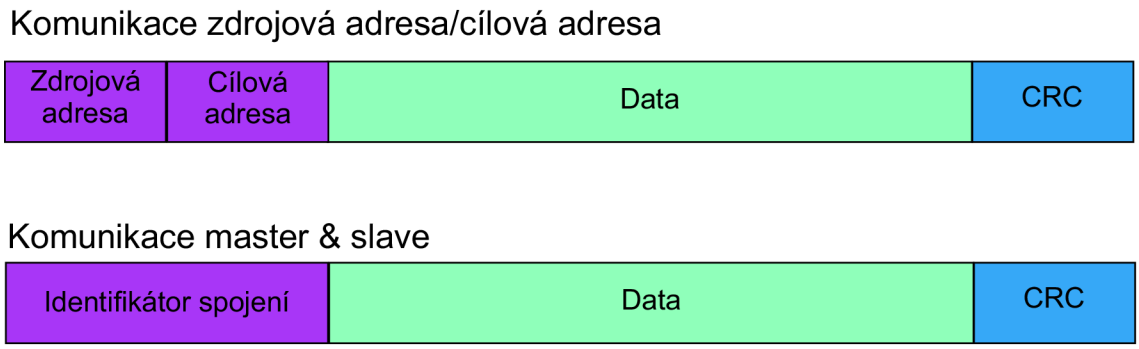


Obr. 1.1: Příklad systému SCADA v průmyslovém odvětví.

1.1 Common Industrial Protocol

Common Industrial Protocol zahrnuje komplexní sadu služeb a zpráv, které slouží k sběru informací pro aplikace zajišťující automatizaci výroby, jako je bezpečnost, řízení, konfigurace a synchronizace. Rodina protokolů CIP je univerzální s ohledem na využívané průmyslové technologie, kde je využito modelu master and slave. Nadřazená stanice (master) řídí chod celé výroby. Nadřazená stanice získává od podřízené stanice pro sběr informací (slave) data o stavu, v jakém se zařízení nachází. Na základě získaných informací nadřazená stanice posílá řídící instrukce podřízené stanici (slave). Podřízené stanice sbírají data, ale nedokáží je vyhodnotit. Díky skutečnosti, že CIP protokol využívá modelu master and slave na rozdíl od častěji využívaného schématu zdrojová adresa/cílová adresa, tak můžeme ve své podstatě říct, že se jedná o komunikaci zařízení ve skupině. Na obrázku 1.2 je vidět rozdíl mezi komunikačními modely. [4]

Každý uzel (skupina zařízení) CIP je modelován jako kolekce objektů. Objekt poskytuje abstraktní reprezentaci konkrétního komponentu v celém systému (například teploměr). Vše, co není popsáno v podobě objektu, není prostřednictvím protokolu CIP vidět. CIP objekty jsou strukturovány do tříd, instancí a atributů. [16]

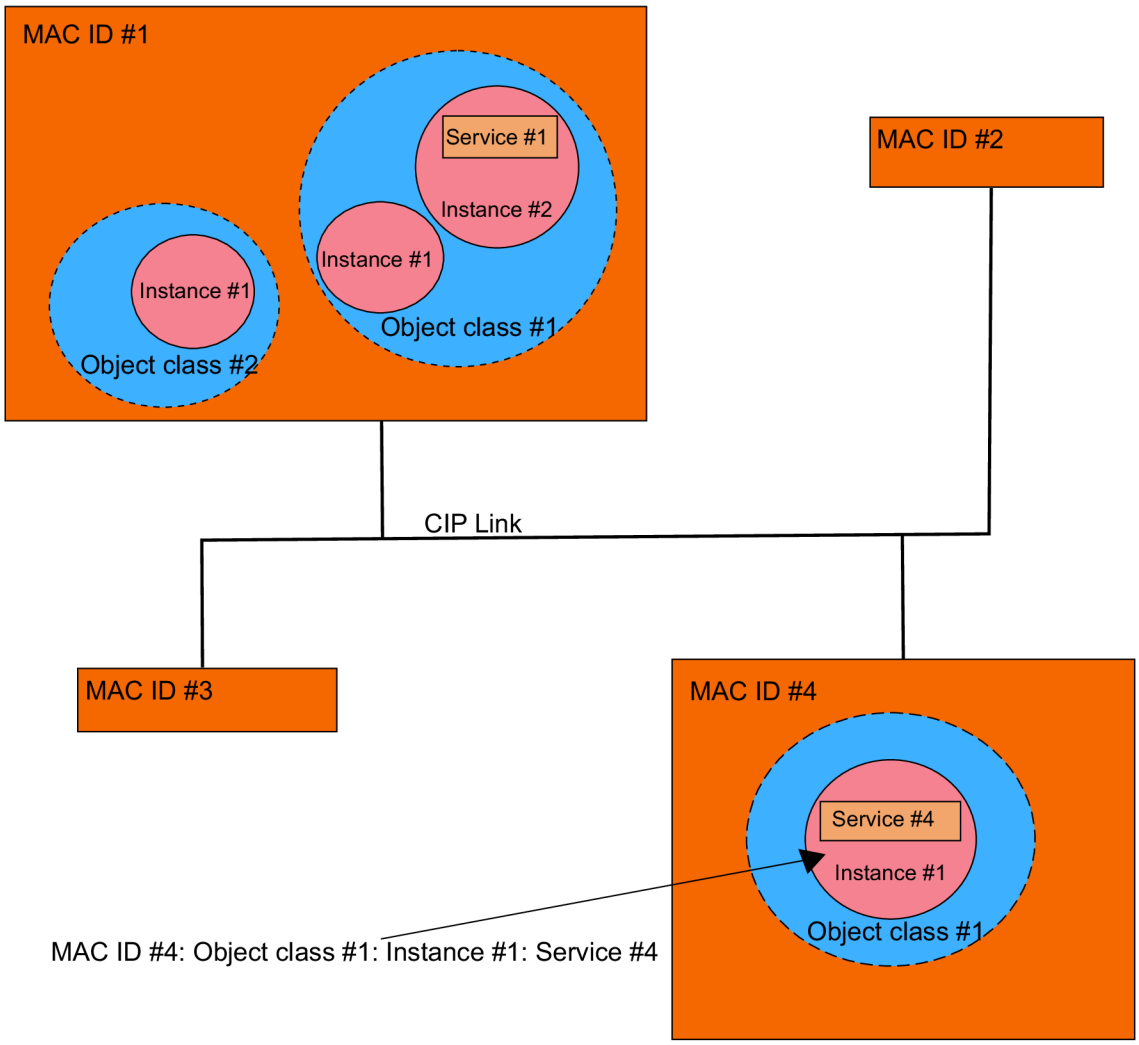


Obr. 1.2: Rozdíl mezi komunikačními modely.

Třída je sada objektů, do které spadají všechny systémové komponenty stejného druhu. Instance objektu je jeho skutečná reprezentace pro konkrétní objekt v rámci dané třídy. Atributy jsou stejné pro instance v jedné třídě, ale se svou vlastní sadou hodnot pro tyto atributy.

1.1.1 Komunikace protokolu CIP

Protokol CIP využívá jednotné schéma adresování pro objekty a jejich podčásti. K funkčnosti tohoto schématu je nutné odlišit od sebe jednotlivé prvky. Pro toto rozlišení jsou využity identifikátory. **MAC ID** - Media Access Control Identifier je identifikátor uzlu v síti CIP, který je prezentován celočíselnou identifikační hodnotou.
CLASS ID - identifikátor třídy je celočíselná hodnota přiřazena každé třídě objektů, která je přístupna ze sítě.
INSTANCE ID - identifikátor instance je celočíselná hodnota přiřazena každé instanci objektu v dané třídě.
ATTRIBUT ID - identifikátor atributu je celočíselná hodnota přiřazena každému atributu třídy nebo instance.
SERVICE CODE - servisní kód je celočíselná hodnota přiřazena objektu nebo třídě, která označuje jeho konkrétní funkci. [5]
Příklad schéma adresování můžete vidět na obrázku 1.3. Zde jsou vidět čtyři uzly v síti MAC ID #1–#4. Teď podrobně rozebereme uzel s MAC ID #4. Nachází se v něm třída objektů class ID #1, ve které je instance s ID #1 a ta má v sobě zařízení se servisním kódem #4. V reálném provozu by toto logické členění znamenalo, že v síťovém uzlu #4 se nachází třída #1, například třída pro správu motoru. V této třídě je instance #1 typicky otáčky motoru. Uvnitř instance #1 je pouze jedno zařízení poskytující informace o otáčkách motoru a to otáčkoměr se servisním kódem #4.



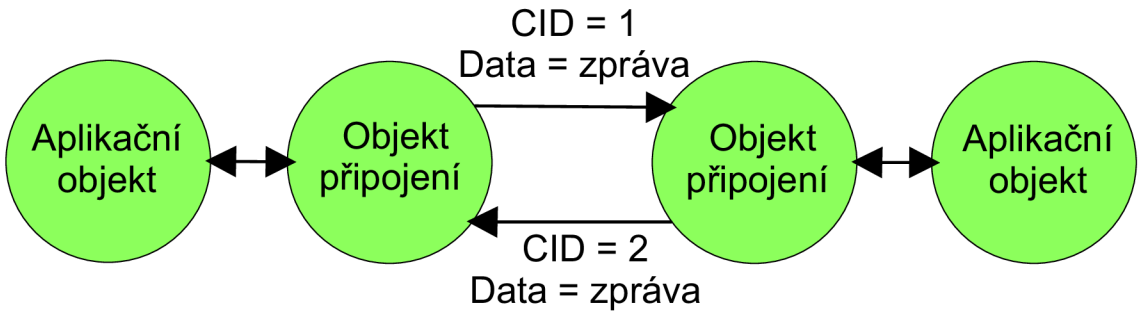
Obr. 1.3: Schéma adresování CIP.

CIP je spojově orientovaný protokol, který po připojení poskytuje cestu mezi více aplikačními objekty a po té co je spojení navázáno dostane identifikátor spojení neboli CID (Connection Identificator). Pokud je zapotřebí obousměrné spojení, jsou přiřazeny dvě hodnoty CID. Podle typu sítě je definován formát CID. Pro navázání spojení mezi zařízeními, které ještě nejsou připojena, byl vytvořen proces správce nespojených zpráv UCMM (Unconnected Message Manager), který je zodpovědný za zpracování nepřipojených explicitních požadavků a odpovědí. Pro navázání spojení je odeslání zprávy o požadavku na službu UCMM Forward Open. Ve zprávě Forward Open jsou obsažena všechna potřebná data, k vytvoření spojení mezi zařízeními. Výměna dat je realizována jako jednosměrná nebo obousměrná. V otevřené žádosti „Foward Open“ jsou obsaženy následující informace:

- informace o vypršení časového limitu pro toto připojení,

- CID pro připojení od stanice k cíli,
- CID pro připojení od cíle k stanici,
- informace o totožnosti stanice (ID dodavatele a sériové číslo),
- maximální velikost zpráv pro toto spojení,
- typ spojení (mezi dvěma zařízeními nebo ve skupině),
- spouštěcí mechanizmus (cyklický nebo při změně stavu),
- cesta,
- elektronický klíč (volitelné),
- datový segment obsahující konfigurační informace pro uzel (volitelné),
- informace o směrování, v případě připojení do více sítí (volitelné).

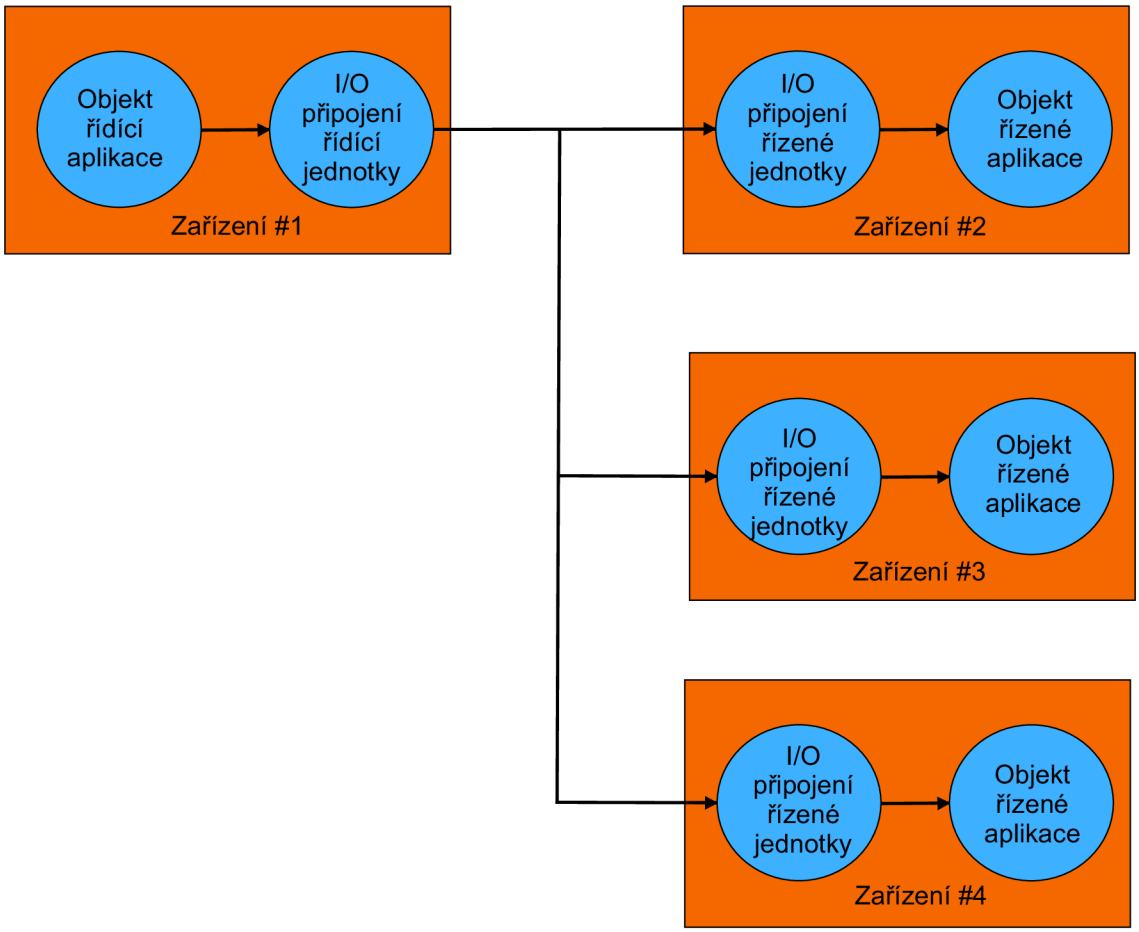
Komunikace objektů je na obrázku 1.4. Princip komunikace objektů spočívá v tom, že aplikační objekty se vůbec nestarají o připojení a komunikaci. Svá data předávají objektu připojení, ten naváže spojení k druhému objektu připojení s identifikátorem připojení CID=1, který předá data svému aplikačnímu objektu. Pro odpověď je postup stejný jen se naváže druhé spojení pro odpověď s identifikátorem připojení CID=2. [3]



Obr. 1.4: Komunikace objektů.

1.1.2 Typy komunikace u protokolu CIP

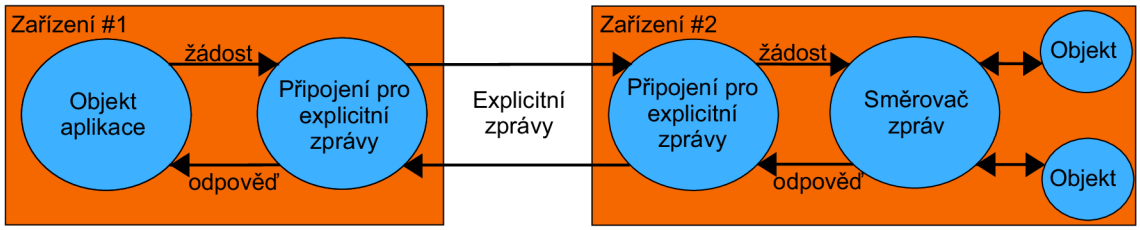
I/O připojení, které je někdy nazýváno jako implicitní zasílání zpráv, poskytuje vyhrazené účelové komunikační cesty mezi řídící aplikací (master) s jednou nebo více řízenými aplikacemi (slave). Zprávy jsou zasílány mezi dvěma zařízeními nebo ve skupině dle potřeby. Příklad tohoto spojení můžeme vidět na obrázku 1.5, kde máme objekt řídící aplikace, za kterou se nachází objekt připojení řídící jednotky pro spojení typu I/O a ten je připojen k objektům pro řízené jednotky pro spojení typu I/O a následně přeposílá data objektům řízených aplikací. [7]
Explicitní zprávy představují orientovanou komunikaci na bázi dotaz/odpověď typu bod/bod neboli point-to-point, kde je přímo uvedeno, ze které služby nebo objektu



Obr. 1.5: Spojení I/O.

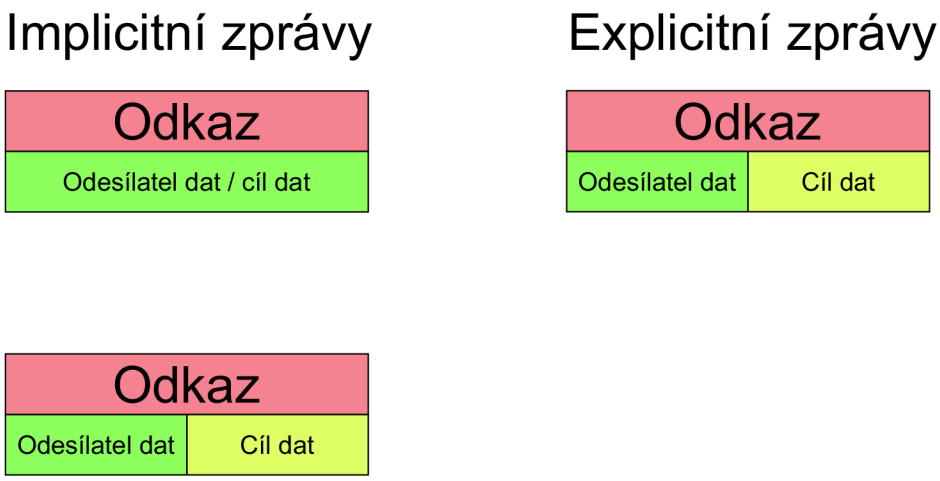
mají data pocházet. Využívá k tomu obecné nebo víceúčelové komunikační cesty mezi dvěma zařízeními. Příklad tohoto typu spojení máme na obrázku 1.6. Objekt aplikace zařízení #1 vysílá žádost do objektu připojení pro explicitní zprávy. Ten pak zasílá žádost ve formě explicitní zprávy objektu pro připojení explicitních zpráv v zařízení #2, kde je explicitní zpráva přetransformována opět na žádost a předána směrovači zpráv, který ji předává danému objektu. Objekt vytvoří odpověď na žádost, kterou předává směrovači pro zprávy. Ten ji opět posílá objektu připojení pro explicitní zprávy, kde ji přetransformuje na explicitní zprávu a odešle objektu připojení pro explicitní zprávy v prvním zařízení a formou odpovědi je předána původci žádosti, tedy objektu aplikace.

Středem veškeré komunikace u rodiny protokolů CIP jsou komunikační objekty, které spravují a zajišťují výměnu zpráv za provozu. Každý komunikační objekt obsahuje odkaz, který se skládá z části od odesílatele dat nebo části od cíle dat, popřípadě části od každého z nich. U připojení I/O jsou dvě možnosti tvoření odkazu komunikačních objektů. První z nich nám dává na výběr, jestli je z části komunikačního



Obr. 1.6: Spojení za pomocí explicitních zpráv.

objektu od odesílatele dat nebo cíle dat. Druhá možnost udává, že je složen z částí od obou účastníků, tedy od odesílatele dat a cíle dat. U explicitní zprávy je oproti předchozímu typu připojení pouze jedna možnost a to s kombinací dvou částí. Jedna od odesílatele dat a druhá od cíle dat. Tvoření odkazu je znázorněno na obrázku 1.7.



Obr. 1.7: Tvoření odkazu.

Hodnotu atributu v komunikačních objektech definuje několik sad atributů, které slouží k popisu důležitých parametrů pro funkčnost tohoto spojení. Explicitní zprávy jsou vždy směrované do komunikačního objektu, nazývaným směrovač zprav. Hodnoty atributu v komunikačních objektech nám určují:

- zda se jedná o komunikaci I/O nebo pomocí explicitních zprav,
- maximální velikost dat,
- zdroj dat,
- cíl dat.

Další atributy nám definují stav a chování připojení. Bezpochyby je nejdůležitějším prvkem způsob spouštění zprávy. Zde je hned několik způsobů, z jakého důvodu zprávu odeslat. Prvním z nich je změna stavu neboli COS (Change Of State), což

znamená, že se změnil jistý sledovaný stav, například při zjištění maximálního stavu hladiny v nádrži, zařízení zprávu o změně okamžitě posílá své nadřazené stanici. Dalšími důvody jsou cyklické odesílání dat prostřednictvím síťových událostí, načasováním připojení a předefinovanou akcí. CIP umožňuje souběh více připojení na jednom zařízení.

Pro snazší implementaci protokolu CIP je možnost využít knihovnu objektů. Jedná se o sbírku běžně definovaných objektů, které jsou v sadách tříd objektů a ty dělíme do následujících skupin:

- Pro obecné použití,
- pro konkrétní aplikace,
- specifické pro síť.

Předdefinované objekty nejsou vhodné pro všechny síťové úpravy CIP. Důvodem je, že některé objekty mohou požadovat omezení nebo specifické úpravy pro jednotlivé varianty sítě s protokolem CIP. Přehled nejčastěji používaných objektů, které jsou předdefinovány v knihovně objektů, se nachází v tabulce, která je v příloze A.2.

Jak je již z názvu dělení knihovny objektů patrné, tak předdefinované objekty pro obecné použití můžeme nalézt v mnoha různých zařízeních. Oproti tomu předdefinované objekty pro konkrétní aplikace můžeme najít pouze v zařízeních, které tuto aplikaci podporují. Knihovna je průběžně doplňována o další objekty, které přidávají je vývojáři ze společnosti ODVA. Má speciální zájmovou skupinu SIG. Také jsou silně podporování externí vývojáři, protože politická myšlenka je taková, že je lepší položit základ předdefinovaného objektu, vytvořit jeho společnou definici a přidat jej do knihovny, než aby vznikala spousta soukromých objektů. Příkladem třídy pro externí vývojáře je třída ID 100 - 199. Neexistuje jedna veřejná knihovna předdefinovaných objektů, protože některé předdefinované objekty jsou soukromé. Zařízení využívají typicky pouze podmnožinu z předdefinovaných objektů. Typické zařízení využívá tyto objekty:

- Objekt pro identifikaci,
- objekt správce připojení nebo objekt připojení,
- objekt směrovač zpráv nebo alespoň jeho funkce,
- jeden nebo více propojovacích objektů, který je specifický pro danou síť .

Následně se již přidávají objekty specifické pro zařízení. Tímto způsobem implementace je dosažena co nejnižší režie, protože zařízení není zbytečně zatěžováno objekty, které nevyužívá.

Pro přiblížení funkce objektu CIP jsou dále popsány objekt identity ID třídy 0x01, který se řadí k základním objektům a jsou na něm dobře znázorněny principy objektů CIP. Také každé zařízení musí obsahovat objekt identity. Většina zařízení podporuje pouze jednu instanci objektu identity. Povinné atributy:

- Typ zařízení,

- ID dodavatele,
- sériové číslo,
- kód produktu,
- jméno výrobku,
- revize,
- stav.

Volitelné atributy:

- Aktivní jazyk,
- přiřazené jméno,
- přiřazený podpis,
- state,
- interval prezenčního signálu,
- seznam podporovaných jazyků,
- mezinárodní název produktu,
- konfigurační konzistenční hodnota,
- geografická lokace,
- semafor,
- režim ochrany,
- modbus identifikační informace.

Nyní jsou jednotlivé atributy přiblíženy podrobněji:

Typ zařízení je UINT hodnota, tedy jednoznačný identifikátor, určující profil, který byl použit pro dané zařízení.

ID dodavatele identifikuje dodavatele, který zařízení prodává. Tato hodnota je celosvětově unikátní (UNIT) a je přiřazována společností ODVA. U každé sítě CIP od stejného dodavatele je stejná UINT hodnota.

Sériové číslo je 32 bitové číslo, které v kombinaci s ID dodavatele tvoří jedinečnou identifikaci zařízení. Žádné dvě CIP zařízení od jednoho dodavatele nesmí mít stejné sériové číslo.

Kód produktu je opět UINT číslo, které definuje výrobce zařízení. Většinou se jedná o volné spojení mezi kódem produktu a katalogovým číslem. Využívá se k rozlišení více produktů od stejného dodavatele se stejným ID dodavatele.

Jméno výrobku se skládá až z 32 znaků a umožňuje dodavateli dát zařízení smysluplný název pomocí řetězce ASCII.

Revize se skládá ze dvou hodnot USINT, pro hlavní revizi a vedlejší menší revize. Když změníme nastavení zařízení, jako je změna chování zařízení v síti nebo změny v logickém rozhraní zařízení, tak se tato změna projeví minimálně ve vedlejší revizi.

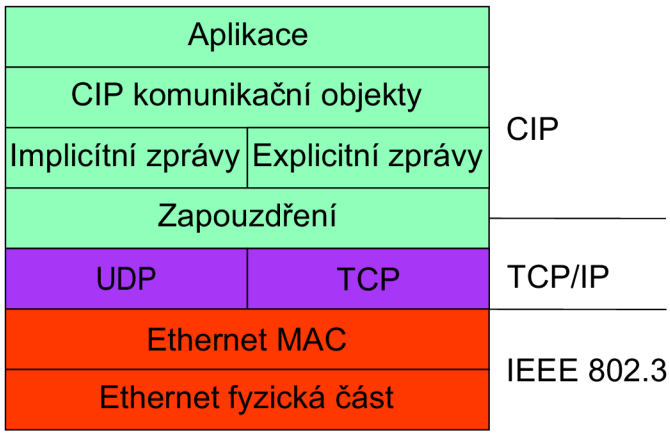
Stav je atribut, který poskytuje informace o stavu daného zařízení, jako je stav konfigurace, jestli je ovládáno jiným zařízením a zda se vyskytly závažné nebo menší poruchy.

Aktivní jazyk určuje, který z podporovaných jazyků se používá.
Přiřazené jméno je atribut, který nastavuje uživatel zařízení, aby pojmenoval dané zařízení dle vlastní potřeby.
Přiřazený podpis obdobný atribut jako přiřazené jméno.
State obdobný atribut jako stav, ale méně podrobný a využívající pouze jednu hodnotu UINT.
Interval prezenčního signálu umožní zasílání zprávy prezenčního signálu. Jedná se o nepřipojenou zprávu o změně stavu, u které lze nastavit cyklický interval na pozadí v rozsahu 1 – 255 sekund. Momentálně lze tento atribut definovat pouze u konfigurace sítě typu DeviceNet.
Seznam podporovaných jazyků. Jak již z názvu vyplývá, jedná se o seznam jazyků, které jsou podporovány.
Mezinárodní název produktu lze použít k popisu produktu ve více podporovaných jazycích.
Konfigurační konzistenční hodnota, je hodnota, která napomáhá v rozlišení zařízení, které již nakonfigurováno je a zařízením, které nakonfigurované není nebo rozlišit různé konfigurace, které jsou na jednom zařízení. Za pomocí tohoto atribut se lze vyhnout zbytečnému stahování konfigurace, která již je na zařízení nakonfigurována.
Geografická lokace je nastavitelná individuálně uživatelem zařízení k odlišení od ostatních na základě geografické polohy.
Semafor je nástroj pro synchronizaci přístupu klientů do zařízení.
Režim ochrany indikuje současný režim ochrany zařízení.
Modbus identifikační informace poskytuje informace o identitě formátu Modbus, jaký dané zařízení podporuje. [10]

1.2 EtherNet/IP

EtherNet/IP je jedním z čtyř členů rodiny protokolů CIP, který byl představen v roce 2000. EtherNet/IP je zkrátka pro ethernet industrial protokol neboli ethernet průmyslový protokol, který nesmí být zaměňován s Ethernet TCP/IP, u kterého koncovka IP znamená internetový protokol. Při použití protokolu CIP na horní vrstvě je klasický Ethernet TCP/IP rozšířen na EtherNet/IP, který je ideální pro použití v průmyslovém prostředí, jak je ukázáno na obrázku 1.8. Jednou z výhod použití EtherNetu/IP je možnost koexistov jakýmkoliv jiným protokolem na nižší vrstvě než transportní. Tím pádem jej lze běžně připojit k internetu, bez potřeby použití speciálních síťových prvků. Jediný požadavek oproti běžnému Ethernetu TCP/IP

je mechanismus pro zapouzdření zpráv CIP do Ethernetových rámců. EtherNet/IP pracuje podle hierarchického modelu ISO/OSI pro síťovou komunikaci. [8]



Obr. 1.8: Hierarchie u EtherNet/IP.

Jedná se o transportní protokol, který využívá dva mechanismy pro zapouzdření:

- Spolehlivým přenose - TCP,
- nespolehlivý přenosem - UDP.

Díky délce ethernetových rámců implementace CIP EtherNet/IP nemá žádné zvláštní omezení a při použití plně duplexní komunikace nemůže dojít ke ztrátám paketů a kolizím. Zřídka kdy je potřeba využít u ethernetových rámců fragmentaci, protože jejich délka je dostatečná. Pokud by přeci jen bylo potřeba fragmentovat, je tento proces proveden automaticky podle typu komunikace, buďto pomocí TCP/IP nebo UDP/IP. [9]

EtherNet/IP má následující vlastnosti:

- Systém je postaven na standardní infrastruktuře.
- Nemá omezení na počet uzlů v síti.
- Lze použít subneting neboli strukturovat síť do podsítí.
- Podporuje přenosovou rychlost 10 Mbit/s, 100 Mbit/s a 1000 Mbit/s.
- Je kompatibilní se standardem IEEE 802.3.
- Ve stejné podsíti může být provozovaná komunikace pracující v reálném čase a komunikace, která v reálném čase nepracuje.
- Spolupracuje s protokoly aplikační vrstvy jako jsou například FTP, HTTP, DNS a další.

1.2.1 Adresování v sítích EtherNet/IP

Adresování v průmyslových sítích se liší od klasického adresování v IT sítích. Běžnému uživateli nezáleží na tom, jakou má přidělenou logickou adresu, ale je pro něj

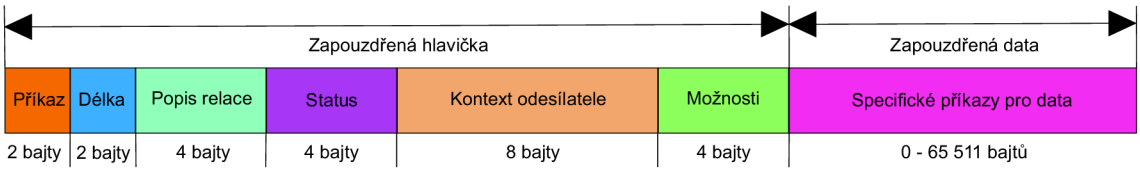
důležité, že může v dané síti komunikovat a proto mu může být přidělena dynamická logická adresa. Vlastnost dynamické logické adresy je, že se při každém restartu zařízení může změnit na jinou v daném rozsahu. U průmyslových sítí by systém přidělování logických adres nemohl fungovat z důvodu, že by zařízení nevěděla, jestli má například nadřazené zařízení pořád stejnou logickou adresu nebo ji už má dynamicky zaměněnou a jeho logickou adresu už má jiné zařízení, se kterým nepotřebuje komunikovat. Z tohoto důvodu je nejlepším způsobem používat statické nastavení adres. U EtherNetu/IP se jako výchozí rozhraní využívají Ethernetové porty, protože jsou to jediné porty, které mají všechna zařízení v síti. V případě, kdy jsou logické adresy nastavovány staticky pracovníky, může vlivem lidské chyby vzniknout situace, kdy dvě nebo více zařízení mají stejnou logickou adresu, což je nežádoucí. Aby byly takové konflikty adres zjištěny a řešeny může být v síti spuštěn mechanizmus pro zjišťování konfliktů adres neboli ACD (Address Conflict Detection).[20]

1.2.2 Zapouzdření EtherNet/IP

Zapouzdření u EtherNetu/IP je možné provést dvěma způsoby Ethernetu TCP/IP a UDP/IP, které není třeba nijak upravovat. Každý způsob má své majoritní využití.

Ethernet TCP/IP se využívá pro explicitní zprávy na portu 0xAF12, které se využívají například pro příkazy zapouzdření při nastavování komunikace mezi uzly. Na port 0xAF12 mohou být zaslány příkazy pro zapouzdření zasílané pomocí UDP datagramů. Záhlaví zapouzdření obsahuje příkaz, který určuje význam dat zapouzdření, například společný formát paketů.

Zapouzdřený paket je na obrázku 1.9. Kde pole příkaz určuje typ zprávy přehled příkazu je k nahlédnutí v tabulce A.1 umístěné v příloze A. Následující pole udává celkovou délku zprávy. Popis relace je odpověď na požadavek zaregistrování (Register Session). Status udává, zda byl požadovaný příkaz na zapouzdření vykonán, přehled kódů pro status je umístěn v tabulce, která je v příloze E. Kontext odesílatele je hodnota přiřazená odesílatelem. Pole možnosti je nastaveno na hodnotu 0 odesílatelem, pokud má jinou hodnotu, je paket zahozen. Specifické příkazy pro data je závislý na hodnotě v poli příkaz, podle toho má danou strukturu.



Obr. 1.9: Zapouzdření EtherNet/IP.

Ethernet UDP/IP je využit oproti tomu k zasílání implicitních zpráv pomocí portu 0x08AE. Zprávy mohou být zasílány jednomu nebo více příjemcům. Efektivita vícesměrového vysílání je zajištěna využitím dostupné šířky pásma. Řídí se běžným formátem paketů, který nevyužívá záhlaví zapouzdření. [6]

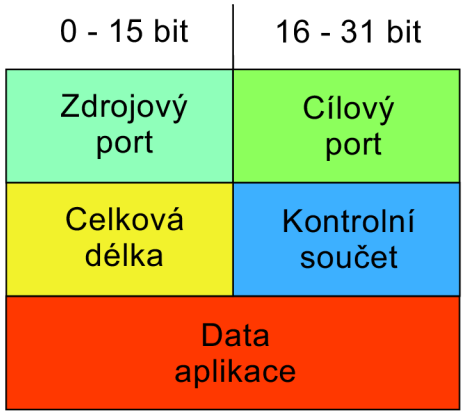
1.2.3 Struktura jednotky EtherNet/IP

EtherNet/IP používá standardní jednotky Ethernet TCP/IP a UDP/IP, které jsou popsány níže, aby byla jasná struktura a použití těchto služeb. Protokol UDP využívá datové jednotky nazvané datagramy a protokol TCP využívá datové jednotky nazvané segmenty.

Datagram UDP

Jedná se o jednoduchý transportní protokol, který poskytuje službu bez spojení, což znamená, že doručuje datagramy na cílový port, aniž by bylo odesílateli potvrzeno, že byl datagram v pořádku doručen. Využívá se v případě, že ztráta některých datagramů nemá dopad na funkce vyšších vrstev. Díky této skutečnosti jej lze označovat za nespolehlivý. Největší výhody této služby jsou minimální zpoždění a minimální režie.

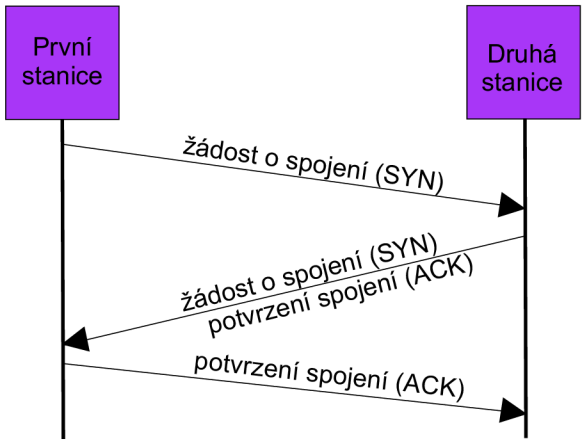
Datagram protokolu UDP je na obrázku 1.10, kde je uvedeno záhlaví UDP, které má velikost 8 bajtů a je složeno z zdrojového a cílového portu. Dále záhlaví obsahuje pole celkové délky, jehož hodnota je dána v bajtech a udává celkovou délku datagramu i se záhlavím. Poslední položka záhlaví je kontrolní součet, který slouží jako ochrana před základními chybami, které mohou vzniknou při přenosu dat. Zbylá část datagramu jsou zapouzdřená aplikační data. [18]



Obr. 1.10: UDP datagram.

Segment TCP

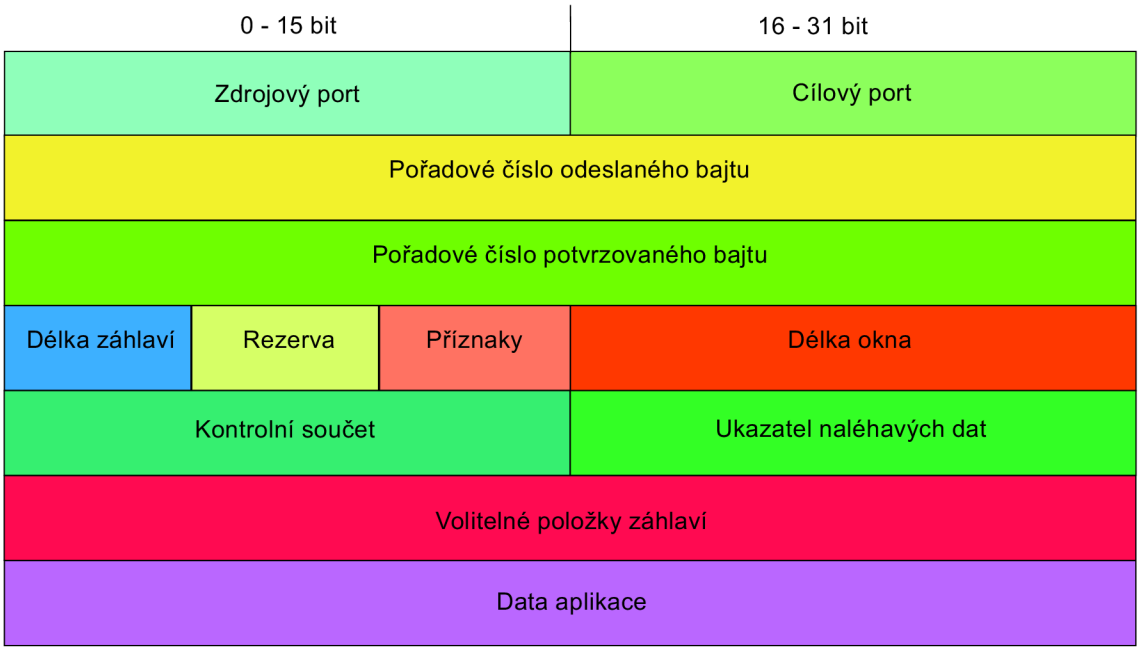
Oproti transportnímu protokolu UDP má transportní protokol TCP mnohem rozsáhlejší funkce. Mezi ty nejvýznamnější patří číslování bajtů, řízení toku dat, řízení chybových stavů a řízení při stavu zahlcení. Před samotnou komunikací musí být sestaveno spojení. Navázání spojení je provedeno ve stylu tří-cestného podání ruky „three-way handshake“. Tří-cestné podání ruky je mechanismus, kdy první stanice vyšle žádost o spojení druhé stanici, která sama začne navazovat spojení na první stanici a zároveň potvrzuje navázání prvního spojení. Posledním krokem je, že první stanice potvrdí navázání spojení druhé stanic. Navázání spojení je vidět na obrázku 1.11. Dále v případě ztráty nebo poškození segmentu TCP zajišťuje znovu zaslání požadavku a tím spolehlivost při doručení dat. Jeho využití je pro aplikace, které pro svou správnou funkci potřebují všechna přenesená data. Má mnohem větší režii a zpoždění než datagramy UDP. [17]



Obr. 1.11: Navázání spojení-třícestné podání ruky.

Na obrázku 1.12 můžeme vidět segment TCP, který se také skládá ze záhlaví, jež může mít délku až 60 bajtů. Stejně jako UDP datagram obsahuje pole zdrojový port, cílový port a kontrolní součet, které mají obdobnou funkci jako u datagramu. Další pole v záhlaví segmentu jsou pořadové číslo odeslaného bajtu, tato hodnota udává pořadové číslo prvního z odeslaných bajtů v daném segmentu a pořadové číslo potvrzovaného bajtu, kde je uvedena hodnota dalšího bajtu který očekává. Délka záhlaví udává délku celého záhlaví, protože záhlaví může mít proměnou délku od 20 bajtů po 60 bajtů. Příznakové bity jsou složeny ze šesti položek, které ukazují jak má být s daty zacházeno. Tyto příznaky jsou pro naléhavá data (URG), potvrzení platnosti přijatých dat (ACK), data jsou předávána aplikaci a nesmí čekat na další segmenty (PSH), odmítnutí spojení (RST), navázání spojení nebo nové číslování sekvencí bajtů (SYN) a ukončení spojení (FIN). Délka okna je hodnota, která určuje

kolik bajtů může být odesláno, než přijde potvrzení o přijetí. Ukazatel naléhavých dat je pole navazující na příznakový bit (URG). Za polem volitelné položky záhlaví jsou již odesílána data aplikace. [19]



Obr. 1.12: Segment TCP.

1.2.4 Společný formát paketů

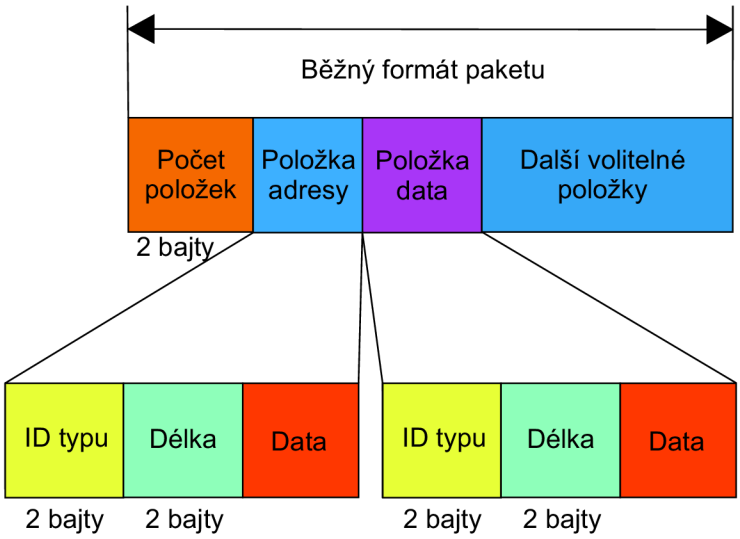
Společný formát paketů neboli Common Packet Format (CPF) je způsob strukturovaného formátování pole zapouzdření dat. Jsou zde definovány položky: [12]

- Počet položek,
- položka adresy - může se skládat z více položek,
- položka data - může se skládat z více položek,
- další volitelné položky.

Když to příkaz vyžaduje, je do jednoho rámce zapouzdření vloženo více položek, jak je vidět na obrázku 1.13.

1.2.5 Explicitní zprávy u protokolu EtherNet/IP

Explicitní zprávy využívají protokol TCP/IP a lze je odesílat dvěma možnými způsoby. První z nich je možnost bez spojení, ke kterému je použit příkaz „SendRRData Encapsulation“. Využívá se mechanismu obecných zdrojů v uzlech, ale to však nemusí být úplně ideální z důvodu, že tyto obecné zdroje mohou být vysoce využívány. V praxi se explicitní zprávy bez spojení využívají pouze v případech, kdy aplikace



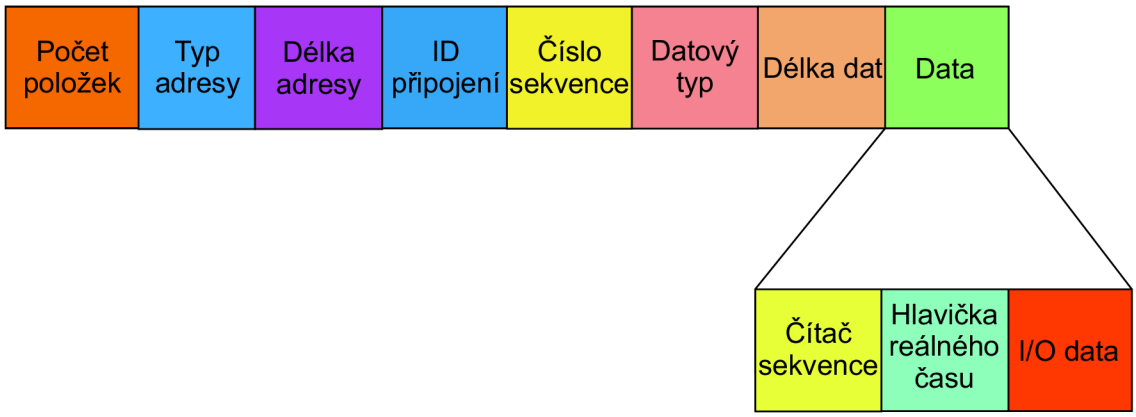
Obr. 1.13: Společný formát paketů.

posílá požadavky v málo častých intervalech nebo nepravidelně. Druhou možností jsou explicitní zprávy se spojením, pro které slouží příkaz „SendUnitData Encapsulation Command“. Před samotnou výměnou zpráv musí být sestaveno spojení, které má vyhrazeny všechny prostředky potřebné pro toto spojení a tyto prostředky udržuje po celou dobu existence spojení. Tím je docíleno včasného doručení žádostí a odpovědí. Toho využívají aplikace, které pravidelně nebo často vysílají žádosti. [11]

1.2.6 Implicitní zprávy u protokolu EtherNet/IP

Implicitní zprávy neboli I/O zprávy využívají protokol UDP/IP. Zpráva má společný formát paketu, ale nevyužívá hlavičku zapouzdření. Komunikaci lze provést, jako jednosměrové vysílání UDP rámce, které můžou být vysílány od původce dat do cíle nebo naopak. V případě potřeby se nabízí možnost vícesměrového vysílání, které může být vysíláno pouze od cíle k původci. Výhodou vícesměrového vysílání je, že více jak jedno zařízení může získávat vstupní data. Nevýhodou je, že se mohou UDP rámce šířit sítí a tím ji zahlcovat. Řešení tohoto problému je vytvoření skupin pro vícesměrové vysílání. Do těchto skupin se zařízení přihlásí a odebírají pakety. Přepínač šíří vícesměrové vysílání pouze tam, kde se nachází zařízení, které o něj má zájem. Vhodné k tomuto účelu jsou přepínače podporující IGMP Snooping. Tyto přepínače tvoří skupiny pro vícesměrové vysílání automaticky. [21] Příklad UDP rámce je na obrázku 1.14.

Čítač sekvencí v sobě nese 16 bitovou hodnotu sekvence, která se nepoužívá pro připojení bezpečného CIP (CIP safety). Hlavička reálného času je zde použita z důvodu vzájemné spolupráce prvků systému. Poslední částí jsou I/O data.



Obr. 1.14: Implicitní zprávy EtherNet/IP.

1.3 DeviceNet

DeviceNet je dalším protokolem CIP, který byl vytvořen a implementován, jako úplně první pro protokol CIP. Pracuje podle hierarchického modelu ISO/OSI pro síťovou komunikaci, který definuje vše od fyzického rozhraní až po aplikace. Pro sběr informačních dat a sdělování řídících instrukcí, jak v jedné sítí, tak i mezi více sítěmi. DeviceNet je založen na síti typu „ovladač místní sítě“ neboli „Controller Area Network (CAN)“ a využívá podmnožinu tohoto protokolu. CAN je sériová komunikační sběrnice navržená k poskytování jednoduché a efektivní komunikace. DeviceNet je v určitých směrech oproti ostatním protokolům omezený. Například je přizpůsobena velikost paketu na 8 bajtů, které standardně využívá protokol CAN. Naopak je jeho výhodou možnost pracovat na jednoduchých zařízeních, která mají minimální procesní výkon. DeviceNet nepoužívá v záhlaví hodnotu reálného času nebo hodnotu počtu sekvencí. Je určen pro komunikaci koncových bodů, jako jsou senzory a akční členy. Nyní zde jsou vypsány základní vlastnosti DeviceNetu:

- Podporuje až 64 uzlů v síti, které mohou být připojovány nebo odpojovány za provozu.
- Podporuje přenosovou rychlost 125 kbit/s, 250 kbit/s a 500 kbit/s.
- Je flexibilní podle individuálních potřeb.
- Je aplikována ochrana proti chybnému zapojení.

DeviceNet byl vytvořen tak, aby mohl být provozován na základních typech mikrokontrolerů. Nejméně hardwarově náročná verze protokolu DeviceNet vyžadovala 8 bitový mikroprocesor, RAM paměť o velikosti 175 bajtů a 4 kilobajty vnitřní paměti.

1.3.1 Struktura rámce

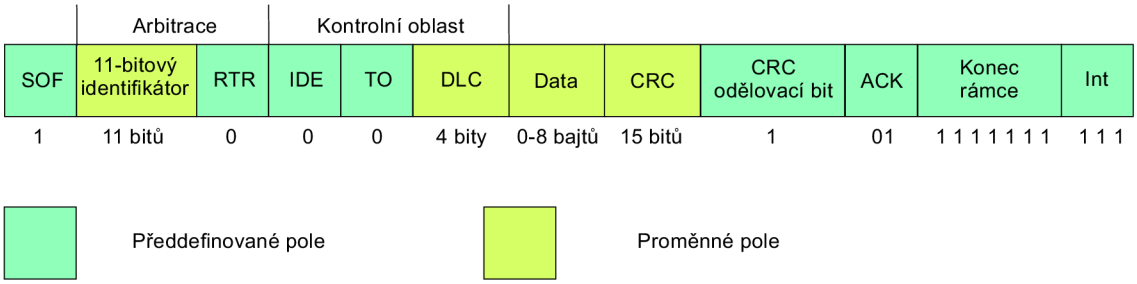
V této podkapitole je nejdříve představena struktura rámce CAN, vysvětlena jeho funkce a důvod použití 11-bitového identifikátoru v standardním provedením CAN. Následně je objasněno využití toho identifikátoru u DeviceNetu.

Struktura rámce CAN

CAN je asynchronní sériová datová sběrnice s více mastery, která k určení přístupu používá rozlišení „CSMA/CR (Carrier Sense Multiple Access/Collision Resolution)“. Komunikace probíhá čtyřmi typy rámců:

- Datové rámce,
- rámce přetížení,
- chybové rámce,
- RTR rámce (Remote Transmit Request).

Tvar datového rámce můžeme vidět na obrázku 1.15. Položka SOF je startovacím bitem, který signalizuje přechod sběrnice do dominantního stavu z stavu recesivního. Jedenácti bitový identifikátor slouží k řízení přístupu na sběrnici, hodnota identifikátoru určuje prioritu zprávy, význam zprávy a je v něm i adresa odesílatele. Vzdálený požadavek neboli RTR (Remote Request) udává, zda se jedná o vysílání dat nebo ne. Položky IDE a TO jsou rezervovány pro budoucí využití. DLC informuje o délce dat, tedy kolik bude přeneseno datových bajtů, maximální povolený počet je 8 bajtů. Pole pro cyklický redundantní součet neboli CRC (Cyclic redundancy check) slouží pro ověření zda nedošlo ke změně hodnot v rámci během přenosu. CRC oddělovací bit má za úkol vytvořit prodlevu pro zpracování pole CRC. Pole ACK slouží k potvrzení správnosti přenosu, po kterém následuje pole indikující konec rámce. Poslední pole (Int) slouží jako mezera mezi zprávami.



Obr. 1.15: Struktura rámce CAN.

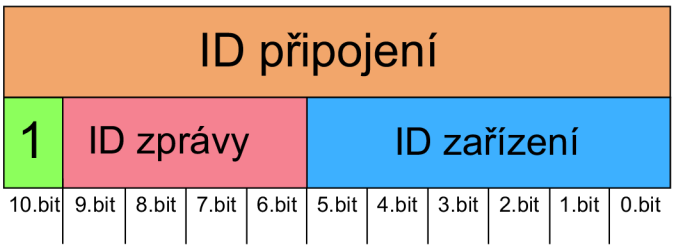
11-bitový identifikátor je jedinečná hodnota u standardní formátu, kterou má každý datový rámec a je využit u DeviceNetu. Protokol CAN umožňuje i rozšířený

formát. Formou 29-bitového identifikátoru, který se však DeviceNet nevyužívá. Identifikátor je nejprve použit pro určení, kterému zařízení je umožněno odesílat zprávy při soupeření o přistup na sběrnici. Poté se identifikátory využívají pro filtraci nežádoucích zpráv na straně přijímače. [13]

Identifikátor CAN využitý v DeviceNet

Z důvodu použití 11 bitového identifikátoru může rozlišovat 2048 typů zpráv. Mechanismus CAN určuje prioritu zpráv pomocí 11 bitového identifikátoru, čím nižší hodnota identifikátoru, tím má vyšší prioritu. Identifikátor je rozdělen do 4 skupin. Nejvyšší prioritu má první skupina a nejnižší prioritu má 4 skupina. Identifikátor neboli ID připojení je složeno z ID skupiny, ID zpráv a ID zařízení. [14]

První skupina zpráv má rozsah identifikátoru CAN v rozmezí 0x0000 - 0x03FF, takže rozlišuje 1024 zpráv, což nám dává polovinu všech identifikátorů. Identifikátor je složen z 4 bitového ID zprávy a 6 bitového ID zařízení (MAC ID). Při soupeření dvou zařízení o sběrnici vyhrává zařízení s nižším ID zprávy. Pokud však mají dvě zařízení zprávy ze stejné prioritní skupiny, je nahlédnuto na ID zařízení a opět vítězí zařízení s nižší hodnotou. Zprávy první skupiny jsou využívány pro data s vysokou prioritou, například výměna procesních dat. Složení identifikátoru připojení pro první skupinu je na obrázku 1.16.

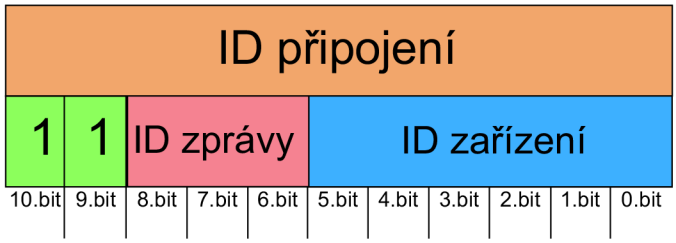


Obr. 1.16: Identifikátor CAN - první skupina.

Druhá skupina zpráv má rozsah identifikátoru CAN v rozmezí 0x0400 - 0x05FF, takže rozlišuje 512 zpráv, tedy čtvrtinu všech identifikátorů. Identifikátor je složen z 6 bitového ID zařízení (MAC ID) a 3 bitového ID zprávy. Když soupeří dvě zařízení o sběrnici, tak mechanizmus funguje obráceně. Prioritní hodnotou je ID zařízení a v rámci potřeby je přihlíženo k ID zprávy. ID zprávy ve větším případě mohou být definovány volitelně a většinou je využita jako předefinovaná sada pro připojení nadřazený a podřazený (master a slave). Integrovaný řadič CAN s 8bitovou maskou je hojně využívaný pro DeviceNet. Z důvodu, aby tento řadič mohl odfiltrovat své vlastní zprávy, je zde upřednostněný identifikátor zařízení. Složení identifikátoru připojení pro druhou skupinu je na obrázku 1.17.

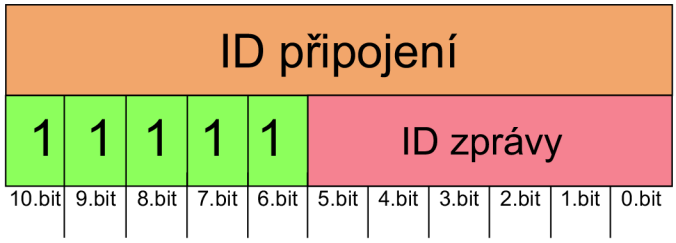
Obr. 1.17: Identifikátor CAN - druhá skupina.

Třetí skupina zpráv má rozsah identifikátoru CAN v rozmezí 0x0600 - 0x07EF, takže rozlišuje 448 zpráv. Princip funkčnosti je stejný jako u první skupiny, jen s tím rozdílem, že třetí skupina je určena pro data s nízkou prioritou a ID zpráv má 3 bitovou hodnotu. Složení identifikátoru připojení pro třetí skupinu je na obrázku 1.18.



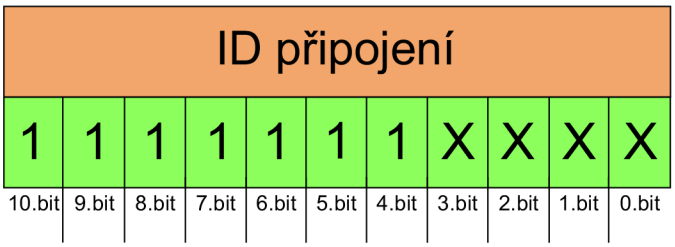
Obr. 1.18: Identifikátor CAN - třetí skupina.

Čtvrtá skupina zpráv má rozsah identifikátoru CAN v rozmezí 0x07C0 - 0x07EF, takže rozlišuje 512 zpráv. Identifikátor je roven 6 bitovému ID zpráv. Slouží pro správu sítě. Složení identifikátoru připojení pro čtvrtou skupinu je na obrázku 1.19.



Obr. 1.19: Identifikátor CAN - čtvrtá skupina.

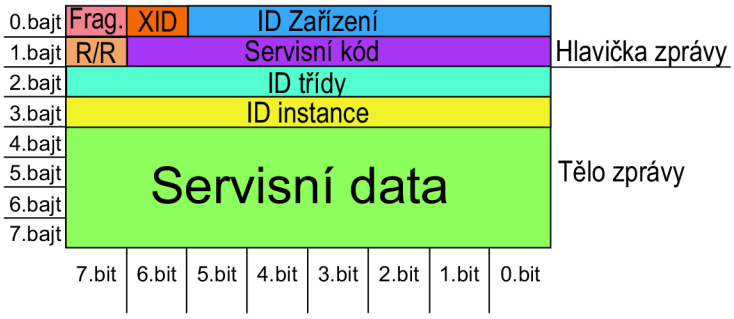
Posledních 16 identifikátorů CAN je neplatných v rozsahu 0x07C0 - 0x07EF. Složení identifikátoru připojení je na obrázku 1.20. [15]



Obr. 1.20: Identifikátor CAN - neplatný rozsah.

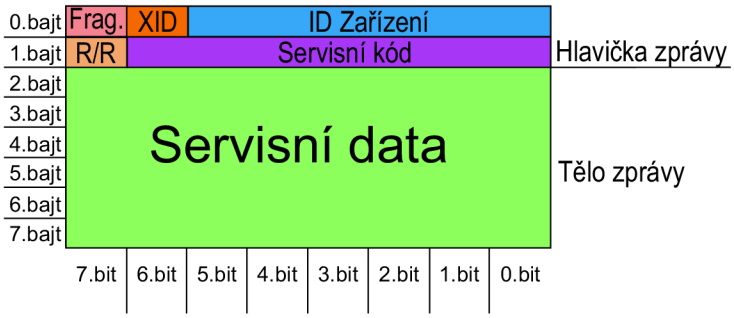
1.3.2 Explicitní zprávy u protokolu DeviceNet

Explicitní zprávy mají vcelku jednoduchý tvar, které se bez potřeby fragmentace vejdou do 8 bajtového paketu. Objekty pro komunikaci pomocí explicitních zpráv musí být nastaveny v zařízení. K aktivaci objektu statického připojení lze využít předdefinované sady připojení nadřízený (master) podřízený (slave) z druhé skupiny zpráv identifikátoru. Pro nastavení dynamických objektů připojení pro explicitní zprávy, je využít port správce nepřipojených zpráv (UCMM). Dynamické objekty připojení slouží k nastavení nebo rušení explicitního spojení pro zasílání zpráv. UCMM může zasílat pouze dva typy požadavků a to jsou požadavky otevřít a zavřít. Při komunikaci pomocí explicitních zpráv musí zprávy procházet objektem směrovač zpráv, který byl popsán výše. Na obrázku 1.21 je znázorněna žádost ve formátu explicitních zpráv.



Obr. 1.21: Žádost DeviceNet - explicitní zprávy.

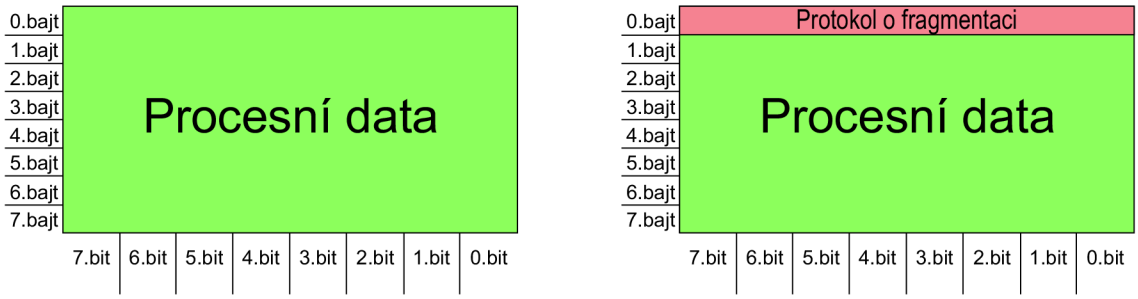
Odpověď na žádost je vyobrazena na obrázku 1.22. Změní se hodnota bitu R/R (žádost/odpověď) z hodnoty 0 na hodnotu 1, přičemž servisní kód zůstává stejný. Navíc nejsou zapotřebí položky identifikátor třídy v druhém bajtu a identifikátor instance v třetím bajtu.



Obr. 1.22: Odpověď DeviceNet - explicitní zprávy.

1.3.3 Implicitní zprávy u protokolu DeviceNet

Implicitní zprávy využívá DeviceNet k výměně procesních a aplikačních dat s vysokou prioritou. Komunikace je založena na modelu producent/konzument. Zprávy jsou přenášeny z jednoho aplikačního objektu producenta do jednoho nebo více aplikačního objektu konzumenta. Datové zprávy jsou o maximální velikosti 8 bitů, proto je v případě větších zpráv potřeba fragmentace. Pokud jsou data fragmentována, tak nultý bajt obsahuje protokol o fragmentaci. Ukázka implicitní zprávy s fragmentací a bez fragmentace je na obrázku 1.23.



Obr. 1.23: DeviceNet - implicitní zpráva s fragmentací a bez fragmentace.

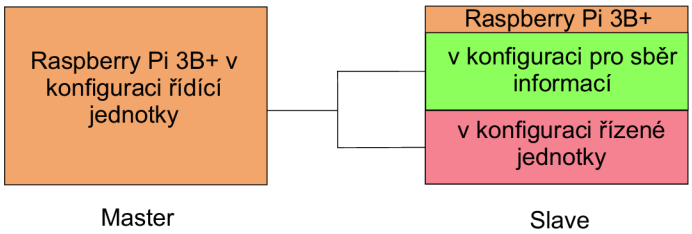
Aby mohla komunikace za pomoci implicitních zpráv fungovat, tak musí být tento typ komunikace nastaveny na zařízeních. Toto nastavení lze provést pomocí předdefinovaného připojení nadřazený (master) a podřazený (slave), které je nastaveno pro aktivaci potřebných statických objektů, jež jsou v zařízení k dispozici nebo za pomoci explicitních zpráv. [22]

2 Návrh simulace průmyslového protokolu a možnosti komunikace

V této kapitole je popsán návrh simulace průmyslového protokolu CIP v provedení Ethernet/IP a možné typy komunikace. Cílem této simulace je se co nejlépe přiblížit situacím v reálném průmyslovém prostředí. Také je navrženo pracoviště, které simuluje část reálné průmyslové infrastruktury.

2.1 Návrh simulace protokolu

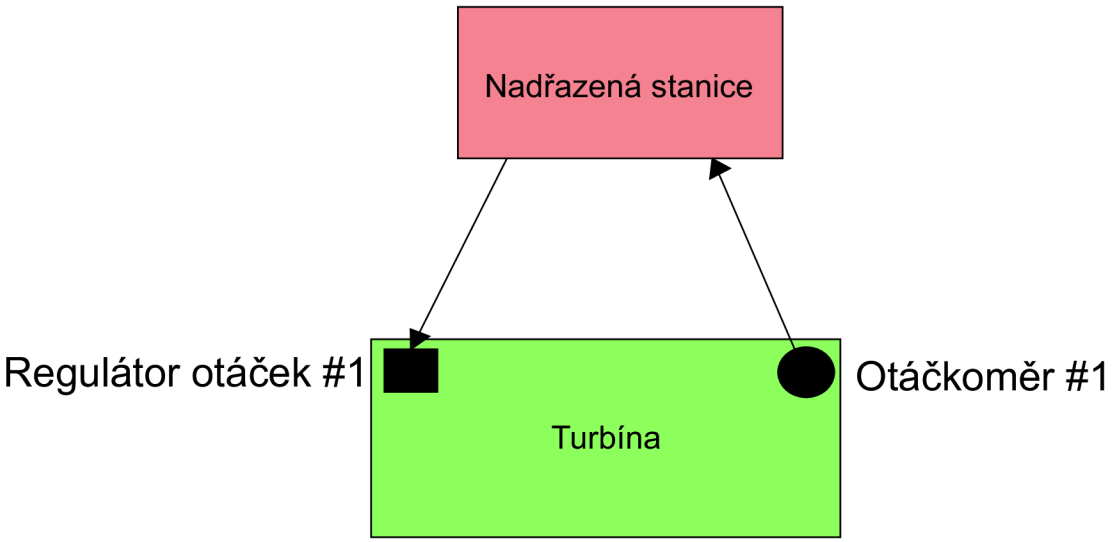
Jak již bylo uvedeno v předešlé kapitole, komunikace v protokolu Ethernet/IP se skládá z řídících (Master) a podřízeních stanic (Slave). Na základě toho byla navržena komunikace pomocí knihovny cpppo [23] mezi dvěma jednodeskovými počítači Rasberry pi 3B+, které simulují zmíněné stanice Master a Slave. Zařízení v reálném provozu jsou řídící jednotka nejčastěji typu PLC a podřízené zařízení se skládá ze dvou druhů komponentů. Prvním komponenta slouží pro sběr informací jako jsou teploměry, otáčkoměry, tlakoměry, hladinometry, průtokoměry, měření vodivosti. Zasílají informace o aktuálním stavu provozu řídící PLC jednotce, která stav provozu vyhodnocuje. Druhou skupinou jsou komponenty, které jsou na základě těchto informací řízeny. To jsou regulační a uzavíratelné armatury, spouštění motoru, regulace otáček a další. Zapojení zařízení pro simulaci je na obrázku 2.1. Simulaci je možné realizovat pomocí n zařízení typu Raspberry Pi nebo jiného zařízení s operačním systémem linux. Zařízení simulují jednotlivé komponenty pro sběr informací, řízené komponenty nebo řídící PLC. Avšak pro naše účely by to bylo nadbytečné, protože Raspberry Pi se chová jako n komponentů na jednom fyzickém zařízení. Součástí simulace jsou dva scénáře, které vycházejí ze skutečného provozu. První scénář demonstruje funkce komunikačního protokolu Ethernet/IP, scénář je blíže popsán v podkapitole 2.2. Druhý scénář je komplexnější a ukazuje funkci celého systému, který je popsán v podkapitole 2.3.



Obr. 2.1: Nahrazení skutečných typů zařízení pomocí Raspberry Pi.

2.2 Scénář regulace turbíny

Regulace otáček turbíny je klasický scénář z energetického průmyslu a je vidět na obrázku 2.2. Turbína slouží k převodu tepelné energie na mechanickou energii. Vysokotlaký díl turbíny se otáčí 8000 krát za minutu. V scénáři je hodnota z otáčkoměru #1, který slouží jako stanice pro sběr informací. Počet otáček za minutu je sdělen nadřazené stanici, která stav turbíny vyhodnotí a odešle řídící instrukce regulátoru otáček #1, který zastává funkci řízené stanice.



Obr. 2.2: Scénář regulace turbíny.

2.3 Scénář zpracování odpadní vody

Z důvodu, aby byla funkce a zaměření komunikačního protokolu CIP v provedení Ethernet/IP dobře prezentována a každý mohl správně pochopit jeho funkčnost, je vytvořen scénář, který prezentuje přečerpání sběrné nádrže odpadních vod na jaderné elektrárně. Nyní pomocí obrázku 2.3 je vysvětlena funkce celého scénáře.

Sběrná nádrž odpadních vod slouží k shromažďování všech vod z celého výrobního bloku a jsou do ní také vyvedeny drenáže všech místností bloku. Na nádrži jsou dva ukazatele měření hladiny. Ukazatel měření hladiny #1 slouží k indikaci, že je nádrž plná. Pokud hladina v nádrži dosáhne hodnoty 900 cm otevře se uzavírací armatura #1, regulační armatura #1 se otevře na 20 % a vypínač motoru se sepne, aby mohlo začít přečerpávat. Ukazatel měření hladiny #2 má opačnou funkci, než ukazatel měření hladiny #1. Pokud hladina klesne na hodnotu 40 cm, je vydána instrukce, aby se uzavírací armatura #1 uzavřela a vypínač motoru vypne pohon

čerpadla. Tím je přečerpávání ukončeno. I když by absence kapaliny čerpadlu nijak neuškodila, přesto je tu ukazatel měření hladiny #2 z důvodu, že v odpadních vodách se nachází i pevné části, které na dně nádrže sedimentují a vytváří nános kalu, který by ucpal sání čerpadla a to by nebylo provozuschopné.

Obdobnou funkci mají ukazatelé měření hladin nádrží na zpracování odpadních vod. Pokud v nádrži na zpracování odpadních vod #1 dosáhne hodnoty 10000 cm, uzavře se uzavírací armatura #2 a následně řídící jednotka zkontroluje ukazatel hladiny #4. Pokud neindikuje, že hladina v nádrži dosahuje 1000 cm, tak je dán pokyn k otevření uzavírací armatury #3. Když by byla nádrž zpracování odpadních vod #2 plná (tedy ukazatel měření hladiny indikuje hodnotu 1000 cm), pak pokračuje kontrolou ukazatele měření hladiny #5. Pokud neupozorňuje na to, že je nádrž na zpracování odpadních vod #3 plná, tak otevře uzavírací armaturu #4 a nádrž je naplněna. V případě, že všechny tři ukazatelé měření hladin u nádrží zpracování odpadních vod vykazují hodnotu 1000 cm, uzavře se uzavírací armatura #1, vypínač motoru vypne motor čerpadla a vypíše chybovou hlášku.

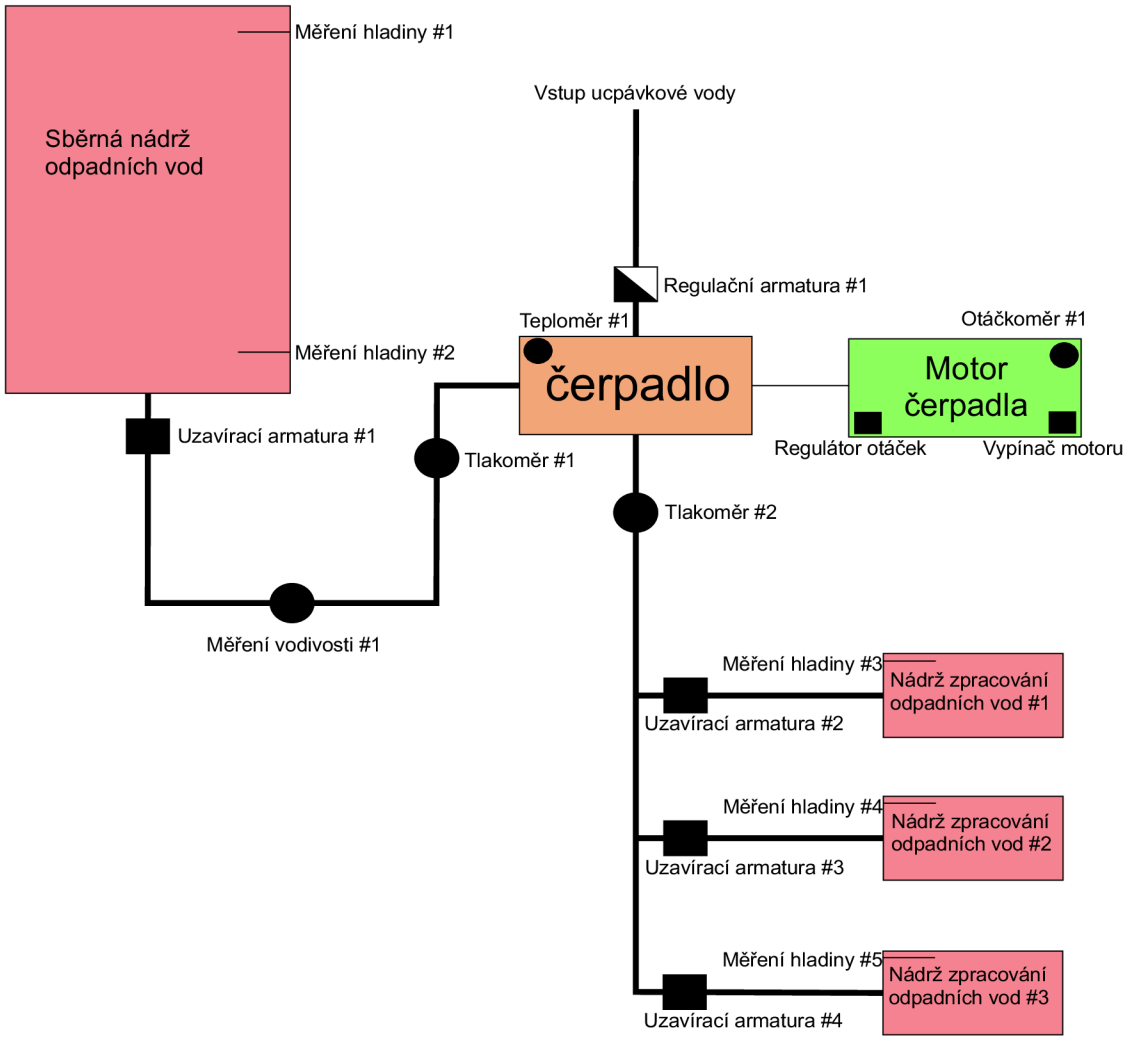
Dalším zdrojem provozních informací je měření vodivosti #1, které kontinuálně měří vodivost, protože tímto způsobem se zjišťuje míra radioaktivity v daném médiu. Když hodnota vodivosti překročí hodnotu 330 μ sievertů na centimetr čtvereční, tak je uzavřena uzavírací armatura #1 a vypínač motoru vypne motor čerpadla.

V případě, že je vypínač motoru zapnutý, tím pádem čerpadlo jede, vypočítá se součet tlaku mezi sáním a výtlakem čerpadla. Z důvodu, že by rozdíl tlaku převyšoval tlak, který přidává čerpadlo, tak musí být přivřena regulační armatura #1 na vstupu ucpávkové vody, protože to je jedna ze dvou příčin zvýšeného tlaku. Druhou možností je selhání hlášení, že všechny nádrže na zpracování odpadních vod jsou plné, tato možnost je v simulaci nepravděpodobná, ale v reálném provozu k ní může dojít. Čerpadlo přidává tlak na výtlaku 0,58 MPa a předřadné potrubí má tlak 0,4 MPa, proto je maximální hodnota tlaku z tlakoměru #1 nastavena jako hodnota tlaku na tlakoměru #2 + 0,58 MPa. Pokud je tato hodnota překročena, tak je uzavřena uzavírací armatura #1 a odstaven motor čerpadla pomocí vypínače motoru.

Další důležité informace jsou z teploměru #1. Teplota v čerpadle nesmí přesáhnout hodnotu 80 stupňů celsia. V nominálním provozu se teplota v čerpadle udržuje na 60 stupních celsia. Pokud teplota přesáhne daných 60 °C, tak se regulační armatura #1 pootevře o 10 %, protože ucpávková voda má dvě využití. Její primární účel je, aby z čerpadla neunikal přečerpávaná kapalina a sekundární účelem je chlazení rotační části čerpadla, kde třením vzniká teplo.

Posledním parametrem provozu je sledování informací z otáčkoměru #1, který poskytuje informace o tom, kolik otáček vykoná motor za jednu vteřinu. Rychlost otáček by měla být konstantní, avšak může dojít k chybě.Tím pádem by čerpadlo dávalo mnohem vyšší tlak než by mělo a mohlo by dojít k zničení potrubí nebo

dalších technologií. Proto je zde regulátor otáček, který otáčky udržuje na nominální hodnotě a to je 2950 otáček za minutu.



Obr. 2.3: Scénář zpracování odpadní vody.

2.4 Hodnota zpoždění

Hodnota zpoždění je nejdůležitější sledovaný parametr u průmyslových protokolů obecně. Čím je zpoždění větší, tím je přenášená hodnota pomocí průmyslového protokolu méně aktuální. Bohužel v žádném z odborných zdrojů není tato hodnota definována, ani společnost ODVA, Inc. jako globální asociace sdružující přední poskytovatele automatizačních protokolů neposkytuje definované hodnoty maximálního zpoždění pro různé automatizační aplikace. Autor této práce kontaktoval zástupce

společnosti Rockwell Automation s.r.o., která je poskytovatel automatizačních protokolů CIP v České republice, aby mohla být hodnota zpoždění v simulaci průkazná. Avšak hodnota zpoždění je u každé implementace originální a nedá se standardizovat. Snaha je vždy o dosažení co nejnižší hodnoty zpoždění. Stejného názoru jsou i zástupci společnosti Siemens, s.r.o., která zaštiťuje v České republice obdobný konkurenční automatizační protokol Profinet. Zpoždění dělíme na dva typy.

První typ zpoždění je způsoben komunikační soustavou. Což znamená, že je zpoždění tvořeno fyzikálními vlastnostmi přenosových médií a zpožděním vzniklým při průchodu přes aktivní prvky sítě jako jsou směrovače nebo přepínače. Výraznější zpoždění obecně vzniká na aktivních prvcích.

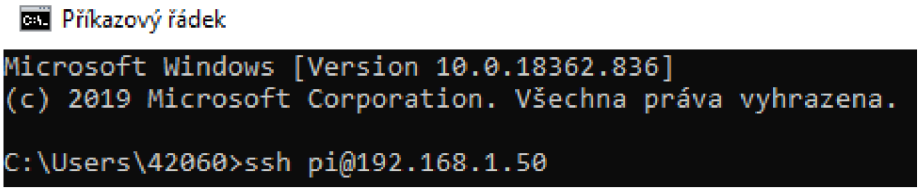
Druhý typ zpoždění je způsoben výpočetním výkonem zařízení, na kterých je simulace spuštěna. Toto zpoždění je úměrné množství prováděných operací a je možné toto zpoždění snížit za pomoci separátně prováděných operací. Vzhledem k výše uvedeným skutečnostem je hodnota zpoždění v této simulaci nejnižší možná.

2.5 Konfigurace zařízení

V této kapitole je popsána konfigurace jednotlivých zařízení. Na obrázku 2.1 můžeme vidět, reálné zapojení pracoviště pro simulaci. Jsou zde dva již dříve zmíněné jednočipové počítače Raspberry Pi 3B+, první v konfiguraci řídící jednotky, druhý v konfiguracích pro sběr informací a pro řízené jednotky. Komunikace probíhá přes ethernetové rozhraní. Aby bylo možné sledovat síťový provoz mezi Raspberry Pi, tak je nutné použít směrovač, pomocí kterého je do sítě připojen uživatelský počítač. Na uživatelském počítači je možno sledovat výsledky a chování simulace, popřípadě lze konfigurovat parametry sledované simulace. Veškerá komunikace je řešena v rámci místní sítě.

2.5.1 Připojení pomocí uživatelského počítače

Pro monitorování průběhu a výsledků je do simulace připojen uživatelský počítač, na kterém je možné sledovat stav simulace obou zařízení Raspberry Pi. Připojení k Raspberry Pi je přes internetové rozhraní a je využit protokol SSH. Nejjednodušší způsob připojení je využití příkazového řádku (Command Prompt). Po spuštění příkazového řádku stačí napsat příkaz *ssh pi@IP adresa daného zařízení*, jak je vidět na obrázku 2.4. Následně jsme vyzváni k napsání hesla, po jeho ověření už jsme vzdáleně připojeni k Raspberry Pi a můžeme s ním dále pracovat.



Obr. 2.4: Připojení k Raspberry Pi pomocí SSH.

2.5.2 Konfigurace Raspberry Pi

Nyní je zde popsaná společná konfigurace pro obě zařízení Raspberry Pi 3B+, ve kterých je implementován operační systém Raspbian verze 9. Pro spojení přes SSH je tu nutné v konfiguračním souboru na Raspberry Pi povolit komunikaci. Postup je následující: do terminálu je napsán příkaz sudo raspi-config, následně se otevře nabídka, ve které se uplatní následující postup:

$$Interfacing\ options \Rightarrow SSH \Rightarrow Enable.$$

Dalším důležitým krokem je nastavení logických adres (IP adres), aby mohla zařízení mezi sebou komunikovat v rámci jedné sítě. Logické adresy jsou nastaveny staticky, protože není důvod pro jiné řešení. V běžném provozu jsou také komponentám přiřazeny statické adresy z důvodu, aby nedošlo k záměně jednotlivých prvků a přiřazování adres dynamicky by také zvyšovalo zátěž sítě. Postup pro nastavení statické adresy je následující. Otevřeme soubor „sudo nano /etc/dhcpcd.conf“, Do tohoto souboru dopíšeme na jeho konec dva řádky:

- „interface eth0“,
- „static ip_address=192.168.1.X/30“,

Písmeno X je nahrazeno konkrétní hodnotou logické adresy pro dané Raspberry Pi. Konkrétní hodnoty logických adres jsou:

- Raspberry Pi server - řídící stanice IP adresa 192.168.1.50,
- Raspberry Pi klient - podřízená stanice IP adresa 192.168.1.51,
- maska sítě je nastavena na hodnotu 255.255.255.0.

Heslo je pro obě zařízení nastaveno stejně, protože nevidím důvod proč by měla být hesla různá. Znění hesla je „generatoriec61850“. Toto heslo již bylo přednastaveno a považuji jej za bezpečné, proto jsem jej neměnil. Pro simulaci je použita knihovna cpppo, která využívá programovacího jazyku Python, tento jazyk je v poslední době hodně populární a je uživatelsky příjemný. Ovšem musí být programovací jazyk python implementován do Raspberry Pi. Toho docílíme použití následujících příkazů:

- „sudo apt-get update“,
- „sudo apt-get install python3“,
- „sudo apt-get install python3-pip“.

Ještě zbývá implementace knihovny cpppo, Tuto implementaci uskutečníme vložením příkazu „sudo pip3 install cpppo“. Důležité při implementaci modulů je, aby byl zpřístupněn internet pro zařízení Raspberry Pi. Jinak je logické, že zařízení nemá odkud tyto knihovny stáhnout.

2.5.3 Konfigurace pro jednosměrnou komunikaci

V rámci testování knihovny cpppo, byla nejdříve realizována jednoduchá komunikace mezi Raspberry Pi typu producent-konzument, což znamená, že podřízená jednotka Raspberry Pi (pi@192.168.1.51) produkuje zprávy, které zasílá řídící jednotce Raspberry Pi (pi@192.168.1.50) ta zprávy konzumuje a vypisuje.

Pro spuštění řídící jednotky Raspberry Pi stačí zadat příkaz:

```
1 python3 -m cpppo.server.enip -v
```

Pro zasílání zpráv od řízené jednotky Raspberry Pi je nutno zadat příkaz s tímto textem zprávy:

```
1 python3 -m cpppo.server.enip.client -v --print SCADA[1]=99 SCADA[0-10]
2 'TEXT[1]=(SSTRING)"Zkušební zpráva"' TEXT[0-3] -a 192.168.1.50
```

Zpráva byla zachycena programem Wireshark. Z tohoto programu bylo patrné, že se jedná o zprávu protokolu CIP v kofiguraci Ethernet/IP, protože zachycené pakety mají správný tvar, proto nadále autor této práce použil knihovnu cpppo. Výstup z programu Wireshark je na obrázku 2.5. Důležité je v obou těchto příkazech uvést číslovku 3 za slovem python, protože u novější verze operačního systému Raspbian stačí napsat pouze python a Rapsbian už automaticky pracuje s verzí python3. Starší verze operačního systému Raspbian touto funkcí nedisponuje a vypisuje se chybová hláška nedefinovaného objektu, proto může uživatel zvyklí na modernější operační systém Rapsbian strávit mnoho času řešením tohoto triviálního problému, což byl i můj případ. Také je důležité, aby na zařízeních Raspberry Pi nebyla instalována vyšší verze skriptovacího jazyka python než 3.6.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.50	192.168.1.49	CIP	118	Success: Multiple Service Packet: Service (0x53)
2	0.000029345	192.168.1.49	192.168.1.50	TCP	66	54372 → 44818 [ACK] Seq=1 Ack=53 Win=502 Len=0 TSval=2270711973 TSecr=2063881286
3	0.006247622	192.168.1.49	192.168.1.50	CIP CM	142	Unconnected Send: Multiple Service Packet: Service (0x4c)
4	0.053283076	192.168.1.50	192.168.1.49	TCP	66	44818 → 54372 [ACK] Seq=53 Ack=77 Win=227 Len=0 TSval=2063881339 TSecr=2270711979
5	0.081259664	192.168.1.50	192.168.1.49	CIP	124	Success: Multiple Service Packet: Service (0x4c)
6	0.081282854	192.168.1.49	192.168.1.50	TCP	66	54372 → 44818 [ACK] Seq=77 Ack=111 Win=502 Len=0 TSval=2270712054 TSecr=2063881367
7	0.142725835	192.168.1.51	192.168.1.49	SSH	150	Server: Encrypted packet (len=84)
8	0.142761610	192.168.1.49	192.168.1.51	TCP	66	34356 → 22 [ACK] Seq=1 Ack=85 Win=586 Len=0 TSval=479682849 TSecr=93979982
9	0.163955493	192.168.1.50	192.168.1.49	CIP	132	Success: Multiple Service Packet: Service (0x4c)
10	0.163978888	192.168.1.49	192.168.1.50	TCP	66	54372 → 44818 [ACK] Seq=77 Ack=177 Win=502 Len=0 TSval=2270712137 TSecr=2063881450

Obr. 2.5: Výpis z programu Wireshark.

2.5.4 Konfigurace pro obousměrnou komunikaci za pomoci vyčítání z konzole

Jak bylo znázorněno na obrázku 2.1, je zde popsaná konfigurace, která je použita pro obousměrnou komunikaci za pomocí vyčítání z konzole. Tento model pracuje, tak, že na straně podřízené stanice pro sběr dat jsou generovány hodnoty, které jsou zasílány na řídící stanici. V řídící stanici jsou tato data vypsána do konzole a z této konzole je řídící stanice vyčítá a zpracovává. Na základě výsledků zpracování odesílá řídící instrukce podřízené stanici. Simulované chování znázorňuje scénář regulace otáček motoru na čerpadle.

Konfigurace stanice pro sběr dat

Nyní jsou zde popsány funkce, které se nachází ve skriptu „io_client.py“ psaném v jazyce python, který se nachází na zařízení Raspberry Pi s logickou adresou 192.168.1.51.

Nejdříve jsou nastaveny základní parametry komunikace jak je vidět na výpisu 2.1. Zprávy jsou zasílány na zařízení s logickou adresou 192.168.1.50 tedy na nadřízenou stanici. Port je nastaven na základní hodnotu 44818. Hodnota hloubky je nastavena na jednu transakci. Je povoleno maximálně 20 servisních paketů. Fragmentace je zakázána. Pokud je doba doručení zpráva vyšší než 1 vteřina je zpráva považována za ztracenou. Data se tisknou do konzole.

Výpis 2.1: Parametry komunikace.

```
1 if __name__ == "__main__":
2     logging.basicConfig( **cpppo.log_cfg )
3
4     host                        = '192.168.1.50'
5     port                        = address[1]
6     depth                       = 1
7     multiple                    = 20
```

```
8       fragment                    = False
9       timeout                     = 1.0
10      printing                    = True
```

Následně je vložen blok, jehož úkolem je generovat náhodné hodnoty, aby bylo možné sledovat reakce nadřízené stanice na náhlé změny v provozu (jak je vidět na výpisu 2.2). Generované hodnoty jsou v rozmezí od 500 do 1500, protože maximální hodnota v řídící stanici je nastavena na 1000, takže můžeme sledovat časté překročení této hranice a následnou reakci. Data jsou zasílána do scriptu „pollserver“ na nadřazené stanici v nekonečné smyčce, tak by tomu bylo i v reálném provozu.

Výpis 2.2: Generátor náhodných hodnot.

```
1  while True:
2   i = random.randrange(500,1500)
3   with client.connector( host=host, port=port, timeout=timeout ) as
    ↪  connection:
4          tags = ["Tag[0-9]+16=(DINT){}".format(i), "@0x2/1/1", "Tag[3-5]"]
5          operations = client.parse_operations( tags )
6          failures,transactions       = connection.process(
7          operations=operations, depth=depth, multiple=multiple,
8          fragment=fragment, printing=printing, timeout=timeout )
9
10 sys.exit( 1 if failures else 0 )
```

Výpis na konzoly ze zařízení pro sběr informací je na výpisu 2.3. Celý skript io_client.py je k nahlédnutí v příloze B.1.

Výpis 2.3: Výpis na konzoly pro sběr informací.

```
1  b'Tag[    4-4    ] <=[693]'
2  b'Tag[    4-4    ] <=[1029]'
3  b'Tag[    4-4    ] <=[1208]'
```

Konfigurace stanice pro zpracování dat

Nadřazená stanice pro zpracování dat je taktéž zařízení Raspberry Pi, které má logickou adresu 192.168.1.50. V jeho adresáři se nachází script „pollserver.py“, který má za úkol výčet dat z konzole a jejich zpracování.

Parametry komunikace jsou nastaveny obdobný způsobem, jen s rozdílem ve třech položkách. Logická adresa cíle je 192.168.1.51, nastavena na podřízené zařízení

Raspberry Pi, nastavený port je 8090 a je zrušený tisk na konzoly. Výpis můžete vidět na 2.4.

Výpis 2.4: Parametry komunikace.

```
logging.basicConfig( **cpppo.log_cfg )

host                        = '192.168.1.51'
port                        = 8090
depth                       = 1
multiple                    = 20
fragment                    = False
timeout                     = 1.0
printing                    = False
```

Tato funkce potvrzuje přijetí dat od podřízeného zařízení pro sběr informací 2.5.

Výpis 2.5: Potvrzení přijetí zprávy.

```
def sendResponse(data):
 sys.stdout = open(os.devnull, "w")
 with client.connector( host=host, port=port, timeout=timeout ) as
  ↪  connection:
     tags = ["Tag[0-9]+16=(DINT){}".format(data), "@0x2/1/1", "Tag[3-5]"]
     operations = client.parse_operations( tags )
     failures,transactions       = connection.process(
     operations=operations, depth=depth, multiple=multiple,
     fragment=fragment, printing=printing, timeout=timeout )
 sys.stdout = sys.__stdout__
```

Blok pro zpracování dat, přijatá data vypisuje z „stdout“ do konzole, následně jsou data připravená k analýze. Poté jsou vyčtena data a je proveden výpočet pro regulaci, který je odeslán řízené stanici. Výpis 2.6 k nahlédnutí.

Výpis 2.6: Zpracování dat.

```
while True:
  output = process.stdout.readline()
  if process.poll() is not None:
    break
  if output:
    print(output.strip())
```

```
7       data = str(output.strip()).split("<=")
8       result = int(json.loads(data[1][0:-1])[0])
9       sendResponse(1000-result)
10
11  rc = process.poll()
```

Informace o přijetí zpráv od zařízení pro sběr informací je na výpisu 2.7. Celý skript „ pollserver.py“ je k nahlédnutí v příloze B.2.

Výpis 2.7: Informace o přijetí zprávy.

```
1       Tag[0][    0--7   ]+ 16 <= [693]:  'OK'
2   @0x0002/1/1                    == [0]:  'OK'
3       Tag[3][    3-5    ]+  0 <= [0, 693, 0]:  'OK'
4       Tag[0][    0--7   ]+ 16 <= [1029]:  'OK'
5   @0x0002/1/1                    == [0]:  'OK'
6       Tag[3][    3-5    ]+  0 <= [0, 1029, 0]:  'OK'
7       Tag[0][    0--7   ]+ 16 <= [1208]:  'OK'
8   @0x0002/1/1                    == [0]:  'OK'
9       Tag[3][    3-5    ]+  0 <= [0, 1208, 0]:  'OK'
```

Konfigurace pro řízenou stanici

Tato konfigurace je nastavena na podřízeném zařízení, které má být řízeno na logické adrese 192.168.1.51. Úkolem této konfigurace je přijmout řídící instrukce a vypsat je na konzoly. Je nutno jej spustit za pomoci modulu naslouchání „START.sh“. Naslouchá na portu 8090 a může přijímat zprávy od zařízení s libovolnou logickou adresou. Obsah „START.sh“ je následující:

```
1   python3 -m cpppo.server.enip -a 0.0.0.0:8090 Tag=DINT[10]
```

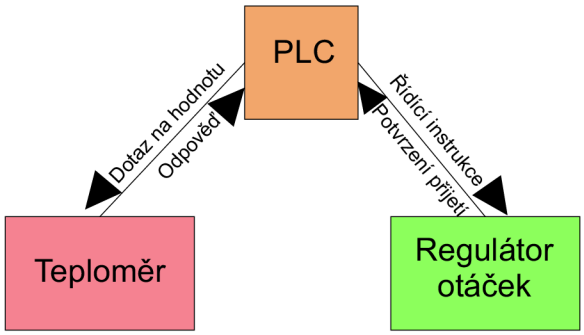
Výpis řídící instrukce na konzoly je k nahlédnutí na výpisu 2.8, kde je vidět o kolik otáček má regulátor přidat nebo ubrat.

Výpis 2.8: Výpis na konzoly řízeného zařízení.

```
1   Tag[    4-4    ] <=[307]
2   Tag[    4-4    ] <=[-29]
3   Tag[    4-4    ] <=[-208]
```

2.5.5 Základní konfigurace pro obousměrnou komunikaci v reálném čase

Kvůli faktu, že obousměrná komunikace za pomoci vyčítání z konzole má vysoké zpoždění, které se pohybovalo kolem 4 vteřin, tak tuto metodu nelze použít u zařízení pracující v reálném čase. Nyní je zde popsána konfigurace pro komunikaci se zpožděním v rámci milisekund. Schéma komunikace je na obrázku 2.6.



Obr. 2.6: Schéma komunikace v reálném čase.

Konfigurace podřízených stanic

Obě podřízené stanice jsou na jednom zařízení Raspberry Pi, jak je vidět na obrázku 2.1. Nejdříve musí být spuštěno zařízení pro sběr informací, na kterém musí být napsán následující příkaz, který komunikuje za pomoci portu 44818, jak je vidět níže.

```
1 python3 -m cpppo.server.enip -v -a :44818 scada=DINT[1000]
```

Výpis z konzole na zařízení pro sběr informací po odeslání zprávy je v příloze C.2. Řízené zařízení se spouští obdobným způsobem, jako zařízení pro sběr informací. Spustí se naslouchání na portu 44819 (můžeme vidět pod tímto odstavcem). Naslouchání na řízeném zařízení musí být spuštěno, jako druhé jinak by přeposlaná zpráva nedorazila, ke svému cíli.

```
1 python3 -m cpppo.server.enip -v -a :44819 a=DINT[1000]
```

Výpis z konzole na řízeném zařízení je v příloze C.3.

Konfigurace nadřazené stanice

Na nadřazené stanici musí být spuštěn skript „controller.py“ za pomoci příkazu:

```
python3 controller.py
```

Ve scriptu jsou definovány logické adresy a porty, které jsou připojeny k daným zařízením, jak je vidět ve výpisu kódu 2.9.

Výpis 2.9: Definice logických adres a portů.

```
hostTemp  = ("192.168.1.50", 44818)
hostMotor = ("192.168.1.50", 44819)
tags = [ "scada[0-10]", "scada[1]=99", "scada[0-10]" ]
```

Dále je ve scriptu funkce 2.10 pro komunikaci se zařízením pro sběr informací. Nadřazená stanice se dotazuje podřazené stanice na hodnotu teploty a ta mu zasílá hodnotu jako odpověď. Také je zde funkce pro výpis času odeslání dotazu a výpis času přijetí odpovědi, aby bylo možno sledovat zpoždění.

Výpis 2.10: Dotazování se podřízené stanice.

```
with client.connector( host=hostTemp[0], port=hostTemp[1] ) as conn:
    t1 = datetime.datetime.utcnow()
    req1 = conn.write( "scada[1-3]", data=[111,222,333] )
    req2 = conn.read( "scada[2]" )
    t2 = datetime.datetime.utcnow()
    pprint(req2)
    print("---------->",t1,"\n---------->",t2)
    assert conn.readable( timeout=1.0 ), "Failed to receive reply 1"
    rpy1 = next( conn )
    assert conn.readable( timeout=1.0 ), "Failed to receive reply 2"
    rpy2 = next( conn )
```

Na následujícím výpisu 2.11 z konzole je vidět nastavení komunikace, čas a datum odeslání dotazu a přijetí odpovědi. Pomocí těchto údajů víme, že zpoždění dosáhlo hodnoty 0.05119 vteřin.

Výpis 2.11: Na konzoly pro komunikaci se zařízením pro sběr dat.

```
{'input': bytearray(b'R\x05\x91\x05scada\x00(\x02\x01\x00\x00\x00\x00\x00'),
 'path.segment[0].symbolic': 'scada',
 'path.segment[1].element': 2,
 'read_frag.elements': 1,
 'read_frag.offset': 0,
 'service': 82}
----------> 2019-10-23 18:18:04.809417
----------> 2019-10-23 18:18:04.814536
```

Funkce 2.12 pro odeslání řídící instrukce podřízenému zařízení je velmi obdobná, jako funkce dotazující se na hodnotu teploty. Jediný rozdíl je v tom, že se nadřazená stanice nedotazuje, ale posílá řízenému zařízení hodnotu a čeká na odpověď.

Výpis 2.12: Dotazování se podřízené stanice.

```
with client.connector( host=hostMotor[0], port=hostMotor[1] ) as conn:
    t1 = datetime.datetime.utcnow()
    req1 = conn.write( "scada[1-3]", data=[111,222,333] )
    req2 = conn.read( "scada[2]" )
    t2 = datetime.datetime.utcnow()
    pprint(req2)
    print("---------->",t1,"\n---------->",t2)
    assert conn.readable( timeout=1.0 ), "Failed to receive reply 1"
    rpy1 = next( conn )
    assert conn.readable( timeout=1.0 ), "Failed to receive reply 2"
    rpy2 = next( conn )
```

Nyní zde je výpis 2.13 z konzole pro odeslání řídící instrukce. Z časových údajů je vidět, že hodnota zpoždění je 0.05613 vteřin a celá komunikace od zaslání dotazu na hodnotu teploty až po potvrzení přijetí řídící instrukce trvá 0,426672 vteřin. Celý script „controller.py“ je k nahlédnutí v příloze C.1.

Výpis 2.13: Na konzoly pro komunikaci s řízeným zařízením.

```
{'input': bytearray(b'R\x05\x91\x05scada\x00(\x02\x01\x00\x00\x00\x00\x00'),
 'path.segment[0].symbolic': 'scada',
 'path.segment[1].element': 2,
 'read_frag.elements': 1,
```

```
5   'read_frag.offset': 0,
6   'service': 82}
7  ----------> 2019-10-23 18:18:05.230476
8  ----------> 2019-10-23 18:18:05.236089
```

3 Realizace simulace průmyslového protokolu

Aby bylo možné zprovoznit komunikace na vyšší úrovni, jako je spouštění scénářů, implementace vlastních scénářů a plynulé funkčnosti simulace, je potřeba využít více navzájem provázaných knihoven a skriptů pro zařízení Raspberry Pi, kterou jsou psány ve skriptovacím jazyce Python. Dále jsou popsány jednotlivé prvky knihoven a skriptů, které byly vytvořeny. Popis je chronologický, jak simulace postupuje mezi vytvořenými knihovnami a skripty. Veškerá zařízení pro simulaci musí obsahovat knihovnu „rpscada“, která byla vytvořena autorem této práce. Nadřazená stanice využívá skript „master“ a podřízené stanice skript „slave“. Veškeré skripty a knihovny použité pro simulaci jsou v příloze F.

3.1 Implementace knihoven

Nejdříve musí být spuštěna skript s názvem „slave.py“ na podřízeném zařízení Raspberry Pi, které představuje oba typy podřízených stanic. Jsou implementovány knihovny zobrazené na výpisu z kódu 3.1.

Výpis 3.1: Importované knihovny do slave.py

```
1
2  import rpscada
3  import cpppo
4  from mollyab import RPI_MACHINE
5  from threading import Thread
6  import time
7  import os
8  from cpppo.server.enip import client
9  from pprint import pprint
10 import datetime
11 from pprint import pprint
12 import sys
```

Jako druhý je spuštěn skript „master.py“, který implementuje potřebné knihovny zobrazené na výpisu z kódu 3.2. Důležitá je zde společná knihovna s názvem „rpscada“, kterou taktéž vytvořil autor této práce na výpisu 3.3 jsou knihovny implementovány do této knihovny.

Výpis 3.2: Importované knihovny do master.py

```
import rpscada
import sys
import importlib
from threading import Thread
import time
```

Výpis 3.3: Importované knihovny do rpscada.py

```
import json
import os
import socket
import locale
from dialog import Dialog
import time
from datetime import datetime
from datetime import timedelta
from threading import Thread
from cpppo.server.enip import client
from time import sleep
import logging
import sys
import threading
import cpppo

from cpppo.server.enip import poll
from mollyab import RPI_MACHINE as device
```

3.2 Simulované hodnoty

Seznam hodnot, se kterými tato simulace pracuje, je vidět na výpisu z kódu 3.4. Simulované hodnoty jsou definovány názvem, typem proměnné a velikostí. V simulaci jsou využity jen tři typy proměnných. Prvním typem je proměnná „BOOL“, která může nabýt pouze dvou hodnot a to logické jedničky nebo logické nuly. Proměnná „BOOL“ je využita jako náhrada zařízení pro sběr informací, které pouze indikují změnu stavu. Například pokud hladina dosáhne hodnoty 2 metry, sepne se čidlo, které indikuje logickou jedničku místo logické nuly. Když hladina klesne

pod 2 metry, čidlo vysílá opět logickou nulu. Druhým typem je proměnná „DINT“, která představuje datový typ Integer pracující s celými čísly. Tento datový typ je využit u zařízeních, které představují konkrétní hodnoty. Třetím datovým typem je „REAL“, který pracuje s desetinnými čísly. Datový typ „REAL“ je použit v simulovaných hodnotách, u kterých je potřeba sledovat i minimální změny, jako například měření tlaku. Tento seznam hodnot je umístěn ve skriptu „slave.py“ a knihovně „rpscada“. Ve skriptu „slave.py“ jsou simulované hodnoty doplněné o výchozí simulované hodnoty.

Výpis 3.4: Simulované parametry skript slave.py

```
parametters={
    "turbina1"          : {"type":"BOOL","size":"10","default":"false"},
    "otackomer2"        : {"type":"DINT","size":"1000","default":"0"},
    "tlak2"             : {"type":"REAL","size":"1000","default":"0"},
    "tlak1"             : {"type":"REAL","size":"1000","default":"0.4"},
    "mereni_vodivosti1" : {"type":"DINT","size":"1000","default":"330"},
    "u_armatura4"       : {"type":"BOOL","size":"10","default":"false"},
    "u_armatura3"       : {"type":"BOOL","size":"10","default":"false"},
    "u_armatura2"       : {"type":"BOOL","size":"10","default":"false"},
    "hladina5"          : {"type":"BOOL","size":"10","default":"false"},
    "u_armatura1"       : {"type":"BOOL","size":"10","default":"false"},
    "hladina4"          : {"type":"BOOL","size":"10","default":"false"},
    "hladina2"          : {"type":"BOOL","size":"10","default":"true"},
    "hladina1"          : {"type":"BOOL","size":"10","default":"false"},
    "hladina3"          : {"type":"BOOL","size":"10","default":"false"},
    "r_armatura1"       : {"type":"DINT","size":"1000","default":"0"},
    "cerpadlo1"         : {"type":"BOOL","size":"10","default":"false"},
    "teplota1"          : {"type":"DINT","size":"1000","default":"55"},
    "otackomer1"        : {"type":"DINT","size":"1000","default":"2950"}
}
```

3.3 Spuštění programu v nadřazené stanici

Na výpisu z kódu 3.5 vidíme zdrojový kód pro následující funkce z skriptu „master“. Nejdříve je spuštěn celý program. Ve spouštěcím skriptu „master“ je vytvořen objekt typu „master“ a jeho definice je uložena v knihovně „rpscada“. Následně je vytvořeno základní grafické rozhraní, které je taktéž definované v knihovně „rpscada“ a následně je volána funkce „mainMenu“, která zobrazí uživateli hlavní menu této aplikace a je popsána v podkapitole 3.3.

Výpis 3.5: Spuštění programu v nadřazené stanici.

```
if __name__ == "__main__":
    master = rpscada.Master()
    gui = rpscada.Gui("CIP simulace")
    mainMenu()
```

3.4 Funkce hlavního menu

Pomocí funkce „mainMenu“ je díky knihovně „pythondialogs“ zobrazeno grafické rozhraní, která slouží pro grafické znázornění celé simulace v knihovně „rpscada“. Toto rozhraní bylo vytvořeno z důvodu pohodlnější manipulace s celou simulací. Na výpisu kódu 3.6 můžeme vidět tuto definici, která zobrazuje možnosti výběru. Ve výběru vidíme dvojici parametrů ve formátu „(číslice) název možnosti“. Když uživatel vybere jednu z možností, tak se číselná hodnota s název možnosti uloží do proměnné „tag“ a vybrané tlačítko „OK“ nebo „CANCEL“ do proměnné „code“. V případě, že zvolíme možnost „OK“, tak nám celá funkce vrací vybranou hodnotu „code“. Pokud vybereme možnost „CANCEL“, tak nám to vrátí prázdnou hodnotu.

Výpis 3.6: Funkce mainMenu v třídě Gui.

```
def MainMenu(self,title=""):
        menu = [("(0)","Find Devices"),
                        ("(1)","Load Devices"),
                        ("(2)","Run Scenario"),
                        ("(3)","Run speed test")
                        ]
        code, tag = self.d.menu(title, choices=menu)

        if code == self.d.OK:
            return tag
        else:
            return None
```

Nyní je zde popsána funkce „mainMenu“ v skriptu „master“. Dokud není proměnné „code“ přiřazena hodnota, tak se opakuje neustále ve smyčce. Jak je vidět na výpisu z kódu 3.7.

Výpis 3.7: Funkce mainMenu bez přiřazené hodnoty.

```
def mainMenu():
    code = gui.MainMenu("Menu")
    while code is not None:
```

3.5 Vyhledání podřízených zařízení

Když vybereme volbu „(0) Find Devices“, tak se nám v skriptu „master“ ve funkci „mainMenu“ uloží hodnota „0“ do proměnné „tag“ a je spuštěno vyhledávání podřízených stanic, jak je vidět na výpisku kódu 3.8. Funkce pro kontaktování podřízených stanic je popsána 3.5.1 a pro čekání na odpověď od podřízených stanic 3.5.2. Nalezené a zobrazené podřízené stanice jsou následně uloženy do proměnné „devices“. Následuje podmínka, že pokud je nalezeno, alespoň jedno zařízení spustí se funkce „findMenu“. Když to žádné zařízení nenalezne, tak je vypsána uživateli hláška „no devices Found“ a je navrácen do hlavního menu.

Výpis 3.8: Funkce mainMenu s vybranou volbou „(0) Find Devices“.

```
if code == "(0)":
            master.broadcast()
            devices = master.waitForDevs(gui = gui)
            if devices:
                findMenu(devices)
            else:
                gui.msg("no devices Found")
```

3.5.1 Funkce broadcast

Tato funkce, která je na výpisu kódu 3.9 a je umístěna v knihovně „rpscada“. Odešle UDP datagram se zprávou „RPSCADA-DISCOVERY“ pomocí všesměrové zprávy na všechny adresy v síti na portu 5621.

Výpis 3.9: Funkce broadcast.

```
def broadcast(self):
        with socket.socket(socket.AF_INET,
                           socket.SOCK_DGRAM,
                       socket.IPPROTO_UDP) as s:
            s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
            s.sendto(b'RPSCADA-DISCOVERY', ('255.255.255.255', 5621))
```

3.5.2 Funkce waitForDevs

Následně musí program počkat, než jsou nalezena všechna zařízení v síti s konfigurací pro podřízené stanice. Za tímto účelem volá funkci „waitForDevs“, která se nachází v knihovně „rpscada“ a je vidět na výpisu kódu 3.10. Nejdříve otevře přijímací soket a naslouchá na všech rozhraních jestli mu přijde odpověď na „RPSCADA-DISCOVERY“. V grafického rozhraní je zobrazeno pole s vypsánou hláškou „Finding devices ...“, jak je vidět na obrázku 3.2 a následně jsou do tohoto pole vpisovány nalezené podřízené stanice. Celá tato funkce je nastavena tak, aby trvala 5 vteřin, což je v tomto případě dostačující. Na řádku číslo 9 je zobrazena podmínka ukončení čekání na odpověď od podřízených stanic. Ukončení čekání nastane, když hodnota aktuálního časového údaje „datetime.now“ přestane být menší než hodnota konečného časového údaje „endTime“. Hodnota konečného časového údaje „endTime“ je získána tak, že se do této proměnné po spuštění funkce „waitFor“ uloží hodnota aktuálního časového údaje, ke které je přičtena hodnota argumentu „time“, což je hodnota pěti vteřin, jak je vidět na řádku číslo 4. Poté očekává přijetí dat. Tento proces je blokový a pokud nejsou žádná data přijata, došlo by k zaseknutí, proto bylo nutné vytvořit časovač „sock.settimeout“, který je nastavený na hodnotu jedné vteřiny. To znamená, že pokud za jednu vteřinu neobdrží data, tak nastane chyba a program pokračuje na řádek 20, následně 21 a je opět posunut zpět na podmínku na řádku 9, která zde již byla popsána. Když obdrží data, tak jsou uložena do pole „clients“. Funkce pro zasílání hodnot je popsána v 3.12.3. Přijatá data jsou přečtena a je jim přiřazena nulová hodnota „False“. Nulová hodnota je přiřazena, protože jinak by uživatel nemohl vybrat sledované parametry v grafickém rozhraní (jak je vidět na obrázku 3.4). Následuje uložení hodnot názvu podřízené stanice „hostname“, názvů měřených parametrů „values“ a IP adresy do prázdné proměnné pro databázi „db“. Pokud není k dispozici grafické rozhraní „gui“, tak nenastane chyba a program dále pokračuje. Když grafické rozhraní k dispozici je, tak v něm jsou nalezená zařízení postupně vypisována. Nakonec je navráceno pole „clients“ zpět o úroveň výše.

Výpis 3.10: Funkce waitForDevs.

```
1  def waitForDevs(self, time=5,gui=None):
2          with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
3              sock.settimeout(1)
4              endTime = datetime.now() + timedelta(seconds=time)
5              server_address = ('', 5620)
6              print('starting up on {} port {}'.format(*server_address))
7              sock.bind(server_address)
8              clients = ["Finding devices ..."]
```

```
9               gui.info("\n".join(clients))
10              while datetime.now() < endTime:
11                  try:
12                      data, address = sock.recvfrom(1024)
13                      data = json.loads(str(data, 'utf-8'))
14                      values ={}
15                      for value in data["values"]:
16                          values[value]=False
17                      self.db[data['hostname']]={"values":values,
18                                                 "ip":address[0]}
19                      if gui is not None:
20                          clients.append("{}) {}".format(len(clients),
21                                                         data['hostname']))
22                          gui.info("\n".join(clients))
23                  except:
24                      continue
25
26              return self.db if self.db else None
```

3.5.3 Funkce findMenu

Tato funkce je volána funkcí „mainMenu“ 3.4, aby zobrazila v grafickém rozhraní nalezená zařízení a uživatel si mohl zvolit, se kterou podřízenou stanici pro sběr informací chce pracovat. K tomuto výběru slouží funkce „discoverylist“, která je popsána v 3.5.4. Pokud vybrané zařízení existuje, je spuštěna funkce „setValues“, která slouží k výběru simulovaných parametrů (tato funkce je popsána v podkapitole 3.5.5 a grafické zobrazení můžeme vidět na obrázku 3.3). Pokud se nevrátí požadovaná hodnota od „setValues“ (což znamená, že bylo použito tlačítko „Cancel“ a hodnoty nebyli uloženy), tak se znovu spustí funkce „findMenu“. Když jsou vráceny požadované hodnoty, tak jsou pod názvem zařízení uloženy vybrané hodnoty do listu zařízení. Následně je vše uloženo do databáze „devices.db“, která je umístěna ve složce „data“. Funkce pro ukládání do souboru „saveDB“ je popsána v 3.5.6. Po uložení je program znovu navrácen do funkce „findMenu“. Tento návrat je zde proto, aby uživatel mohl upravovat sledované hodnoty na všech dostupných zařízeních pro sběr informací. Ve chvíli, kdy už není potřeba měnit nastavení zařízení pro sběr informací, se v simulaci vrátí zpět do „mainMenu“ tlačítkem „Cancel“. Funkci „findMenu“ můžete vidět na výpisu z kódu 3.11.

Výpis 3.11: Funkce findMenu.

```
def findMenu(devices):
    device, name = gui.discoveryList(devices,title="Vyberte zařízení:")
    if device is not None:
        device = gui.setValues(device,title="vyberte hodnoty:")
        if device is not None:
            devices[name]=device
            master.db = devices
            master.saveDB()
            findMenu(devices)
        else:
            findMenu(devices)
```

3.5.4 Funkce discoverylist

Aby měly přijaté informace vhodný formát pro čtení uživatelem, do složených závorek se vypíše proměnná psána za tečkou za složenými závorkami. V našem případě to je „key“ pro vypsání názvu uživatele a „devices[key][´ip´)“ pro výpis logické adresy zařízení, jak je vidět na výpisu kódu 3.12. V této funkci opět pracujeme s proměnnými „tag“ a „code“ jak již bylo vysvětleno v 3.4. Tímto způsobem si uživatel vybere podřízenou stanici pro sběr informací, se kterou chce dále pracovat. Nakonec je vrácena hodnota „tag“ do funkce „mainMenu“.

Výpis 3.12: Funkce discoverylist.

```
def discoveryList(self,devices={},title=""):
        localdevices=[]
        for i, key in enumerate(list(devices.keys())):
            localdevices.append(("{}".format(key),
                                 "{} ({})".format(key,devices[key]['ip'])))

        code, tag = self.d.menu(title, choices=localdevices)
        if code == self.d.OK:
            return devices[tag], tag
        else:
            return None, None
```

3.5.5 Funkce setValues

Této funkci je předána hodnota zařízení z funkce „findMenu“. Všechny simulované parametry jsou zde uloženy, jako nevybrané s hodnotou „False“, jak již bylo zmíněno v 3.5.1. Je vytvořen výpis všech dostupných hodnot s přiřazenými hodnotami z databáze, to zapříčiní jestli uživatel uvidí hodnoty jako vybrané nebo nevybrané. Tato funkce je zde umístěna z důvodu, když uživatel zvolí scénář, že uvidí, které simulované parametry pro tento scénář musí použít. Zde je použita stejná proměnná za tečkou „key“, jako u funkce „discoverylist“ a v proměnné „value[key]“ je hodnota „True“ (položka je vybrána) nebo „False“ (položka není vybrána). Následně je vytvořeno grafické rozhraní, pro výběr uživatelem, které je možno vidět na obrázku 3.4. Nakonec je navrácen slovník všech hodnot do funkce „findMenu“ a jednotlivým simulovaným parametrům jsou přiřazeny hodnoty, podle toho jestli jsou vybrané „True“ nebo nevybrané „False“ uživatelem. Funkci „setValues“ můžeme vidět na výpisu z kódu 3.13.

Výpis 3.13: Funkce setValues.

```
def setValues(self,device,title="",backtitle=""):
        blacklist=["product_name","tcpip","identity"]
        values = device['values']
        for black in blacklist:
            values.pop(black, None)

        values = [("{}".format(key),"",values[key]) for key in
        ↪  list(values.keys())]

        code, tags = self.d.checklist(title,
                              choices=values,
                              title="",
                              backtitle=backtitle)
        if code == self.d.OK:
            newdevice = device
            for key in list(newdevice['values'].keys()):
                newdevice['values'][key] = False
            for key in tags:
                newdevice['values'][key] = True

            return newdevice
        else:
            return None
```

3.5.6 Funkce saveDB

Na výpisu kódu 3.14 je vidět, že je otevřen soubor pro zápis. Data z funkce „findMenu“ jsou uložena do souboru „devices.db“ ve formátu JSON. V tomto souboru jsou zapsána a uložena všechna zařízení a jejich vybrané simulované parametry s hodnotou „True“ a nevybrané simulované parametry s hodnotou „False“.

Výpis 3.14: Funkce saveDBs.

```
def saveDB(self):
        with open(self.devicesFile, 'w') as outfile:
            json.dump(self.db, outfile, sort_keys=True, indent=4)
```

3.6 Načítání zařízení

Když vybereme volbu „(1) Load Devices“, tak se nám ve funkci „mainMenu“ v skriptu „master“ uloží hodnota „1“ do proměnné „tag“ a je spuštěno načtení databáze ze souboru, jak je vidět na výpisku kódu 3.15. Načtení zařízení je zde z důvodu, když není potřeba pracovat s nově připojenými stanicemi pro sběr informací. Protože v případě, kdy je program spuštěn a již nadřazená stanice pracovala s těmito podřízenými stanicemi, tak stačí potřebné informace načíst z databáze a nemusí znovu vyhledávat zařízení. Načtení z databáze je za pomoci funkce „loadDB“, která je popsána v podkapitole 3.6.1. Po zvolení zařízení uživatelem, se opět dostáváme do funkce „setValue“ v „findMenu“, aby zvolil s jakými simulovanými parametry je potřeba pracovat, jak již bylo popsáno v 3.5.5.

Výpis 3.15: Načítání zařízení.

```
elif code == "(1)":
            master.loadDB()
            devices = master.db
            if devices:
                findMenu(devices)
            else:
                gui.msg("Local database is empty")
```

3.6.1 Funkce loadDB

Tato funkce, která je vidět na výpisu kódu 3.16, otevírá soubor s databází zařízení pro čtení a případný zápis. Pokud není soubor nalezen, tak je vyvolána vyjímka

„fileNotFoundError“, která chybějící soubor vytvoří a spustí se znovu celá funkce „loadDB“. Po nalezení souboru „devices.db“ je načten a zjistí se zda nenastala při načtení chyba. Pokud se soubor načte v pořádku, je uložen do proměnné „self.db“. V případě, že chyba nastane, tak je soubor smazán, jsou do něj vloženy prázdné závorky a je zavřen. Tím pádem je databáze prázdná.

Výpis 3.16: Funkce loadDB.

```
def loadDB(self):
        try:
            with open(self.devicesFile, 'r+') as f:
                try:
                    self.db = json.load(f)
                except json.decoder.JSONDecodeError:
                        f.seek(0)
                        f.write("{}")
                        f.truncate()
                        self.db={}
        except FileNotFoundError:
            open(self.devicesFile, 'a').close()
            self.loadDB()
```

3.7 Spuštění scénářů

Když vybereme volbu „(2) Run Scenario“, tak se nám ve funkci „mainMenu“ v skriptu „master“ uloží hodnota „2“ do proměnné „tag“ a je spuštěno načtení databáze ze souboru (jak je vidět na výpisku kódu 3.17). Otevře grafické rozhraní, ve kterém jsou zobrazeny všechny dostupné scénáře a uživatel si jeden z nich vybere. Výběr je proveden pomocí funkce „ScenariosMenu“ blíže popsané v 3.7.1. Funkce „ScenariosMenu“ vrací název vybraného scénáře. Název je použit bez posledních tří znaků, protože ve všech případech jsou tyto poslední tři znaky „.py“. Je načten ze složky „scenarios“ jako modul. Tímto způsobem je otevřen soubor „__init__py“, který je popsán v podkapitole 3.7.2. Když program zná vybraný scénář, tak načte všechna zařízení z databáze zařízení pro sběr informací. Je vytvořen prázdný soubor „logfile“, do kterého jsou aktuální hodnoty zapisovány a uživatel může vidět chování simulovaného scénáře. Do proměnné „clients“ jsou nahrané požadavky na stanici pro sběr informací. Tyto požadavky vyhodnocuje funkce v souboru „__init__py“. Když je navrácen seznam vhodných zařízení, tak jsou jednotlivá zařízení procházena. To

znamená, že nejdříve je zjištěna logická adresa a vybrané simulované hodnoty uživatelem daného zařízení pro sběr informací. Nakonec je celý scénář spuštěn v grafickém rozhraní funkce „progressBox“ pro grafické rozhraní je popsána v podkapitole 3.7.4.

Výpis 3.17: Spuštění scénářů.

```
elif code == "(2)":
            scenario_name = gui.ScenariosMenu("Vyberte scénář:")
            if scenario_name:
                scenario_name = str(scenario_name[1])[:-3]
                scenario = importlib.import_module('scenarios')
                master.loadDB()
                devices = master.db
                with open("logfile","w") as f:
                    f.write("")

                T = Thread(target=scenario.run_scenario,args=(scenario_name,
                                                              master,))
                T.start()
                gui.progressBox("logfile")
```

3.7.1 Funkce ScenariosMenu

Veškeré soubory, které jsou uloženy ve složce „scenario“ jsou programem přečteny. Z důvodu, aby nebyly použity soubory, které mají jiné využití než scénáře, je zde vytvořen filtr. Filtrace souborů probíhá jednoduchým způsobem, pokud jejich jméno nezačíná velkými písmeny „SC“, tak nejsou načteny. Následně opět proběhne úprava, pro zobrazení uživateli a po výběru je název scénáře odeslána do „mainMenu“ pomocí tlačítka „OK“. Výpis kódu je možné vidět na 3.18

Výpis 3.18: Funkce ScenariosMenu.

```
def ScenariosMenu(self,title=""):
        scenarios = os.listdir("./scenarios")
        tmp_scenarios = []
        for scenario in scenarios:
            if scenario.startswith("SC"):
                tmp_scenarios.append(scenario)
        scenarios = tmp_scenarios
        scenarios = [("{}".format(i),
             "{}".format(scenario)) for i,scenario in enumerate(scenarios) ]
        code, tag = self.d.menu(title,
```

```
                                    choices= [(a,b[2:]) for a,b in scenarios])

        if code == self.d.OK:
            return scenarios[int(tag)]
        else:
            return None
```

3.7.2 Soubor _init_py

Po spuštění tohoto souboru je provedeno načtení scénářů. Je zadefinována funkce pro spuštění scénáře a zjištění jeho závislostí, kde jako závislost je myšleno každé podřízené zařízení pro sběr informací, se kterým scénář spolupracuje. Jakmile je celý scénář načtený, tak vytvoří objekt „scenario“, kde jsou všechny nadefinované větve scénáře, který je popsán 3.10. Po té je scénář spuštěn a obsah celé funkce je navrácen do „mainMenu“. Obsah souboru „__init__py“ je na výpisku kódu 3.19.

Výpis 3.19: Funkce run_scenario.

```
from . import SCprecerpaniOdpadniVody
from . import SCregulaceTurbiny




def run_scenario(scenario,master):
    scenarios={
    'SCprecerpaniOdpadniVody':SCprecerpaniOdpadniVody.Scenario(master),
    'SCregulaceTurbiny':SCregulaceTurbiny.Scenario(master)
    }
    scenarios[str(scenario)].start()

def get_scenario_dependencies(scenario):
    scenarios={
    'SCprecerpaniOdpadniVody':["DESKTOP-4U2AS9R"],
     'SCregulaceTurbiny':["DESKTOP-4U2AS9R"]
    }
    return scenarios[str(scenario)]
```

3.7.3 Funkce updateValues

Funkce „updateValues“ slouží pro odesílání požadavků pro získání dat od podřízené stanice pro sběr informací a je zobrazena na výpisu kódu 3.20. Pokud není definovaný

název podřízeného zařízení, tak použije předdefinovaný název „localhost“. Když nastane chyba je volána funkce „failure“. Do hodnot funkce „process“ se ukládají proměnné, které posílá podřízená stanice pro sběr informací. Položka „params“ obsahuje parametry, o které žádáme. Toto vlákno je spuštěno separátně. Následně je spuštěna funkce na čtení hodnot „readValues“, kde je spuštěna nekonečná smyčka, která má za úkol získat parametr a hodnotu. Pokud byla přijata hodnota, je vytvořen zámek, pro uzamčení vlákna. Jsou přečtena zařízení z databáze, která je v souboru „data.db“. Pokud není soubor nalezen, tak je vytvořen nový prázdný soubor. Když je soubor „data.db“ nalezen, jsou z něj hodnoty zapsány do proměnné „devices“. Po té je otevřen soubor „data.db“ pro zápis a jsou načteny všechny hodnoty podřízeného zařízení pro sběr informací. Když je načtená hodnota typu „BOOL“ (což znamená, že může mít pouze dva stavy), logickou jedničku například zapnuto nebo logickou nulu například vypnuto. Tak u těchto hodnot typu „BOOL“ je nastaven formát zápisu místo „0“ a „1“ na „False“ a „True“. U jiných datových typů jsou uloženy přímo číselné hodnoty, které byly obdrženy. Hodnoty pro dané podřízené zařízení pro sběr informací jsou tedy upraveny a uloženy do souboru. Následně je uvolněn zámek. Je počkáno jednu deseti tisícinu vteřiny, aby bylo počkáno na novou žádost o hodnoty. Nakonec je proces ukončen a hodnoty předány do „mainMenu“. Tato metoda byla však náročná na zpoždění, proto byla nahrazena funkcí „getValue“, která slouží k získání jednoho parametru z podřízené stanice pro sběr informací a je blíže popsána v podkapitole 3.7.7.

Výpis 3.20: Funkce updateValues.

```
1  def updateValues(self,params,hostname="localhost",name=None,port=44818):
2          if not name:
3              name=hostname
4
5          def failure( exc ):
6              failure.string.append( str(exc) )
7          failure.string                = []
8
9          def process( par, val ):
10             process.values[par]        = val
11
12
13         process.done                 = False
14         process.values                 = {}
15         poller                         = threading.Thread(
16             target=poll.poll, kwargs={
17                 'proxy_class':  device,
18                 'address':      (hostname, port),
```

```
                'cycle':        1,
                'timeout':      1,
                'process':      process,
                'failure':      failure,
                'params':       params,
            })
        poller.start()
        def readValues():
            try:
                while True:
                    while process.values:
                        par,val                 = process.values.popitem()
                        if val:


                            lck.acquire()
                            try:
                                with open(self.dataFile,"r") as f:
                                    try:
                                        devices=json.load(f)
                                    except json.decoder.JSONDecodeError:
                                        devices={}
                            except FileNotFoundError:
                                open(self.dataFile, 'a').close()
                                devices={}

                            with open(self.dataFile,"w") as f:
                                values=devices.get(name,{})
                                if device.PARAMETERS[par][1]=="BOOL":
                                    values[par]= True if val[0] == 1 else
                                    ↪  False
                                else:
                                    values[par]=val[0]
                                devices[name]=values
                                f.write(json.dumps(devices,
                                                   sort_keys=True,
                                                   indent=4))
                            lck.release()


                    while failure.string:
                        exc                 = failure.string.pop( 0 )


                    time.sleep( .0001 )
```

```
65              finally:
66                  process.done                = True
67                  poller.join()
68
69          t = Thread(target=readValues,args=())
70          t.start()
```

3.7.4 Funkce progressBox

Této funkci je dána cesta k souboru, který má být zobrazován v průběhu simulace. Když je soubor přepisován, tak se tato změna promítne ihned do grafického rozhraní, jak je vidět na obrázku 3.6. Na výpisu z kódu 3.21 je zobrazena funkce „progressBox“.

Výpis 3.21: Funkce progressBox.

```
1  def progressBox(self, filepath):
2          excode = self.d.tailbox(filepath)
3          return excode
```

3.7.5 Funkce sV a gV

Obě tyto funkce nalezneme ve třídě „Scenario“ v knihovně „rpscada“ a můžeme je vidět na výpisu z kódu 3.22. Funkce „sV“ slouží k použití funkce „sendValue“ popsané v podkapitole 3.7.6. Je zde definováno s jakou podřízenou stanicí má pracovat na základě logické adresy a kterému parametru je řídící instrukce určena. Zároveň zajišťuje výpis detailních informací, které jsou vidět na obrázku 3.6. Funkce „gV“ je zde z důvodu volání funkce „getValue“, která je popsána v podkapitole 3.7.7 a vrací hodnotu přímo do spuštěného skriptu pro vybraný scénář, která byla zjištěna na podřízené stanici pro sběr informací.

Výpis 3.22: Funkce sV a gV.

```
1  def sV(self,device,variable,value):
2          if self.master.sendValue(self.master.db[device]["ip"],
3                                   variable,value,
4                                   debug=True)==0:
5              self.print("{} is set to {}\n".format(variable,value))
6
```

```
7
8  def gV(self,device,variable):
9          return self.master.getValue(self.master.db[device]["ip"],
10                                     tag=variable,
11                                     debug=True)
```

3.7.6 Funkce sendValue

Funkce „sendValue“ je umístěna ve třídě „master“ v knihovně „rpscada“. Tato funkce posílá požadavky na změnu hodnoty, tedy od nadřazené stanice vysílá řídící instrukce podřízené stanici. Nejdříve je navázáno spojení s podřízenou stanicí, jak je vidět na výpisu z kódu 3.23. Součástí navázání spojení, je nastavení hodnoty atributu. Také je vytvořeno pole operací, která má za úkol vytvořit řídící instrukci ve formátu „název parametru[atribut]=(datový typ) změněná hodnota“. V poli může být obsaženo více než jedna řídící instrukce. Pokud není spojení navázáno za jednu vteřinu, tak je celá operace zrušena. Je navrácena hodnota status kód, který slouží jako zpětná vazba k vyhodnocení komunikace (jednotlivé status kódy jsou k nahlédnutí v příloze E). Celé navázání spojení je uzamčeno zámkem, protože na jeden předefinovaný port může být navázáno pouze jedno spojení. Procesy co chtějí navázat další spojení, jsou řazeny do fronty než je zámek procesem uvolněn. Nakonec je navrácena hodnota status kódu, zdali byla řídící instrukce vykonána.

Výpis 3.23: Funkce sendValue.

```
1  def sendValue(self,hostname,tag, value,debug=False):
2          def write(host,tag="",value="",debug=False):
3              with client.connector( host=host,port=44818 ) as conn:
4                  attr=9
5                  operation = ["{}[{}]=({}){} ".format(tag,
6                                     attr,
7                                     globalParametters[tag]["type"],value)]
8                  for idx,dsc,req,rpy,sts,val in conn.pipeline(
9                              operations=client.parse_operations(operation ,
10                             route_path=[]),
11                             timeout=1):
12                     tag = req["path"]["segment"][0]["symbolic"]
13                     return sts
14         setgetLock.acquire()
15         result = write(hostname,tag,value)
16         setgetLock.release()
17         return result
```

3.7.7 Funkce getValue

Funkce slouží pro zjištění aktuálních hodnot na podřízené stanici určené pro sběr informací. Některé prvky jsou velice podobné funkci „sendValue“, která je popsána v podkapitole 3.7.6 a proto opakující se prvky nejsou znovu popisovány. Je zde odeslán požadavek ve formátu „název parametru[atributu]“, pro získání konkrétní simulované hodnoty. Je navázáno spojení, které navrací pole hodnot a kontroluje hodnotu uloženou v proměnné „tag“. Následně je využit filtr „globalParametters“, který zjišťuje zda, je v proměnné „tag“ uložena hodnota pro jednotku datového typu „BOOL“ a tak upraví typ zápisu z „0“ na „True“ a z „1“ na „False“. Nakonec je navrácena hodnota, o kterou bylo zažádáno. Zdrojový kód této funkce je zobrazen na výpisu z kódu 3.24.

Výpis 3.24: Funkce getValue.

```
def getValue(self,hostname,tag,debug=False):
        def read(host,tag="",debug=False):
            with client.connector( host=host,port=44818 ) as conn:
                attr=9
                operations = client.parse_operations(["{}[{}]".format(tag,
                                                                      attr)],
                                                     route_path=[])

                for idx,dsc,req,rpy,sts,val in conn.pipeline(
                                                operations=operations,
                                                multiple=20,
                                                depth=20,
                                                timeout=1):
                    tag = req["path"]["segment"][0]["symbolic"]
                    ret = val[0]
                    if globalParametters[tag]["type"] == "BOOL":
                        ret = True if ret==1 else False
                    return ret
        setgetLock.acquire()
        result = read(hostname,tag,debug)
        setgetLock.release()
        return result
```

3.8 Měření zpoždění

Když vybereme volbu „(3) Run Speed Test“, tak se nám ve funkci „mainMenu“ v skriptu „master“ uloží hodnota „3“ do proměnné „code“ a je spuštěno testování

připojení protokolu CIP, jak je vidět na výpisku kódu 3.25. Testování má za úkol navázat TCP spojení pro přenos průmyslového protokolu CIP (které je následně vyhodnoceno v podkapitole 4.4). Doba trvání provedených TCP spojení je sečtena a výsledná hodnota je dělena počtem měření. Zpoždění je měřeno od prvního TCP paketu pro navázaní komunikace až po poslední paket potvrzující ukončení TCP spojení a uvolnění prostředků pro další spojení.

Výpis 3.25: Test zpoždění.

```
def SpeedMenu(self,title=""):
        m = Master()
        ip = self.input("ip address","localhost")
        if not ip:
            return None
        tag = self.input("tag","otackomer2")
        if not tag:
            return None
        count = self.input("count","1000")
        if not count:
            return None
        def count_to_int(count):
            try:
                count = int(count)
                if count <= 0:
                    raise ValueError("count is lower then zero")
            except ValueError as e:
                if "count is lower then zero" in str(e):
                    self.msg("count is lower then one")
                    return count_to_int(self.input("count","1000"))
                else:
                    self.msg("count is not number")
                    return count_to_int(self.input("count","1000"))
            except TypeError as e:
                return None
            return count

        count = count_to_int(count)
        self.info("test speed running ...")
        count = int(count)
        try:
            measure = m.speedTest(ip,tag,count)
        except ConnectionRefusedError:
            self.msg("can't reach slave")
            return None
        except Exception:
```

```
37              self.msg("something is wrong with speedtest try it again later")
38              return None
39          average = sum(measure) / len(measure)
40          self.msg("{} microseconds\n average from {} repeats of tag {} to
            ↪  slave {}".format(average, count,tag,ip)
```

3.9 Scénář regulace turbíny

Do skriptu se scénářem jsou nejdříve naimportovány knihovny. Je vytvořeno vlákno s názvem „vetev_turbina“, které je asynchronně spuštěno. Vlákno „vetev_turbina“ je blíže popsáno v podkapitole 3.9.1. Skript scénáře pro simulaci turbíny můžete vidět na výpisu z kódu 3.26. Algoritmus tohoto scénáře naleznete v příloze D.1.

Výpis 3.26: Spuštění scénáře regulace turbíny.

```
1  from time import sleep
2  import json
3  from rpscada import Scenario as SC
4  from rpscada import Master
5  import time
6
7  class Scenario():
8      def __init__(self,master):
9          self.s = SC(master)
10         self.done=False
11         self.t1 = Thread(name="vetev_turbina", target=self.vetev_turbina,
12                                                 args=(self.s,))
13
14     def start(self):
15         self.t1.start()
16
17
18
19 if __name__ == "__main__":
20     master = Master()
21     scenario = Scenario(master)
22     scenario.start()
```

3.9.1 Větev turbína

Ve vlákně „vetev_turbina“ jsou hojně využívané funkce „gV“ a „sV“, popsáné v podkapitole 3.9.1. Celé vlákno je tvořeno za pomocí dvou podmínek. První podmínka rozhoduje, zda je turbína zapnutá či nikoliv. Ukazatel stavu turbíny je datového typu „BOOL“. Dokud je hodnota proměnné „turbina1“ nastavená na „False“, tak se opakuje v nekonečné smyčce. Až uživatel změní hodnotu proměnné „turbina1“ na „True“, tak scénář přejde k druhé podmínce, kterou je počet otáček za minutu. Zde je ukazatelem hodnota proměnná „otackomer2“, která je datového typu Integer. Při prvním průchodu druhou podmínkou má „otackomer2“ hodnotu nula. Aby bylo dosaženo nominálního stavu 8000 otáček za minutu, tak je vydána řídící instrukce „přidej na hodnotu 8000“ a program se navrátí zpět před první podmínku. V reálném provozu je běžné, že nelze najet okamžitě na parametry nominálního provozu turbíny, proto je zde přidávání otáček skokově vždy po 100 otáčkách. Obdobně se chová simulace v případě pokud je překročena hranice 8000 otáček za minutu, jen jsou otáčky ubírány, místo přidávání. Pokud je turbína ustálena na požadovaných 8000 otáčkách za minutu, tak simulace běží v cyklu kdy jsou kontrolovány parametry obou proměnných.

Výpis 3.27: Větev turbína.

```
def vetev_turbina(self,s):
        while not self.done:
            if s.gV("client51","turbina1") is False:
                while s.gV("client51","turbina1") is False:
                    time.sleep(0.1)
                s.print("Turbina najeta")




            hodnota_otacek = s.gV("client51","otackomer2")
            if hodnota_otacek == 8000:
                self.vetev_turbina(s)
            else:
                if hodnota_otacek < 8000:
                    while s.gV("client51","otackomer2")<8000:
                        s.sV("client51","otackomer2",s.gV("client51",
                                                          "otackomer2")+100)
                else:
                    while s.gV("client51","otackomer2")>8000:
                        s.sV("client51","otackomer2",s.gV("client51",
                                                        "otackomer2")-100)
```

3.10 Scénář zpracování odpadní vody

Skript sloužící k simulaci přečerpání odpadní vody je více komplexní, než scénář sloužící k regulaci turbíny, ale uživatel musí být více obezřetný při změně simulovaných hodnot, protože se v něm nachází šest větví a každá větev má krizové stavy, které vedou k přerušení zpracování odpadních vod a ukončení celé simulace. Algoritmus k tomuto scénáři se nachází v příloze D.2. Jako v předchozím scénáři jsou zde hojně využívané funkce „gV“ a „sV“, popsáné v podkapitole 3.9.1. Ve skriptu jsou nejdříve naimportovány potřebné knihovny. Následně je vytvořeno šest vláken, které běží asynchronně. Každé vlákno má svou specifickou funkci. Jednotlivá vlákna jsou vysvětlena v této podkapitole. Spuštění skriptu scénáře zpracování odpadní vody je na výpisu z kódu 3.28. Celý scénář čeká, než je dosaženo maximální hladiny v nádobě na sběr odpadní vody. Maximální hladina je indikována proměnou typu „BOOL“ s názvem „hladina1“. Po dosažení maximální hladiny je spuštěno čerpadlo pro přečerpání odpadní vody. Spuštění čerpadla je zastoupeno opět indikováním hodnoty „BOOL“ v proměnné „cerpadlo1“. Když je čerpadlo zapnuto, jsou spuštěna všechna vytvořená vlákna. Po spuštění čerpadla jsou všechny hodnoty nastavené na nominální hodnotu, takže změny jednotlivých parametrů musí zadat uživatel ručně.

Výpis 3.28: Spuštění scénáře zpracování odpadních vod.

```
1  from threading import Thread
2  from time import sleep
3  import json
4  from rpscada import Scenario as SC
5  from rpscada import Master
6
7  class Scenario():
8      def __init__(self,master):
9          self.s = SC(master)
10
11         self.t1 = Thread(name="vetev_provoz",target=self.vetev_provoz,
12                                                              args=(self.s,))
13         self.t2 = Thread(name="vetev_otacky",target=self.vetev_otacky,
14                                                              args=(self.s,))
15         self.t3 = Thread(name="vetev_teplota",target=self.vetev_teplota,
16                                                              args=(self.s,))
17         self.t4 = Thread(name="vetev_tlak",target=self.vetev_tlak,
18                                                              args=(self.s,))
19         self.t5 = Thread(name="vetev_vodivost",target=self.vetev_vodivost,
20                                                              args=(self.s,))
21         self.t6 = Thread(name="vetev_cilova_nadrz",
```

```
22                                                  target=self.vetev_cilova_nadrz,
23                                                  args=(self.s,))
24          self.client = ["CreapyMachine","DESKTOP-4U2AS9R","client51"]
25          self.client = self.client[0]
26
27      def start(self):
28
29          while not self.s.gV(self.client,"hladina1"):
30              sleep(0.5)
31
32          self.s.sV(self.client,"cerpadlo1",True)
33          if self.s.gV(self.client,"cerpadlo1"):
34              self.t1.start()
35              self.t2.start()
36              self.t3.start()
37              self.t4.start()
38              self.t5.start()
39              self.t6.start()
40
41
42  if __name__ == "__main__":
43      master = Master()
44      scenario = Scenario(master)
45      scenario.start()
```

3.10.1 Větev provoz

Větev provoz slouží k nominální funkci tohoto scénáře. Je vypsána hláška „Zapnulo se cerpadlo.“. Od nadřazené jednotky je poslána řídící instrukce „u_armatura1=True“ pro otevření uzavírací armatury #1, která je datového typu „BOOL“ a řídící instrukce „r_armatura1=20“ pro otevření regulační armatury #1 na 20 %, která je datového typu „DINT“. Nyní simulace běží a očekává indikaci nízké hladiny „hladina2=False“, která je také datového typu „BOOL“. Hodnotu nízké hladiny musí zadat uživatel, protože když by byl nastaven na určitou časovou hodnotu, tak by uživatel nemusel stihnout v simulaci kroky, které si naplánoval. Po té vyšle nadřazená jednotka řídící instrukce „u_armatura1=False“ pro uzavření uzavírací armatury #1, „r_armatura1=0“pro uzavření regulační armatury #1 a „cerpadlo1=False“ pro vypnutí čerpadla. Nakonec je vypsaná informační hláška „Vypnulo se cerpadlo.“, jak je vidět na výpisu z kódu 3.29.

Výpis 3.29: Větev provoz.

```
def vetev_provoz(self,s):
        s.print("Zapnulo se cerpadlo.")
        s.sV(self.client,"u_armatura1",True)
        s.sV(self.client,"r_armatura1",20)
        while s.gV(self.client,"hladina2"):
            sleep(0.5)
            continue
        s.sV(self.client,"r_armatura1",0)
        s.sV(self.client,"u_armatura1",False)
        s.sV(self.client,"cerpadlo1",False)
        s.print("Vypnulo se cerpadlo.")
```

3.10.2 Větev otáčky

Větev pro regulaci otáček byla použita z předchozího scénáře, protože otáčky u čerpadla jsou stejně sledovány jako otáčky u turbíny. Jediný rozdíl je v sledované hodnotě otáček, jelikož čerpadlo dosahuje 2950 otáček za minutu. Větev otáčky je vidět na výpisu z kódu 3.30.

Výpis 3.30: Větev otáčky.

```
def vetev_otacky(self,s):
        if s.gV(self.client,"cerpadlo1")==True:
            if s.gV(self.client,"otackomer1")<2950:
                if not s.gV(self.client,"hladina2"):
                    s.sV(self.client,"otackomer1",2950)
                    self.vetev_otacky(s)
            elif s.gV(self.client,"otackomer1")>2950:
                s.sV(self.client,"otackomer1",2950)
                self.vetev_otacky(s)
            elif s.gV(self.client,"otackomer1")==2950:
                self.vetev_otacky(s)
```

3.10.3 Větev teplota

Větev teplota sleduje simulovaný parametr „teplota1“, který je datového typu „DINT“ a je sledovaný z důvodu, aby nedocházelo k přehřívání čerpadla #1. Pokud je teplota větší než 60 °C, tak nadřazená stanice vysílá řídící instrukci „r_armatura1=+10“, tato akce probíhá do chvíle, než hodnota „teplota1“ klesne pod 60 °C nebo je regulační armatura #1 otevřena na 100 %. Pokud teplota klesla pod 60 °C nebo tuto

hranici nepřesáhla, tak je zkontrolován stav čerpadla #1 a celá smyčka se opakuje, jak je vidět na výpisu kódu 3.31. V případě, že je regulační armatura #1 otevřena na 100 % a teplota je vyšší než 60 °C nebo teplota přesáhla hodnotu 80 °C, tak je vypnuto čerpadlo a vypsána informační hláška „Vysoka teplota na cerpadle!“.

Výpis 3.31: Větev teplota.

```
def vetev_teplota(self,s):
        cerpadlo=True
        while s.gV(self.client,"teplota1")<=60:
            sleep(0.5)
            if s.gV(self.client,"cerpadlo1"):
                continue
            else:
                cerpadlo=False

        if cerpadlo:
            while 60 < s.gV(self.client,"teplota1"):
                sleep(0.2)
                s.sV(self.client,"r_armatura1",s.gV(self.client,
                                                    "r_armatura1")+10)
                if not cerpadlo: break
                if s.gV(self.client,"r_armatura1")==100:
                    s.sV(self.client,"cerpadlo1",False)
                    cerpadlo=False
                    s.print("Vysoka teplota na cerpadle!")
                    break
                elif s.gV(self.client,"r_armatura1")<100:
                    if s.gV(self.client,"teplota1") > 80:
                        s.sV(self.client,"cerpadlo1",False)
                        cerpadlo=False
                        s.print("Vysoka teplota na cerpadle!")
                        break
            if cerpadlo:
                self.vetev_teplota(s)
```

3.10.4 Větev tlak

Větev tlak slouží k upozornění, zda čerpadlo nejede do zavřeného výtlaku. Sleduje rozdíl tlaku mezi sáním a výtlakem čerpadla, protože čerpadlo #1 přidává tlak do potrubí v hodnotě 0,58 MPa a primární tlak potrubí je 0,4 MPa. Je použit výpočet „tlak2“, musí být menší nebo roven „tlak1 + 0,98“, u tlakoměrů je použit

datový typ „REAL“. Pokud hodnota tlakoměru #2 nepřesáhne vypočtenou hodnotu, tak je zjištěno zda čerpadlo # 1 je zapnuté a celá smyčka se opakuje. V případě převýšení vypočtené hodnoty jsou nadřazenou stanicí poslány řídící instrukce „u_armatura1=False“, „cerpadlo1=False“, vypsána informační hláška „Vysoky rozdil tlaku!“ a simulace ukončena. Jak je vidět na výpisu z kódu 3.32.

Výpis 3.32: Větev tlak.

```
def vetev_tlak(self,s):
        cerpadlo = True
        while s.gV(self.client,"tlak2") <= (s.gV(self.client,
                                                 "tlak1") + 0.98):
            sleep(0.5)
            if s.gV(self.client,"cerpadlo1"):
                continue
            else:
                cerpadlo = False
                break
        if cerpadlo:
            s.sV(self.client,"u_armatura1",False)
            s.sV(self.client,"cerpadlo1",False)
            s.print("Vysoky rozdil tlaku!")
```

3.10.5 Větev vodivost

Větev vodivost je v simulaci použita za účelem sledování parametru vodivost, která určuje velikost radioaktivního znečištění odpadní vody. Parametr „mereni_vodivosti1“ je datovým typem „DINT“ a je sledovaná maximální hodnota 330 μ sievertů na centimetr čtvereční. Pokud je aktuální hodnota vodivosti menší nebo rovna 330, tak je provedena kontrola, zda čerpadlo #1 jede, pokud je hodnota čerpadla nastavena na „True“, tak se celá větev opakuje ve smyčce, když čerpadlo #1 nejede, tak je větev ukončena. V případě, že měření vodivosti #1 přesáhne prahovou hodnotu 330, tak jsou poslány nadřazenou stanicí řídící instrukce na uzavření uzavírací armatury #1, odstavení čerpadla #1, vypsána informační hláška „Vysoka vodivost!“ a simulace ukončena. Větev vodivost je zobrazena na výpisu z kódu 3.33.

Výpis 3.33: Větev vodivost.

```
def vetev_vodivost(self,s):
        cerpadlo = True
        while s.gV(self.client,"mereni_vodivosti1") <= 330 :
```

```
            sleep(0.5)
            if s.gV(self.client,"cerpadlo1"):
                continue
            else:
                cerpadlo = False
                break

        if cerpadlo:
            s.sV(self.client,"u_armatura1",False)
            s.sV(self.client,"cerpadlo1",False)
            s.print("Vysoka vodivost!")
```

3.10.6 Větev cílová nádrž

Větev cílové nádrže slouží k obsluze cílových nádrží, aby nenastala situace, kdy je plněná nádrž plná a čerpadlo se neustále snaží plnou nádrž plnit. Po spuštění větve je zkontrolována hodnota „hladina3“, která slouží k indikaci maximální hladiny v nádrži zpracování odpadních vod #1 a je datového typu „BOOL“. Pokud „hladina3“ neindikuje plnou nádrž, tak je zkontrolován stav hodnoty „u_armatura2“, která reprezentuje vstupní uzavírací armaturu do nádrže zpracování odpadních vod #1 a je opět datového typu „BOOL“. V případě, že je uzavírací armatura #2 zavřená, tak je vyslána řídící instrukce „u_armatura2=True“, což zapříčiní otevření uzavírací armatury #2. Následně je provedena kontrola stavu čerpadla. Když čerpadlo jede celé, tak simulace probíhá ve smyčce znovu, pokud je čerpadlo vypnuté větev se ukončí. V případě, že je indikována vysoká hladina v nádrži zpracování odpadních vod #1, tak je poslána řídící instrukce „u_armatura2=False“, na základě které se uzavře uzavírací armatura #2 a pokračuje se k další smyčce této větve. Následující dvě smyčky jsou totožné s již popsanou smyčkou, jen se mění označení jednotlivých prvků v simulaci, jak je vidět na výpisu z kódu 3.34. V případě, že jsou všechny tři nádrže zpracování odpadních vod plné (nastavené hodnoty měření hladiny #1,#2 a #3 na „True“), tak je vypsána hláška „Plne nadrze!“, odstaveno čerpadlo, uzavřena regulační armatura #1, uzavřena uzavírací armatura #1 a simulace ukončena.

Výpis 3.34: Větev cílová nádrž.

```
def vetev_cilova_nadrz(self,s):
        if not s.gV(self.client,"hladina3"):
            if not s.gV(self.client,"u_armatura2"):
                s.sV(self.client,"u_armatura2",True)
            if s.gV(self.client,"cerpadlo1"):
                self.vetev_cilova_nadrz(s)
```

```
7           else:
8               s.sV(self.client,"u_armatura2",False)
9               def hladina4_r():
10                  if not s.gV(self.client,"hladina4"):
11                      if not s.gV(self.client,"u_armatura3"):
12                          s.sV(self.client,"u_armatura3",True)
13                      if s.gV(self.client,"cerpadlo1"):
14                          hladina4_r()
15                  else:
16                      s.sV(self.client,"u_armatura3",False)
17                      def hladina5_r():
18                          if not s.gV(self.client,"hladina5"):
19                              if not s.gV(self.client,"u_armatura4"):
20                                  s.sV(self.client,"u_armatura4",True)
21                              if s.gV(self.client,"cerpadlo1"):
22                                  hladina5_r()
23                          else:
24                              s.print("Plne nadrze!")
25                      hladina5_r()
26
27              hladina4_r()
```

3.11 Spuštění programu v podřízené stanici

V skriptu je nejdříve vytvořen objekt třídy „slave“, která se nachází v knihovně „rpscada“. Do objektu „slave“ jsou vloženy simulované hodnoty, které jsou zobrazeny na výpisu z kódu 3.4. Jsou definována vlákna, rozdělení do vláken je z důvodu, aby byla zajištěná asynchronní funkčnost tohoto skriptu. První vlákno je nazváno „discoveryServerThread“, které spouští funkci „discovery“, ta je popsána v podkapitole 3.12.2. Druhé vlákno se jmenuje „runServerThread“, spouští funkci „runServer“, popsané v podkapitole 3.12.4. Třetí vlákno nazvané „changerTread“, je hlavním vláknem tohoto skriptu, což znamená, že je vidět v konzoly a spouští funkci „changer“, která je blíže přiblížena v podkapitole 3.12.5. Následně jsou vlákna spuštěna. Po spuštění vlákna „changerThread“, se na něj skript „slave“ připojí. Spuštění programu je vidět na výpis z kódu 3.35.

Výpis 3.35: Spuštění programu v podřazené stanici.

```
slave = rpscada.Slave()
slave.values=list(parametters.keys())

discoveryServerThread=Thread(target=discovery)
runServerThread=Thread(target=runServer)
changerThread=Thread(target=changer)
infoThread=Thread(target=info)


if __name__ == "__main__":
    runServerThread.start()
    discoveryServerThread.start()
    time.sleep(5)
    define()

    chT = changerThread
    chT.start()
    chT.join()
```

3.12 Čekání na nadřazenou stanici

Po spuštění podřízené stanice je potřeba, aby tato podřízená stanice naslouchala žádosti o připojení od nadřazené stanice v místní síti. Z tohoto důvodu byla vytvořena funkce „waitForDevs“, která je blíže popsána v podkapitole 3.12.1.

3.12.1 Funkce waitForDevs

Funkce „waitForDevs“ je umístěna ve třídě „Slave“ v společné knihovně „rpscada“ a můžeme ji vidět na výpisu z kódu 3.36. Tato funkce je spuštěna po zapnutí skriptu „slave.py“. Dokud není přijat paket „RPSCADA-DISCOVERY“, tak podřízená stanice stále na tento packet čeká. Po přijetí packetu „RPSCADA-DISCOVERY“, je uzavřeno spojení a navrácena hodnota logické adresy skriptu „slave“ do funkce „discovery“, která je popsána v podkapitole 3.12.2.

Výpis 3.36: Funkce waitForDevs.

```
def waitForDevs(self):
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
            server_address = ('', 5621)
            print('starting up on {} port {}'.format(*server_address))
            sock.bind(server_address)
            while True:
                data, address = sock.recvfrom(1024)
                print(data,address)
                data = str(data, 'utf-8')
                if str(data)=="RPSCADA-DISCOVERY":
                    sock.close()
                    return address[0]
```

3.12.2 Funkce discovery

Tato funkce běží v nekonečné smyčce a očekává přijetí logické adresy. Po navrácení hodnoty logické adresy od funkce „waitForDevs“, která byla popsána v podkapitole 3.12.1, je logická adresa uložena a je napsána informace, že byla nalezena nadřazená stanice na dané logické adrese. Následně je vyvolána funkce „sendValues“, která je popsána v podkapitole 3.12.3. Výpis z kódu této funkce je 3.37.

Výpis 3.37: Funkce discovery.

```
def discovery():
    while True:
        addr = slave.waitForDevs()
        print("found master at ",addr)
        slave.sendValues(addr)
        time.sleep(0.1)
```

3.12.3 Funkce sendValues

Funkce pro zasílání hodnot „sendValues“ se nachází v knihovně „rpscada“ ve třídě „Slave“. Na výpisu kódu 3.38 můžeme vidět, že je otevřeno přímé spojení mezi nadřazenou a podřízenou stanicí. Do této chvíle probíhala komunikace všesměrově a tímto způsobem je zbytečně síť zahlcována. Je vytvořen seznam simulovaných

parametrů, který je upraven do formátu JSON, v podkapitole 3.2 již byly simulované hodnoty představeny. Poté je odeslán název podřízené stanice „hostname“ a seznam simulovaných parametrů „values“, příslušné nadřazené stanici.

Výpis 3.38: Funkce sendValues.

```
def sendValues(self, ip):
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
            forSend = {"hostname":self.hostname,"values":self.values}
            forSend = str(json.dumps(forSend))
            sent = sock.sendto(forSend.encode(), (ip, 5620))
```

3.12.4 Funkce runServer

Zde jsou předány simulované parametry, ze kterých je vytvořeno pole, do kterého jsou vkládány jednotlivé simulované prvky ve formát „název parametru = datový typ [velikost]“. Tyto simulované prvky jsou v poli řazené za sebou a rozděleny mezerou. Následně je spuštěn skript „RPIsimulator“, kterému je předáno pole s daty a je popsán v podkapitole 3.12.6. Na výpisu z kódu 3.39 je zobrazena funkce „runServer“.

Výpis 3.39: Funkce runServer.

```
def runServer():
    global parametters
    params = " ".join(["{}={}[{}]".format(key,
                                          parametters[key]['type'],
                    parametters[key]['size']) for key in parametters.keys()])
    os.system("python3 RPIsimulator.py {}".format(params))
```

3.12.5 Funkce change

Tato funkce očekává vkládání parametrů od uživatele. Tedy když chce uživatel změnit jakékoliv hodnoty za běhu simulace, tak vloží do řádku příkaz „název parametru=požadovaná hodnota“ a potvrdí tlačítkem „enter“. Funkce „change“ z tohoto příkazu vytvoří datové pole o velikosti dvou položek. První položka „param“ obsahuje název parametru a druhá položka „val“ obsahuje požadovanou hodnotu. Následně využije funkci „write“ popsané v podkapitole 3.12.7, na lokálním serveru s logickou adresou (127.0.0.1). Za pomocí funkce „write“, je přepsána hodnota vybraného parametru a navrácen status kód. Funkce „change“ je vidět na výpisu kódu 3.40.

Výpis 3.40: Funkce change.

```
def changer():
    while(True):
        try:
            inputed = input("zadej parametr=hodnota: ").split("=")
            param = inputed[0]
            val = inputed[1]
            write("127.0.0.1",param,val)
        except Exception:
            continue
```

3.12.6 Skript RPIsimulator

Skript „RPIsimulator“ je zobrazen na výpisu kódu v příloze F.4. Tento skript byl připraven v knihovně cpppo, ale autor této práce jej musel upravit, aby vyhovoval požadavkům této simulace. Po převzetí parametrů tento skript spustí komunikační server, přes který prochází požadavky na získání simulované hodnoty nebo řídící instrukce a zároveň je to databáze na straně klienta, kde jsou simulované hodnoty spravovány.

3.12.7 Funkce write

Funkce „write“ je takřka stejná jako funkce „sendValue“ popsaná v podkapitole 3.7.6. Tato funkce posílá požadavky na změnu hodnoty od uživatele, který obsluhuje podřízenou stanici pro sběr informací. Tyto řídící instrukce jsou posílány v logické smyčce podřízené stanice. Je nastavena hodnota atributu na 9. Také je vytvořeno pole operací, které má za úkol vytvořit řídící instrukci ve formátu „název parametru[hodnota atributu]=(datový typ) změněná hodnota“. V poli může být obsaženo více než jedna řídící instrukce. Funkci „write“ můžeme vidět na výpisu z kódu 3.41. Další definovanou funkcí využívající logickou smyčku je funkce „read“, která je opět obdobou funkce „getValue“, popsané v podkapitole 3.7.7.

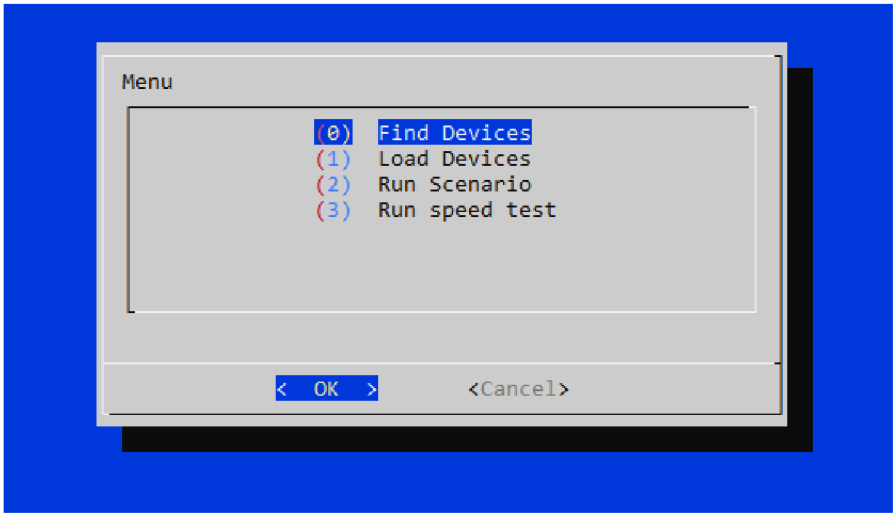
Výpis 3.41: Funkce write.

```
def write(host,tag="",value="",debug=False):
    global parametters
    with client.connector( host=host,port=44818 ) as conn:
        attr=9
        results={}
```

```
6           operation = ["{}[{}]=({}){} ".format(tag,attr,
7                                                parametters[tag]["type"],
8                                                value)]
9           for idx,dsc,req,rpy,sts,val in conn.pipeline(
10                          operations=client.parse_operations( operation ,
11                                                              route_path=[]),
12                                                              timeout=1):
13              tag = req["path"]["segment"][0]["symbolic"]
```

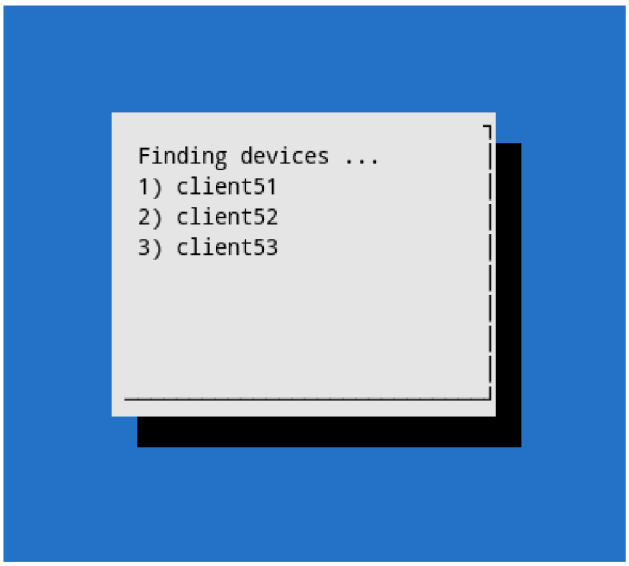
3.13 Grafické rozhraní

V této podkapitole je popsáno grafické rozhraní napomáhající k jednodušší orientaci v simulaci pro uživatele. Po spuštění programu se zobrazí na obrazovce hlavní menu, které je vidět na obrázku 3.1. V celé simulaci je použito intuitivní ovládání pomocí klávesnice. Pro změnu výběru v nabídce je využito šipek nahoru a dolů. Změna akčního tlačítka je ovládána pomocí šipek doprava a doleva. Výběr je potvrzen tlačítkem enter, zaškrtávací pole pomocí tlačítka mezerník a pole do kterých jsou vpisovány hodnoty, jsou ovladatelné myší. Hlavní menu slouží k tomu, aby uživatel mohl zvolit, jakou akci chce provést.



Obr. 3.1: Grafické rozhraní - hlavní menu.

Když uživatel vybere volbu „(0) Find Devices“, která slouží k vyhledání podřízených stanic pro sběr informací v místní síti. Na obrazovce se ukáže dialogové okno, které je vidět na obrázku 3.2, které je zobrazeno pět vteřin. V dialogovém okně jsou postupně vypisovány nalezené stanice pro sběr informací. První vypsaná stanice byla nejdříve nalezena.



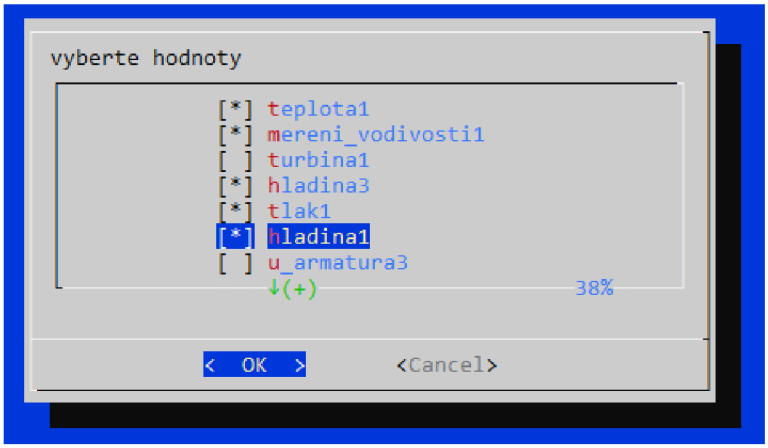
Obr. 3.2: Grafické rozhraní - vyhledávání zařízení.

Následně jsou všechna nalezená zařízení pro sběr informací vypsána, jak je vidět na obrázku 3.3. Název stanice je zde doplněn o logickou adresu této stanice, aby uživatel s jistotou věděl, že vybral správné zařízení i v případě, že více podřízených stanic pro sběr informací má stejný název.

Obr. 3.3: Grafické rozhraní - volba zařízení.

Po výběru z nabídky podřízených stanic je potřeba zvolit se kterými simulova-

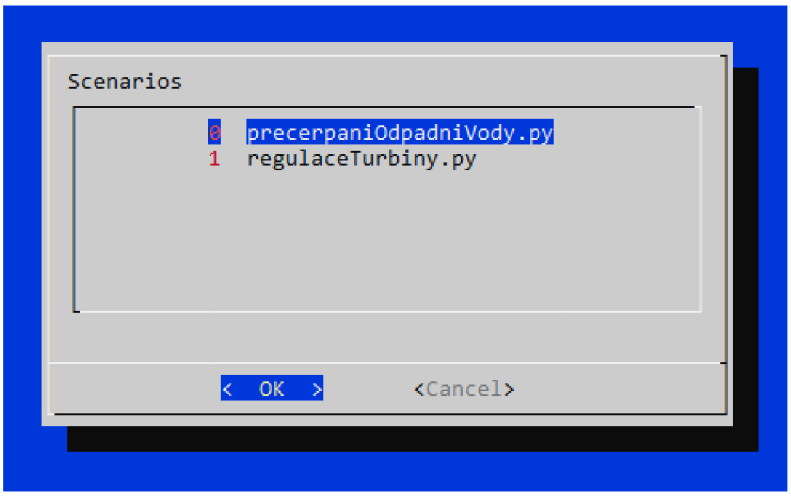
nými parametry s této dané stanice chceme dále pracovat. Výběr parametrů je zobrazen na obrázku 3.4. Pokud je mezi hranatými závorkami závorkami symbol „*“, tak to znamená, že je daný parametr vybraný. Výběr provedeme pomocí mezerníku a uložíme. Výběr hodnot je poslední část volby „(0) Find Devices“ a když v této funkci má uživatel vše nastavené, tak jak potřebuje, musí se vrátit zpět do hlavního menu. Tato funkce slouží pro výběr parametrů, které má funkce „getValues“ shromažďovat do souboru „data.db“.



Obr. 3.4: Grafické rozhraní - volba hodnot.

Když uživatel vybere volbu „(1) Load Devices“, tak je spuštěna funkce, která slouží k načtení podřízených stanic pro sběr informací. Tato funkce je zde z důvodu, aby nebylo potřeba vždy vyhledávat zařízení, když se nezmění složení místní sítě. Zařízení jsou načtena z databáze, která je umístěna v souboru „devices.db“. Po načtení tohoto souboru je zobrazen dialog pro výběr zařízeni, který je vidět na obrázku 3.3 a následuje dialog pro výběr simulovaných hodnot, jak je vidět na obrázku 3.4.

Pokud je vybráno zařízení pro sběr informací a nastaveny vhodné simulované parametry, tak může být spuštěn scénář pomocí volby „(2) Run Scenatio“. Nejprve je zobrazen dialog, který je na obrázku 3.5, kde jsou vypsány všechny dostupné scénáře ze složky „scenarios“. Je zde využita stejná funkce jako v podkapitole 3.7 z důvodu, aby nemusela být vypsána počáteční písmena „SC“, která označují, že se jedná o scénář.



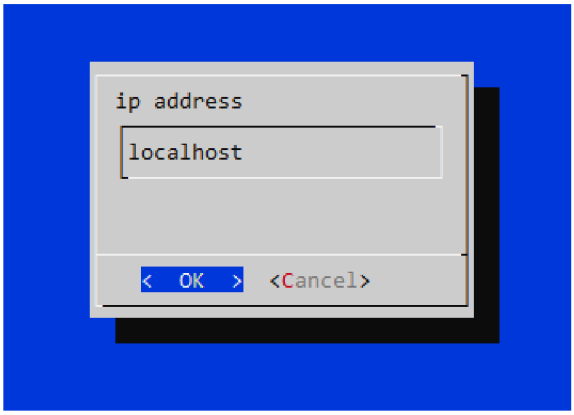
Obr. 3.5: Grafické rozhraní - volba scénáře.

Když uživatel vybere scénář, tak je dotyčný scénář spuštěn a zobrazí se dialog, na kterém je vidět průběh simulace, jak je vidět na obrázku 3.6. Jsou zde vypsány změny stavů v simulaci ve formátu název sledovaného stavu „je změněna na“ hodnotu. Pokud má být na základu změny stavu vypsána informační hláška, jak již bylo popsáno v 2.3. Pokud je zvoleno tlačítko „EXIT“, tak je uživatel navrácen do hlavního menu.

Obr. 3.6: Grafické rozhraní - průběh simulace.

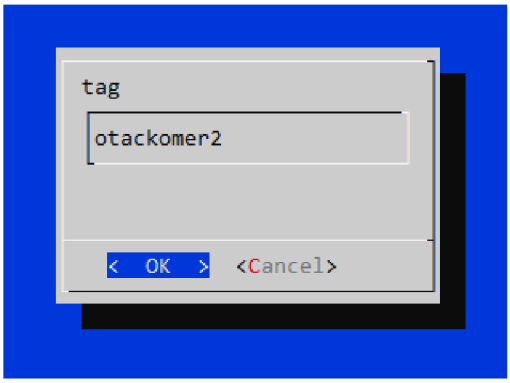
Poslední možností na výběr je „(3) Run speed test“, tato možnost slouží k spuš-

tění testu zpoždění, aby uživatel mohl vidět hodnotu zpoždění po navázání TCP spojení, které využívá pro komunikaci průmyslový protokol CIP. Toto měření je zde z důvodu vnějších vlivů na hodnotu zpoždění jako je zpoždění vznikající na aktivních prvcích v síti nebo délkou přenosového vedení. Protože v případě, kdy je zvolena architektura sítě s menším množství přepínačů nebo při použití výkonnějších přepínačů je hodnota zpoždění menší. Vliv na zpoždění má i délka použité kabeláže. Prvním krokem pří spuštění testu zpoždění je výběr cílové stanice. Na obrázku 3.7 je vidět dialogové okno, ve kterém se nachází pole, do kterého je zadávána logická adresa cílové stanice.



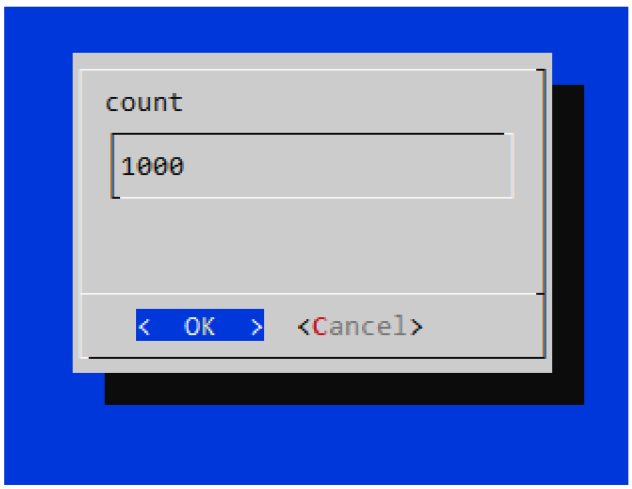
Obr. 3.7: Grafické rozhraní - volba cílové stanice pro test zpoždění.

Po zadání k jakému zařízení se chce uživatel připojit, je zobrazen dialog, který slouží k zadaní simulovaného parametru, na který se uživatel chce dotazovat. Je přednastaven parametr „otackomer2“. Uživatel stejně, jako při předchozím dialogu dopíše parametr do pole v dialogu. Dialog pro výběr parametru je zobrazen na obrázku 3.8.



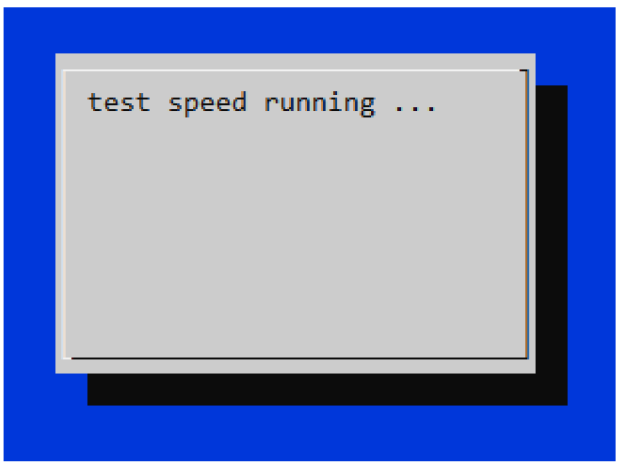
Obr. 3.8: Grafické rozhraní - volba parametru pro test zpoždění.

Poslední parametr, který uživatel volí je počet sledovaných připojení v testu zpoždění. Nejmenší možný počet, který může uživatel zadat je 1 a nejvyšší možný počet měření není omezen (díky tomu si uživatel zvolí potřebný počet navázání spojení). Hodnota počtu měření je po zobrazení dialogového okna nastavena na hodnotu 1000 a opět je přepisovatelná, jak je vidět na obrázku 3.9.



Obr. 3.9: Grafické rozhraní - volba počtu měření.

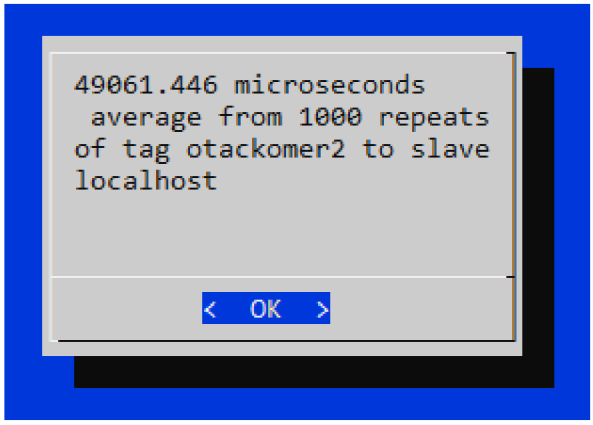
Nežli je test dokončen, zobrazí se dialogové okno, které je vidět na obrázku 3.10. Dialogové okno upozorňuje uživatele, že test probíhá. Délka trvání testu je odlišná. Největší roli v délce testu hraje počet požadovaných měření a hodnota jednotlivých zpoždění.



Obr. 3.10: Grafické rozhraní - průběh měření zpoždění.

Jakmile je test dokončen, zobrazí se dialogové okno zobrazené na obrázku 3.11, které informuje o výsledcích testu zpoždění. Nejdříve je vypsána průměrná hodnota

zpoždění v mikrosekundách. Průměr je vypočten jako součet všech naměřených časů zpoždění pro jednotlivé TCP spojení s protokolem CIP na výše vybraný dotazovaný parametr a hodnota součtu zpoždění je vydělena počtem provedených měření. Uživatel je také informován, na kterou simulovanou hodnotu se dotazoval. Posledním informačním údajem je logická adresa cílové jednotky pro sběr informací, která byla uživatelem vybrána. Po stisknutí tlačítka „OK“ je uživatel navrácen do hlavního menu simulace.



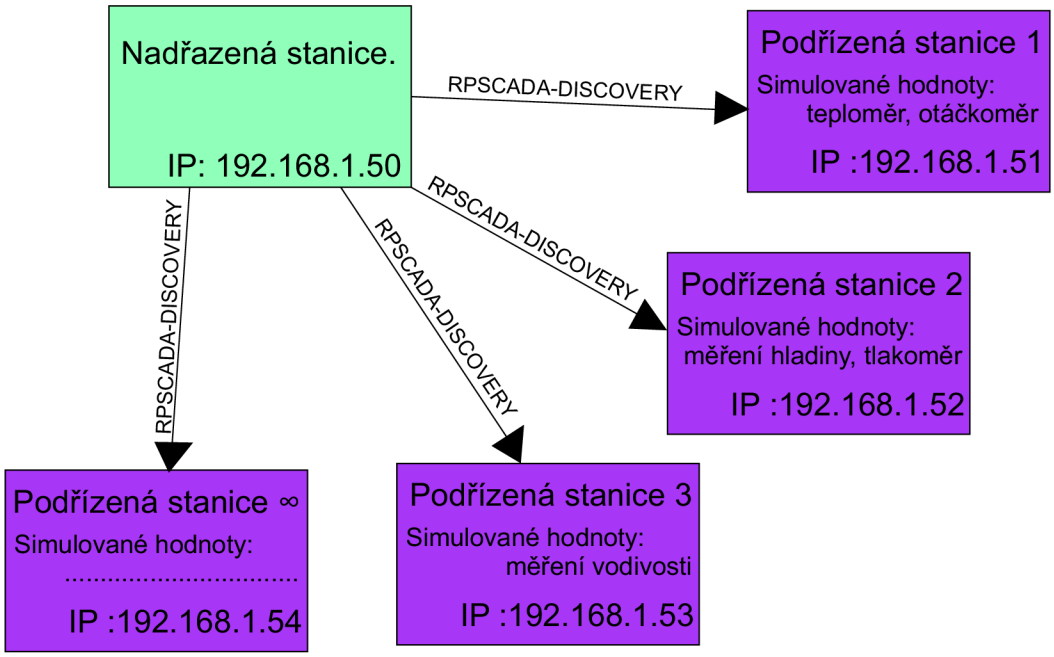
Obr. 3.11: Grafické rozhraní - výsledek měření zpoždění.

4 Analýza komunikace

V této kapitole je simulace popsána z pohledu síťové komunikace, která je pro tuto diplomovou práci stěžejní. Analýza síťového provozu je provedena pomocí programu Wireshark, který je volně dostupný v plné verzi zdarma. Funkce programu Wireshark jsou plně dostačující pro analýzu síťového provozu v simulaci průmyslových protokolů. V podkapitolách níže jsou popsány jednotlivé situace, které nastávají v simulaci průmyslového protokolu CIP. Situace jsou popsány nejdříve teoreticky a následují ukázky z programu Wireshark.

4.1 Vyhledání podřízených stanic – dotaz

Jak již bylo v této práci několikrát zmíněno, je potřeba nejdříve nalézt podřízené stanice, které jsou vhodné pro simulaci a nacházejí se v místní síti. Prvním krokem je, že nadřazená stanice vyšle všesměrovou UDP zprávu „RPSCADA-DISCOVERY“ a je využit port číslo 5621. Jak je vidět na obrázku 4.1.



Obr. 4.1: Dotaz od nadřazené stanice.

4.1.1 Packet RPSCADA-DISCOVERY

Paket „RPSCADA-DISCOVERY“ zachytíme v programu Wireshark je vidět na 4.1, tak že do pole pro filtraci vložíme příkaz „udp.port eq 5621“. V simulaci je využit

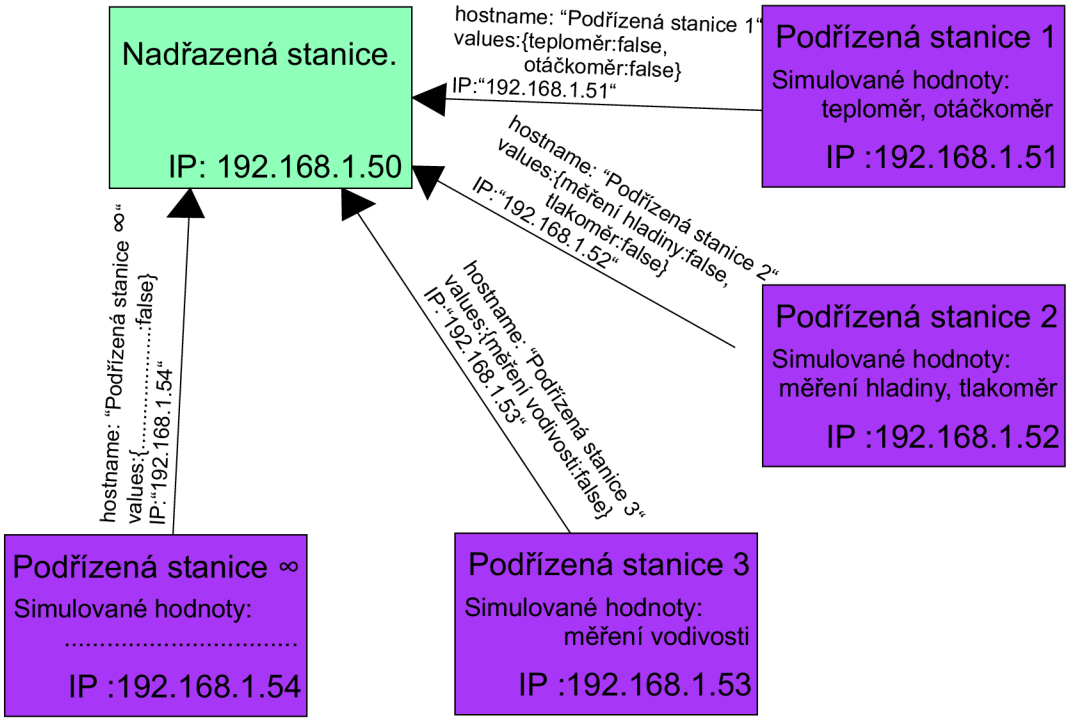
síťový protokol IPv4. Zdrojová logická adresa patří nadřazené stanici (192.168.1.50) a cílová logická adresa je všesměrová pro místní síť (255.255.255.255). Zdrojový port je nastaven dynamicky, což nezpůsobí problém v simulaci, záleží totiž na cílovém portu, na kterém naslouchá podřízená stanice. Cílový port má pevně nastavenou hodnotu 5621. Pole data má velikost 17 bajtů a obsahuje funkci vyhledávání podřízených stanic „RPSCADA-DISCOVERY“.

Výpis 4.1: Zachycený paket RPSCADA-DISCOVERY.

```
Internet Protocol Version 4, Src: 192.168.1.50, Dst: 255.255.255.255
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 45
    Identification: 0x2e97 (11927)
    Time to live: 128
    Protocol: UDP (17)
    Source: 192.168.1.50
    Destination: 255.255.255.255
User Datagram Protocol, Src Port: 63662, Dst Port: 5621
    Source Port: 63662
    Destination Port: 5621
Data (17 bytes)
    Data: 525053434144412d444953434f56455259
    [Length: 17]
--------------------------------------------------------------------------------
0000   02 00 00 00 45 00 00 2d 2e 97 00 00 80 11 00 00   ....E..-........
0010   c0 a8 58 fc ff ff ff ff f8 ae 15 f5 00 19 2a 39   ..X...........*9
0020   52 50 53 43 41 44 41 2d 44 49 53 43 4f 56 45 52   RPSCADA-DISCOVER
0030   59                                                Y
```

4.2 Vyhledání podřízených stanic – odpověď

Podřízená stanice má otevřený port 5621, na kterém naslouchá a očekává příchozí packet „RPSCADA-DISCOVERY“ od nadřazené stanice. Po přijetí packetu „RPSCADA-DISCOVERY“ je uložena logická adresa nadřazené stanice, aby odpověď byla provedena pomocí přímého spojení a místní síť nebyla zbytečně zahlcována všesměrovou komunikací. Na obrázku 4.2 je vidět forma odpovědi. Podřízená stanice poskytne v odpovědi nadřazené stanici informace o hodnotách, které simuluje, svou logickou adresu a svůj název.



Obr. 4.2: Odpověď pro nadřazenou stanici.

4.2.1 Packet s odpovědí na packet RPSCADA-DISCOVERY

Pro zobrazení packetů odpovídajících na packet „RPSCADA-DISCOVERY“ vyhledávající podřízené stanice v zaznamenané síťové komunikaci programem Wireshark, stačí do pole pro nastavení filtru napsat příkaz „udp.port eq 5620“. Odpověď na packet „RPSCADA-DISCOVERY“, používá taktéž síťový protokol IPv4 a transportní protokol UDP. Jak již bylo zmíněno není použita všesměrová komunikace, ale komunikace přímá mezi dvěma stanicemi, proto je nastavena zdrojová logická adresa na adresu podřízené stanice (192.168.1.51) a cílová logická adresa na adresu nadřazené stanice (192.168.1.50). Číslo zdrojového portu se dynamicky mění a cílový port je nastavený pevně na číslo 5620. Tělo této zprávy má velikost 319 bajtů, ale její velikost se mění v závislosti na počtu simulovaných hodnot podřízenou stanicí. V tělu zprávy se nachází název stanice ve formátu „hostname“:„název podřízené stanice“ a výpis všech hodnot v uvozovkách, které simuluje. Jednotlivé hodnoty jsou odděleny symbolem čárky. Celý paket odpovědi zachycený programem Wireshark je vidět na 4.2.

Výpis 4.2: Zachycený paket s odpovědí.

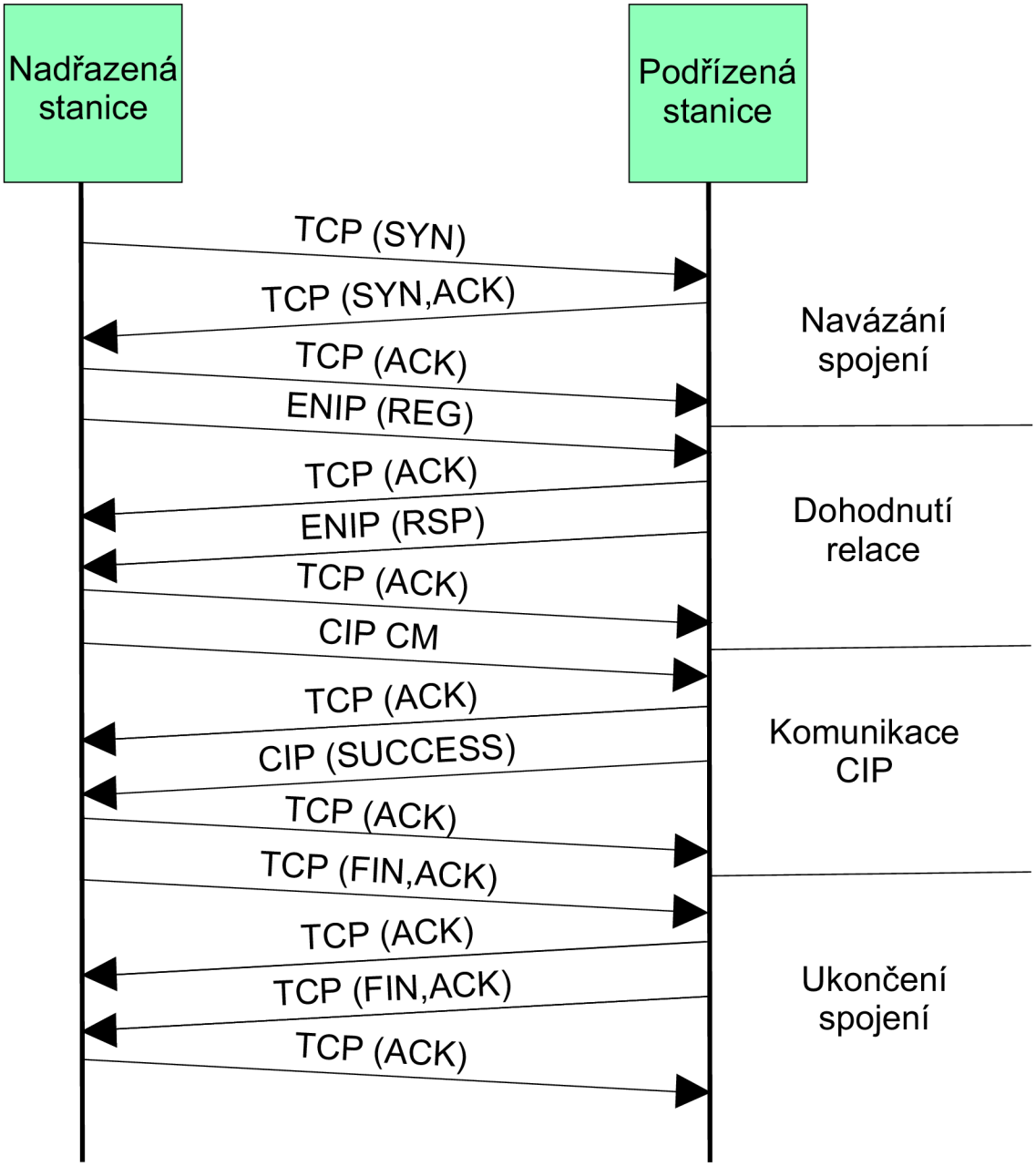
```
Internet Protocol Version 4, Src: 192.168.1.51, Dst: 192.168.1.50
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 347
    Identification: 0x84b0 (33968)
    Time to live: 128
    Protocol: UDP (17)
    Source: 192.168.1.51
    Destination: 192.168.1.50
User Datagram Protocol, Src Port: 52525, Dst Port: 5620
    Source Port: 52525
    Destination Port: 5620
Data (319 bytes)
    Data: 7b22686f73746e616d65223a20224445534b544f502d3455...
    [Length: 319]
```

```
0000   02 00 00 00 45 00 01 5b 84 b0 00 00 80 11 00 00   ....E..[........
0010   c0 a8 58 fc c0 a8 58 fc cd 2d 15 f4 01 47 19 57   ..X...X..-...G.W
0020   7b 22 68 6f 73 74 6e 61 6d 65 22 3a 20 22 44 45   {"hostname": "Cl
0030   53 4b 54 4f 50 2d 34 55 32 41 53 39 52 22 2c 20   ient51",
0040   22 76 61 6c 75 65 73 22 3a 20 5b 22 70 72 6f 64   "values": ["prod
0050   75 63 74 5f 6e 61 6d 65 22 2c 20 22 69 64 65 6e   uct_name", "iden
0060   74 69 74 79 22 2c 20 22 74 63 70 69 70 22 2c 20   tity", "tcpip",
0070   22 74 75 72 62 69 6e 61 31 22 2c 20 22 6f 74 61   "turbina1", "ota
0080   63 6b 6f 6d 65 72 32 22 2c 20 22 74 6c 61 6b 32   ckomer2", "tlak2
0090   22 2c 20 22 74 6c 61 6b 31 22 2c 20 22 6d 65 72   ", "tlak1", "mer
00a0   65 6e 69 5f 76 6f 64 69 76 6f 73 74 69 31 22 2c   eni_vodivosti1",
00b0   20 22 75 5f 61 72 6d 61 74 75 72 61 34 22 2c 20    "u_armatura4",
00c0   22 75 5f 61 72 6d 61 74 75 72 61 33 22 2c 20 22   "u_armatura3", "
00d0   75 5f 61 72 6d 61 74 75 72 61 32 22 2c 20 22 68   u_armatura2", "h
00e0   6c 61 64 69 6e 61 35 22 2c 20 22 75 5f 61 72 6d   ladina5", "u_arm
00f0   61 74 75 72 61 31 22 2c 20 22 68 6c 61 64 69 6e   atura1", "hladin
0100   61 34 22 2c 20 22 68 6c 61 64 69 6e 61 32 22 2c   a4", "hladina2",
0110   20 22 68 6c 61 64 69 6e 61 31 22 2c 20 22 68 6c    "hladina1", "hl
0120   61 64 69 6e 61 33 22 2c 20 22 72 5f 61 72 6d 61   adina3", "r_arma
0130   74 75 72 61 31 22 2c 20 22 63 65 72 70 61 64 6c   tura1", "cerpadl
0140   6f 31 22 2c 20 22 74 65 70 6c 6f 74 61 31 22 2c   o1", "teplota1",
0150   20 22 6f 74 61 63 6b 6f 6d 65 72 31 22 5d 7d       "otackomer1"]}
```

4.3 Komunikace CIP v simulaci

Pro získání aktuálních simulovaných hodnot od podřízené stanice nadřazenou stanicí je využit transportní protokol TCP, díky tomu je docíleno spolehlivého doručení paketů. Teoretický průběh komunikace je vidět na obrázku 4.3. Nejdříve je navázáno

spojení za pomocí tří-cestného podání ruky jak již bylo popsáno v 1.2.3. Následně je dohodnuto číslo relace protokolem ENIP. Je odeslán packet, na který je použit průmyslový protokol „CIP CM“, který obsahuje zprávu s dotazem na požadovanou hodnotu v simulaci nebo zprávu s řídící instrukcí. Přijetí paketu paketu je potvrzeno protokolem TCP. Je odeslán paket protokolem „CIP“ s potvrzením provedení řídící instrukce formou položky status nebo s požadovanou hodnotou. Nakonec je TCP spojení ukončeno.



Obr. 4.3: Komunikace CIP v simulaci.

Celé spojení bylo zachyceno programem Wireshark a je k nahlédnutí na výpisu z programu 4.3. Zde je vidět, že první tři přenesené pakety slouží k navázání TCP spojení. Čtvrtý paket odesílá nadřazená stanice s návrhem relačního čísla. Pátý paket v pořadí je potvrzení přijetí čtvrtého paketu. Paket číslo šest potvrzuje výběr označení relace, na který odpovídá potvrzením sedmý paket. Osmý paket je neregistrovaná zpráva CIP, protože je to první paket přenesený v tomto spojení a obsahuje dotaz na simulovanou hodnotu nebo řídící instrukci. Devátý paket spojení je potvrzení, že osmý paket byl přijat v pořádku. Desátý paket v pořadí je odpověď protokolu CIP obsahující požadovanou hodnotu nebo položku status o provedení řídící instrukce. Jedenáctý paket je TCP potvrzení na o přijetí paketu „CIP“. Následující pakety slouží k ukončení TCP spojení.

Výpis 4.3: Navázání komunikace CIP.

```
Č. Čas:      Zdrojová adresa: Cílová adresa: Protokol: Info:
1  64.947376 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [SYN]
2  64.947448 192.168.1.51     192.168.1.50   TCP      44818 → 50756 [SYN,ACK]
3  64.947511 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [ACK]
4  64.979371 192.168.1.50     192.168.1.51   ENIP     Register Session (Req),
5  64.979404 192.168.1.51     192.168.1.50   TCP      44818 → 50756 [ACK]
6  64.982927 192.168.1.51     192.168.1.50   ENIP     Register Session (Rsp),
7  64.982952 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [ACK]
8  64.986616 192.168.1.50     192.168.1.51   CIP CM   Unconnected Send:
                                                Multiple Service Packet: Service (0x4c)
9  64.986642 192.168.1.51     192.168.1.50   TCP      44818 → 50756 [ACK]
10 65.006489 192.168.1.51     192.168.1.50   CIP      Success:
                                                Multiple Service Packet: Service (0x4c)
11 65.006520 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [ACK]
12 65.014662 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [FIN,ACK]
13 65.014695 192.168.1.51     192.168.1.50   TCP      44818 → 50756 [ACK]
14 65.015394 192.168.1.51     192.168.1.50   TCP      44818 → 50756 [FIN,ACK]
15 65.015430 192.168.1.50     192.168.1.51   TCP      50756 → 44818 [ACK]
```

4.4 Měření zpoždění v simulaci

Jak již bylo v této práci zmíněno, tak měření hodnoty zpoždění, je jeden z hlavních kritérií u průmyslových protokolů obecně. Proto autor této práce provedl automatizované měření zpoždění jak již bylo popsáno v podkapitole 3.8 u komunikace v simulaci protokolu CIP. Zpoždění je měřeno od prvního paketu navazování spojení TCP až po poslední TCP paket, který potvrzuje ukončení spojení jak bylo

vysvětleno v podkapitole 4.3. Aby bylo měření objektivní, tak je provedeno 1000 spojení a z celkového času je vypočten průměr. Měření zpoždění bylo realizováno na navrženém pracovišti s použitím přepínače TP–LINK TL-SF1005D a dvou UTP kabelů kategorie 5e. Pro měření byly použity dva druhy jednodeskových počítačů Rasberry pi. Pro první měření byly použity dvě již dříve zmíněné zařízení Rasberry pi 3B+, které disponují čtyř jádrový procesorem Broadcom BCM2837B0 s frekvenci 1.4 GHz a 1 GB RAM, průměrná hodnota zpoždění byla 345,29036 milisekund. Pro druhé měření byla použita Rasberry pi 4B, které využívá čtyř jádrový 64–bitový procesor s frekvencí 1,5 GHz a 4 GB LPDDR4 SDRAM (Low-Power Double Data Rate Synchronous Dynamic Random Access Memory) s frekvencí 3200 MHz, průměrná hodnota zpoždění byla 117,54677 milisekund. Měření bylo také provedeno v kombinaci, kdy zařízení Rasberry pi 4B představovalo nadřazenou řídící jednotku a zařízení Rasberry pi 3B+ zastupovalo oba typy podřízených stanic, naměřená hodnota průměrného zpoždění byla 187,28871 milisekund. Měření bylo také provedeno softwarově za použití lokální smyčky, kdy výsledné měření nebylo ovlivněno fyzikálními parametry přenosové soustavy, nepůsobilo zpoždění na aktivních prvcích v síti a simulace nebyla omezena výpočetním výkonem Rasberry pi. Průměrná hodnota zpoždění na lokální smyčce je 49,061446 milisekund.

5 Závěr

Cílem diplomové práce byla analýza průmyslových automatizačních protokolů vycházející z protokolu CIP. V první část této diplomové práce je obecné seznámení s protokolem CIP a následuje analýza dvou vybraných průmyslových protokolů. Záměrně byl vybrán nejstarší protokol tohoto typu DeviceNet a nejmodernější protokol EtherNet/IP. Analýza je převážně zaměřena na zobrazení základních parametrů, struktur jednotek, rozdílného využití a provedení zpráv.

Druhá část této práce se věnuje návrhu simulace vybraného průmyslového automatizačního protokolu EtherNet/IP. Toto rozhodnutí bylo učiněno z několika důvodů. V dnešní době je nejvíce implementovaným protokolem v oblasti průmyslové automatizace a lze počítat s tím, že si své prvenství udrží i v budoucnosti. Oproti tomu protokol DeviceNet je již na ústupu, protože je koncipován na jednoduché mikrokontrolery, které jsou v dnešní době, taktéž na ústupu. EtherNet/IP je otevřený protokol, takže informace a knihovny pro zařízení Raspberry Pi jsou k dispozici s bezplatnou licencí a dostatečně přehledné. DeviceNet je uzavřený protokol, k němuž by musely být knihovny pro Raspberry Pi zakoupeny.

Nejdříve je popsán návrh samotné simulace, která je provedena pomocí dvou jednočipových počítačů Raspberry Pi 3B+. Ty jsou pro tyto účely více než dostačující. Dále byly navrženy scénáře výsledné simulace, které jsou koncipovány, tak aby reprezentovala všechny funkce protokolu Ethernet/IP a vychází ze situace ve skutečném provozu. Následuje popis konfigurace jednotlivých zařízení, pro jednosměrnou a obousměrnou komunikaci. Byla použita volně dostupná knihovna pro Raspberry Pi *cpppo*, která využívá scriptovací jazyk python.

Třetí část rozebírá vytvořenou knihovnu a vytvořené skripty autorem této práce a popisuje nezbytné funkce pro správné fungováni výsledné simulace. Je zde popsána realizace a spuštění navržených scénářů. Následně je zde popsáno grafické rozhraní, které je zde vytvořeno z důvodu, aby každý uživatel, který bude se simulací pracovat, nemusel rozumět celému zdrojovému kódu této simulace a bylo zamezeno chybám způsobených nevhodnou manipulací uživatelem v simulaci.

Čtvrtá část se zaobírá analýzou síťové komunikace za běhu simulace průmyslového protokolu Ethernet/IP. Je zde popsán mechanismus vyhledávání podřízených stanic, který vytvořil autor této práce. Následně je popsána komunikace v sítích CIP za využití transportního protokolu TCP a vyhodnocení měření průměrné hodnoty zpoždění, při přenosu řídících instrukcí od nadřazené stanice pro podřízené stanice a získávání aktuálních simulovaných hodnot. Měření průměrného zpoždění je provedeno na navrženém reálném pracovišti pro simulaci protokolu CIP a v softwarovém provedení za použití lokální smyčky.

Literatura

[1] *SCADA System Fundamentals* [online]. New York, 2014 [cit. 2019-12-14]. Dostupné z: <https://www.cedengineering.com/userfiles/SCADA%20System%20Fundamentals.pdf>

[2] Open DeviceNet Vendor Association : *EtherNet/IP Specification, volume 1: CIP Common Specification, Release 1.0*, ControlNet International and Open DeviceNet Vendor Association, June 5, 2001

[3] *8th IEEE International Conference on Emerging Technologies and Factory Automation: proceedings : October 15-18, 2001, Antibes-Juan les Pins, France : ETFA 2001.* 2001. Piscataway, NJ: Institute of Electrical and Electronics Engineers, c2001. ISBN 07-803-7241-7.

[4] Komunikační systémy. *AUTOMA* [online]. 2008, 2008(9), 5 [cit. 2019-12-14]. Dostupné z: <http://www.uamt.feec.vutbr.cz/~zezulka/download/KPPA/A10_08s60_IX.pdf>

[5] Open DeviceNet Vendor Association : *The CIP Networks Library Volume 1: Common Industrial Protocol.* Specification, www.odva.org, 2011

[6] Open DeviceNet Vendor Association : *The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP*. Specification, Dostupné z: <http://read.pudn.com/downloads648/doc/comm/2625669/CIP_Vol2_1.4.pdf>, 2011

[7] Open DeviceNet Vendor Association : *EtherNet/IP Quick Start for Vendors. Handbook,* Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00213R0_EtherNetIP_Developers_Guide.pdf>, 2008

[8] Open DeviceNet Vendor Association : *EtherNet/IPTM - CIP on Ethernet Technology.* Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00138R6_Tech-Series-EtherNetIP.pdf> 2008

[9] Open DeviceNet Vendor Association : *Network Infrastructure for EtherNet/IP - Intrduction and Consideration.* Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00035R0_Infrastructure_Guide.pdf>, 2008

[10] *Integration Technologies for Industrial Automated Systems.* CRC Press, 2018. ISBN 9781351837477.

[11] *EtherNet/IP Adaptation of CIP Specification* [online]. ControlNet International and Open DeviceNet Vendor Association, 2001 [cit. 2019-12-14]. Dostupné z: <http://read.pudn.com/downloads166/ebook/763212/EIP-CIP-V2-1.0.pdf>

[12] *Industrial Communication Technology Handbook: Edition 2.* USA: CRC Press, 2017. ISBN 9781351831376

[13] *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.* York: Springer Science + Business Media, 2007 [cit. 2019-12-14]. Dostupné z: <https://link.springer.com/content/pdf/10.1007%2Fs11241-007-9012-7.pdf>

[14] *The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data* Proceedings of the IEEE, 2007 [cit. 2019-12-14]. Dostupné z: <https://ieeexplore.ieee.org/document/4118467>

[15] Open DeviceNet Vendor Association: *CIP on CAN Technology* [online]. 2016 [cit. 2019-12-14]. Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00026R4_Tech-Adv-Series-DeviceNet.pdf>

[16] Open DeviceNet Vendor Association: *THE COMMON INDUSTRIAL PROTOCOL (CIP)* [online]. 2016 [cit. 2019-12-14]. Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00122R2_CIP-Brochure.pdf>

[17] *Komunikační technologie.* První. Brno: 1, 2013. ISBN 978-80-214-4713-4.

[18] *RFC 768 – User Datagram Protocol*, 1980.

[19] *RFC 793 – Transmission Control Protocol*, 1981

[20] Open DeviceNet Vendor Association (ODVA) ControlNet International (CI) *Recommended IP Addressing Methods for EtherNet/IP Devices* [online]. 2003 [cit. 2019-12-14]. Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00028R0_EtherNet-IP_Addressing_Methods_V1.0.pdf

[21] Brooks, P. *Ethernet/IP – Industrial Protocol. white paper* [online]. 2001 [cit. 2019-12-14]. Dostupné z: <https://literature.rockwellautomation.com/idc/groups/literature/documents/wp/enet-wp001_-en-p.pdf

[22] Open DeviceNet Vendor Association, *The Common Industrial Protocol (CIP) and the Family of CIP Networks.* [online]. 2016 [cit. 2019-12-14]. Dostupné z: <https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R1_Common-Industrial_Protocol_and_Family_of_CIP_Networks.pdf

[23] pjkundert. *Cpppo. Github [online].* 2009 [cit. 2020-05-26]. Dostupné z: <https://github.com/pjkundert/cpppo

Seznam symbolů, veličin a zkratek

ACD	Address Conflict Detection
ACK	Acknowledgement
ASCII	American Standard Code for Information Interface
CAN	Controller Area Network
CID	Connection Identificator
CIP	Common Industrial Protocol
COS	Change Of State
CPF	Common Packet Format
CPPPO	Communications Protocol Python Parser and Originator
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
CRC	Cyclic redundancy check
DNS	Domain Name Server
ENIP	Ecological Network Interaction Protokol
EtherNet/IP	Ethernet Industrial Protokol
FTP	File Transfer Protocol
GB	Gigabyte
GHz	Gigahertz
HTTP	Hypertext Transfer Protocol
ID	Identificator
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IP adresa	Internet protokol - adresa
ISO/OSI	International Organization for Standardization/Open Systems Interconnection
IT	information technology
LPDDR4 SDRAM	Low-Power Double Data Rate Synchronous Dynamic Random Access Memory
MPa	Megapascal
ODVA	Open DeviceNet Vendor Association
PLC	programmable logic controller
PSH	push function
R/R	response/request
RAM	Random Acces Memory
RST	reset the connection
RTR	Remote Transmit Request
SCADA	Supervisory Control And Data Acquisition
SIG	Special Interest Groups

SOF	start of frame
SQL	Structured Query Language
SSH	Secure Shell
SYN	Synchronize Sequence Numbers
TCP	Transmitssion Cotrol Protoco
UCMM	Unconnected Message Manager
UDP	User Datagram Protocol
URG	Urgent

Seznam příloh

A Přehled předdefinovaných kódů

Tab. A.1: Předdefinované kódy pro pole příkaz.

Předdefinované kódy pro pole příkaz	
Název příkazu	Kód
NOP	0x0000
Legacy - vyhrazeno pro zpětnou kompatibilitu	0x0001 - 0x0003
List Servis - seznam služeb	0x0004
Legacy - vyhrazeno pro zpětnou kompatibilitu	0x0005
Rezerva	0x0006 - 0x0062
List Identity - seznam identit	0x0063
List Interfaces - seznam rozhraní	0x0064
Register Session - registrace spojení	0x0065
Un Register Session - zrušení spojení	0x0066
Legacy - vyhrazeno pro zpětnou kompatibilitu	0x0067 - 0x006E
SendRRData	0x006F
SendUnitData	0x0070
Legacy - vyhrazeno pro zpětnou kompatibilitu	0x0071
Indicate Status - indikuje status	0x0072
Cancel - zrušit	0x0073
Legacy - vyhrazeno pro zpětnou kompatibilitu	0x0074 - 0x00C7
Rezerva	0x00C8 - 0xFFFF

Tab. A.2: Přehled předdefinovaných objektů.

Předdefinované objekty pro obecné použití			
definice objektu	ID objektu	definice objektu	ID obj.
shromáždění	(0x04)	správce připojení	(0x06)
router zpráv	(0x02)	konfigurace připojení	(0xF3)
identita	(0x01)	připojení	(0x05)
výběr	(0x2E)	parametr	(0x0F)
soubor	(0x37)	skupina parametrů	(0x10)
registr	(0x07)	podpisové potvrzení	(0x2B)
port	(0xF4)	seznam připojení původce	(0x45)
Předdefinované objekty pro konkrétní aplikace			
definice objektu	ID objektu	definice objektu	ID obj.
analogová skupina	(0x22)	skupina analogových vstupů	(0x20)
analogový vstupní bod	(0x0A)	diskrétní skupina	(0x1F)
diskrétní vstupní skupina	(0x1D)	diskrétní výstupní skupina	(0x1E)
diskrétní vstupní bod	(0x08)	diskrétní výstupní bod	(0x09)
údaje o motoru	(0x28)	přetížení	(0x2C)
regulátor polohy	(0x25)	snímač polohy	(0x23)
objekt řízení spotřeby	(0x53)	kalibrace s-plynu	(0x34)
Předdefinované objekty specifické pro síť			
definice objektu	ID objektu	definice objektu	ID obj.
základní přepínač	(0x51)	QoS	(0x48)
SNMP	(0x52)	most RSTP	(0x54)
port RSTP	(0x55)	rozhraní TCP/IP	(0xF5)
CompoNet připojení	(0xF7)	CompoNet opakovač	(0xF8)
ControlNet	(0xF0)	ControlNet keeper	(0xF1)
DeviceNet	(0x03)	Ethernet připojení	(0xF6)
tabulka uzlů PRP	(0x57)	protokol paralelní redundance	(0x56)

B Kód konfigurace pro obousměrnou komunikaci za pomoci vyčítání z konzole

Výpis B.1: ioclient.py

```
import time
import random
import sys, logging
import cpppo
from cpppo.server.enip import address, client

if __name__ == "__main__":
  logging.basicConfig( **cpppo.log_cfg )
  host                          = '192.168.1.50'
  port                          = address[1]
  depth                         = 1
  multiple                      = 20
  fragment                      = False
  timeout                       = 1.0
  printing                      = True



while True:
 i = random.randrange(500,1500)
 with client.connector( host=host, port=port, timeout=timeout ) as
  ↪  connection:
        tags = ["Tag[0-9]+16=(DINT){}".format(i), "@0x2/1/1", "Tag[3-5]"]
        operations = client.parse_operations( tags )
        failures,transactions       = connection.process(
        operations=operations, depth=depth, multiple=multiple,
        fragment=fragment, printing=printing, timeout=timeout )

sys.exit( 1 if failures else 0 )
```

Výpis B.2: pollserver.py

```
import subprocess
import shlex
import json
import time
import sys, logging
import cpppo
import os
from cpppo.server.enip import address, client

logging.basicConfig( **cpppo.log_cfg )
host                        = '192.168.1.51'
port                        = 8090
depth                       = 1
multiple                    = 20
fragment                    = False
timeout                     = 1.0
printing                    = False



def sendResponse(data):
 sys.stdout = open(os.devnull, "w")
 with client.connector( host=host, port=port, timeout=timeout ) as
  ↪  connection:
     tags = ["Tag[0-9]+16=(DINT){}".format(data), "@0x2/1/1", "Tag[3-5]"]
     operations = client.parse_operations( tags )
     failures,transactions       = connection.process(
     operations=operations, depth=depth, multiple=multiple,
     fragment=fragment, printing=printing, timeout=timeout )
 sys.stdout = sys.__stdout__



process = subprocess.Popen(shlex.split("python3 -m cpppo.server.enip
                                                    Tag=DINT[10]"),sh$



while True:
  output = process.stdout.readline()
  if process.poll() is not None:
```

```
42      break
43    if output:
44      print(output.strip())
45      data = str(output.strip()).split("<=")
46      result = int(json.loads(data[1][0:-1])[0])
47      sendResponse(1000-result)
48
49  rc = process.poll()
```

Výpis B.3: START.sh

```
1  python3 -m cpppo.server.enip -a 0.0.0.0:8090 Tag=DINT[10]
```

C Kód konfigurace pro obousměrnou komunikaci v reálném čase

Výpis C.1: controller.py

```
from __future__ import print_function
from cpppo.server.enip import client
import time
from pprint import pprint
import datetime

hostTemp  = ("192.168.1.50", 44818)
hostMotor = ("192.168.1.50", 44819)
tags = [ "scada[0-10]", "scada[1]=99", "scada[0-10]" ]

with client.connector( host=hostTemp[0], port=hostTemp[1] ) as conn:
     t1 = datetime.datetime.utcnow()
     req1 = conn.write( "scada[1-3]", data=[111,222,333] )
     req2 = conn.read( "scada[2]" )
     t2 = datetime.datetime.utcnow()
     pprint(req2)
     print("---------->",t1,"\n---------->",t2)
     assert conn.readable( timeout=1.0 ), "Failed to receive reply 1"
     rpy1 = next( conn )
     assert conn.readable( timeout=1.0 ), "Failed to receive reply 2"
     rpy2 = next( conn )


with client.connector( host=hostMotor[0], port=hostMotor[1] ) as conn:
     t1 = datetime.datetime.utcnow()
     req1 = conn.write( "scada[1-3]", data=[111,222,333] )
     req2 = conn.read( "scada[2]" )
     t2 = datetime.datetime.utcnow()
     pprint(req2)
     print("---------->",t1,"\n---------->",t2)
     assert conn.readable( timeout=1.0 ), "Failed to receive reply 1"
     rpy1 = next( conn )
     assert conn.readable( timeout=1.0 ), "Failed to receive reply 2"
     rpy2 = next( conn )
```

Výpis C.2: Výpis na konzoly pro sběr informací.

```
10-23 20:28:36.643 MainThread root     NORMAL   main
Loaded config files: []
```

```
10-23 20:28:36.645 MainThread enip.srv NORMAL   main
Delaying all responses by 0.0 seconds
10-23 20:28:36.647 MainThread enip.srv NORMAL   main
Creating tag: scada                                DINT[1000]
10-23 20:28:36.647 MainThread root     NORMAL   main
EtherNet/IP Simulator: ('', 44818)
10-23 20:28:36.648 MainThread network  NORMAL   server_mai
enip_srv server PID [  708] running on ('', 44818)
10-23 20:28:36.648 MainThread network  NORMAL   server_mai
enip_srv server PID [  708] responding to external done/disable
signal in object 1989538896
10-23 20:28:36.650   enip_srv enip.srv NORMAL   enip_srv
EtherNet/IP Server enip_UDP begins serving peer None
10-23 20:28:48.330   enip_srv enip.srv NORMAL   enip_srv
EtherNet/IP Server enip_56830 begins serving peer ('192.168.1.50', 56830)
10-23 20:28:48.585   enip_srv enip.ucm NORMAL   __init__
UCMM accepts route_path: (any)
10-23 20:28:48.588   enip_srv enip.lgx NORMAL   setup
Logix                         Instance    1, Attribute    1 added: scada
DINT[1000] == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
               scada[    1-3    ] <= [111, 222, 333]
               scada[    2-2    ] == [222]
10-23 20:28:48.855   enip_srv enip.srv NORMAL   enip_srv_t enip_56830 done;
processed   3 requests over   180 bytes/  180 received (0 connections remain)
10-23 20:29:55.636   enip_srv enip.srv NORMAL   enip_srv
EtherNet/IP Server enip_56834 begins serving peer ('192.168.1.50', 56834)
               scada[    1-3    ] <= [111, 222, 333]
               scada[    2-2    ] == [222]
10-23 20:29:56.249   enip_srv enip.srv NORMAL   enip_srv_t enip_56834 done;
processed   3 requests over   180 bytes/  180 received (0 connections remain)
```

Výpis C.3: Výpis na konzoly řízeného zařízení.

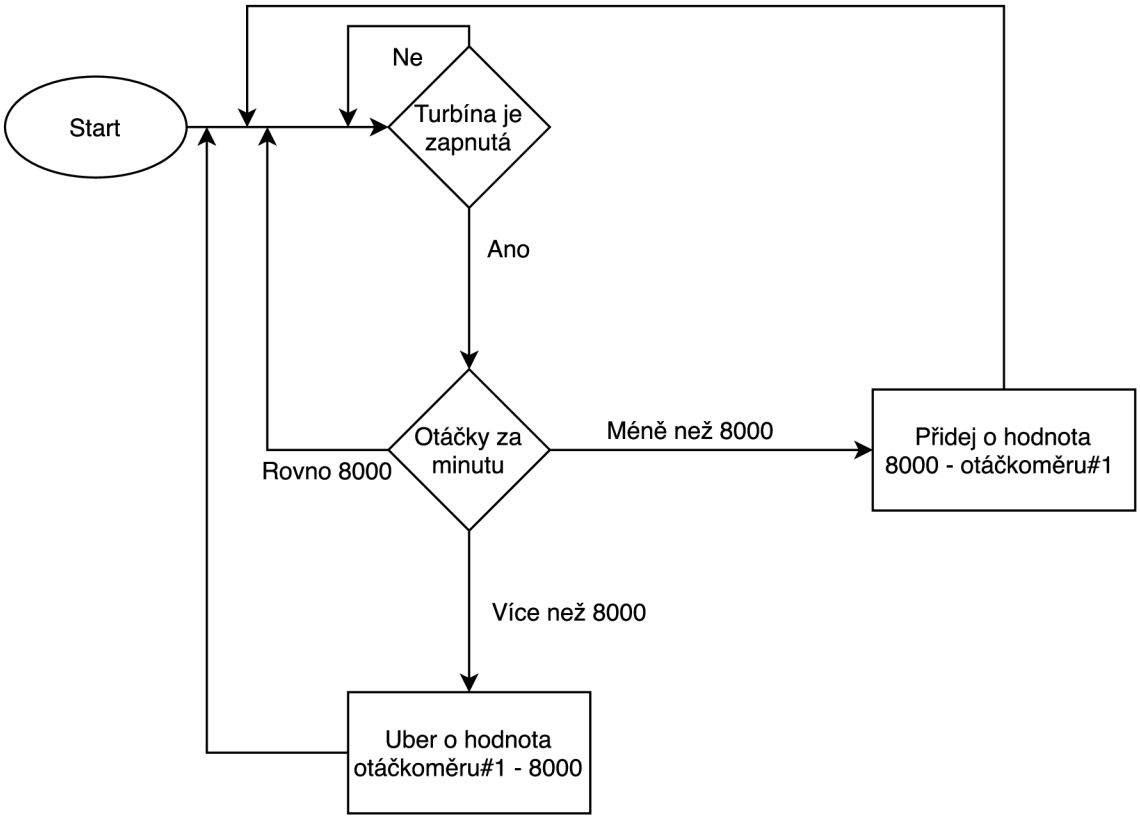
```
10-23 20:29:48.576 MainThread root     NORMAL   main
Loaded config files: []
10-23 20:29:48.577 MainThread enip.srv NORMAL   main
Delaying all responses by 0.0 seconds
10-23 20:29:48.579 MainThread enip.srv NORMAL   main
Creating tag: a                              DINT[1000]
10-23 20:29:48.580 MainThread root     NORMAL   main
EtherNet/IP Simulator: ('', 44819)
10-23 20:29:48.580 MainThread network  NORMAL   server_mai
enip_srv server PID [  712] running on ('', 44819)
10-23 20:29:48.581 MainThread network  NORMAL   server_mai
enip_srv server PID [  712]
responding to external done/disable signal in object 1985181616
10-23 20:29:48.582   enip_srv enip.srv NORMAL   enip_srv
EtherNet/IP Server enip_UDP begins serving peer None
10-23 20:29:56.040   enip_srv enip.srv NORMAL   enip_srv
EtherNet/IP Server enip_50680 begins serving peer ('192.168.1.50', 50680)
10-23 20:29:56.229   enip_srv enip.ucm NORMAL   __init__
```

```
UCMM accepts route_path: (any)
10-23 20:29:56.231   enip_srv enip.lgx NORMAL   setup
Logix                         Instance   1, Attribute   1 added: a
DINT[1000] == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
10-23 20:29:56.323   enip_srv enip.dev WARNING  route
Logix Failed attempting to resolve path
```

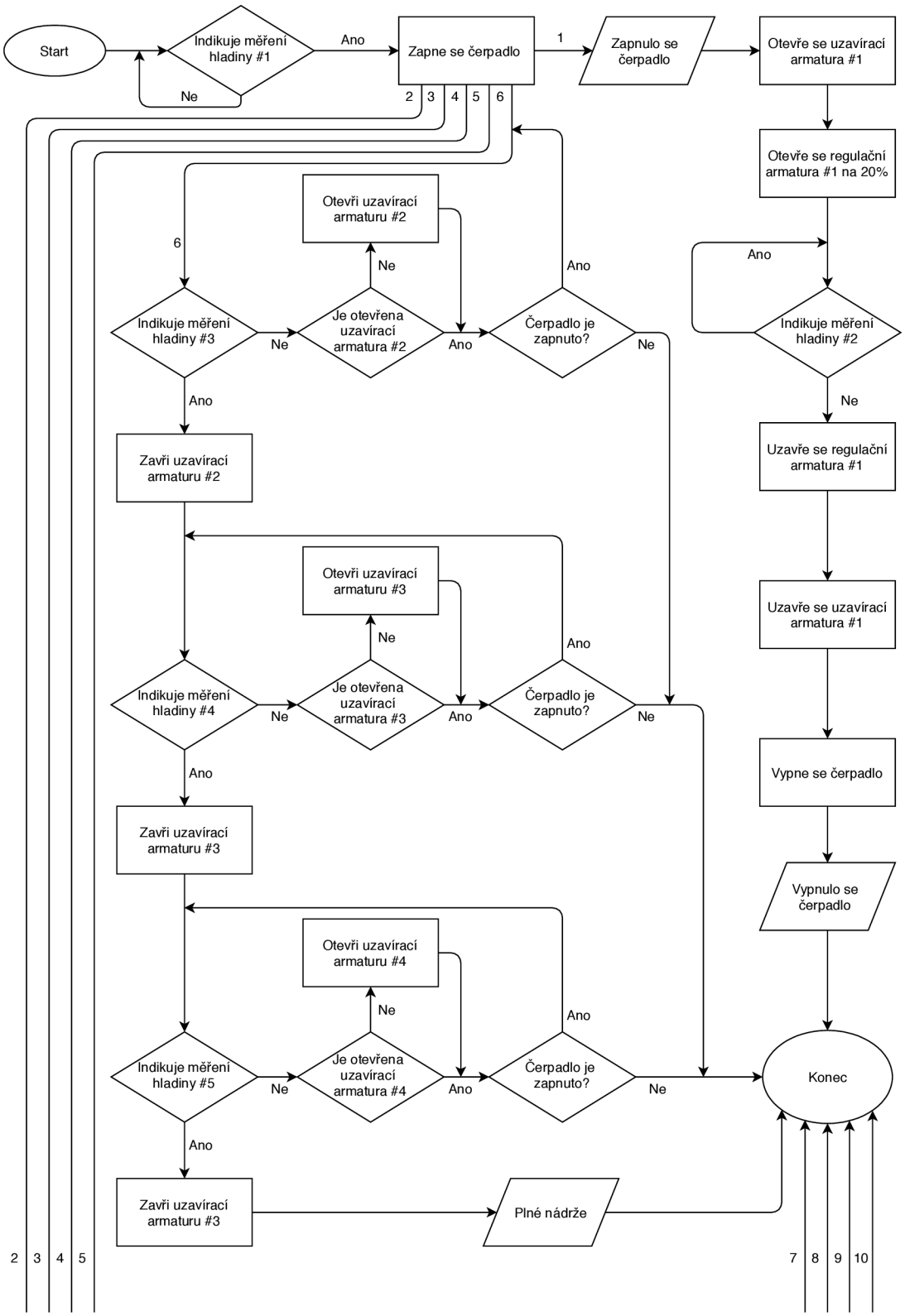
```
{'size': 5, 'segment': [{'symbolic': 'scada'}, {'element': 1}]}:
class,inst,addr: None, target: None
10-23 20:29:56.325   enip_srv enip.lgx NORMAL   request    (0x02,  1)
Logix Service 0xd3 Write Tag Fragmented Reply failed with Exception:
Unrecognized symbolic name 'scada' found in path [{'symbolic': 'scada'},
{'element': 1}]
Request: {
    "input": "array('B', [83, 5, 145, 5, 115, 99, 97, 100, 97, 0, 40, 1,
195, 0, 3, 0, 0, 0, 0, 0, 111, 0, 222, 0, 77, 1])",
    "service": 211,
    "path.size": 5,
    "path.segment[0].symbolic": "scada",
    "path.segment[1].element": 1,
    "status": 5,
    "write_frag.type": 195,
    "write_frag.data": [
        111,
        222,
        333
    ],
    "write_frag.offset": 0,
    "write_frag.elements": 3,
    "status_ext.size": 1,
    "status_ext.data": [
        0
    ]
}

10-23 20:29:56.387   enip_srv enip.dev WARNING  route
Logix Failed attempting to resolve path {'size': 5, 'segment':
[{'symbolic': 'scada'}, {'element': 2}]}:
class,inst,addr: None, target: None
10-23 20:29:56.389   enip_srv enip.lgx NORMAL   request    (0x02,  1)
Logix Service 0xd2 Read Tag Fragmented Reply failed with Exception:
Unrecognized symbolic name 'scada' found in path [{'symbolic': 'scada'},
{'element': 2}]
Request: {
    "input": "array('B', [82, 5, 145, 5, 115, 99, 97, 100, 97,
0, 40, 2, 1, 0, 0, 0, 0, 0])",
    "service": 210,
    "path.size": 5,
    "path.segment[0].symbolic": "scada",
    "path.segment[1].element": 2,
    "status": 5,
    "read_frag.offset": 0,
    "read_frag.elements": 1,
    "status_ext.size": 1,
```

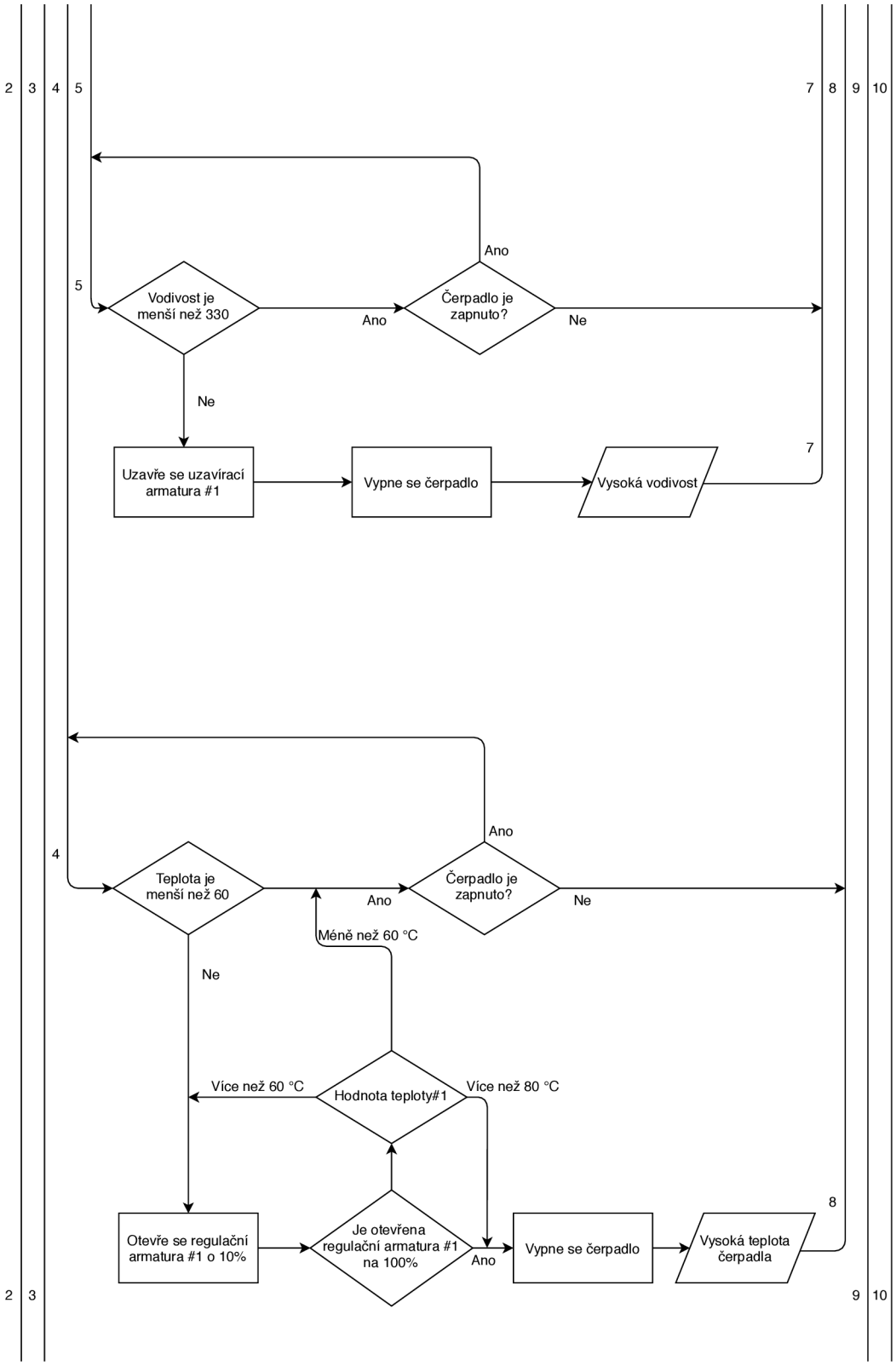
```
    "status_ext.data": [
        0
    ]
}

10-23 20:29:56.460   enip_srv enip.srv NORMAL   enip_srv_t enip_50680 done;
processed   3 requests over   180 bytes/  180 received (0 connections remain)
```

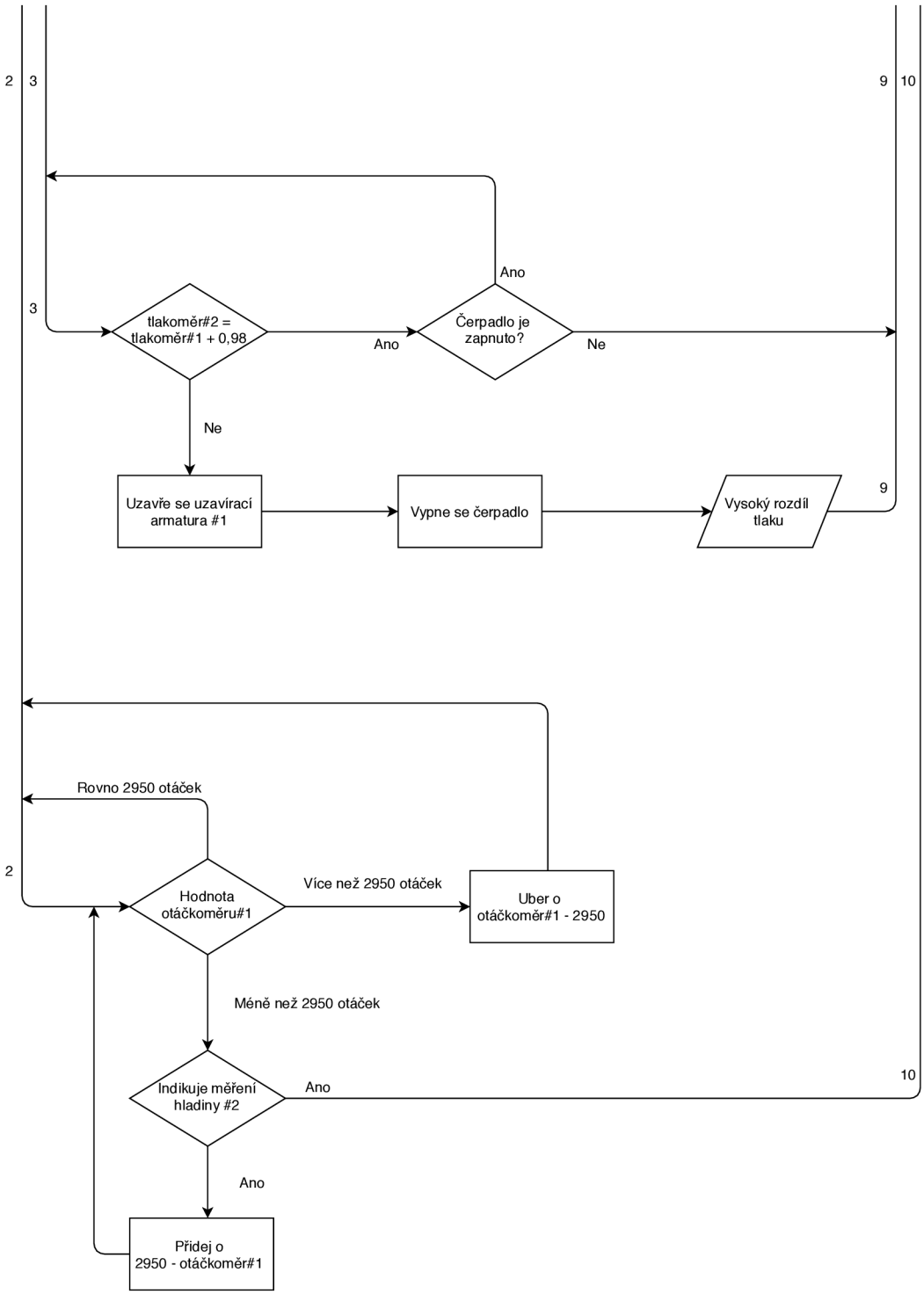
D Algoritmy scénářů



Obr. D.1: Algoritmus scénáře pro regulaci turbíny


Start
Indikuje měření hladiny #1
Ano
Ne
Zapne se čerpadlo
1
Zapnulo se čerpadlo
Otevře se uzavírací armatura #1
Otevře se regulační armatura #1 na 20%
Indikuje měření hladiny #2
Ano
Ne
Uzavře se regulační armatura #1
Uzavře se uzavírací armatura #1
Vypne se čerpadlo
Vypnulo se čerpadlo
Konec
2 3 4 5 6
6
Otevři uzavírací armaturu #2
Indikuje měření hladiny #3
Je otevřena uzavírací armatura #2
Čerpadlo je zapnuto?
Zavři uzavírací armaturu #2
Otevři uzavírací armaturu #3
Indikuje měření hladiny #4
Je otevřena uzavírací armatura #3
Čerpadlo je zapnuto?
Zavři uzavírací armaturu #3
Otevři uzavírací armaturu #4
Indikuje měření hladiny #5
Je otevřena uzavírací armatura #4
Čerpadlo je zapnuto?
Zavři uzavírací armaturu #3
Plné nádrže
2 3 4 5
7 8 9 10


2
3
4
5
7
8
9
10
Ano
Vodivost je menší než 330
Čerpadlo je zapnuto?
Ano
Ne
Ne
Uzavře se uzavírací armatura #1
Vypne se čerpadlo
Vysoká vodivost
Ano
Teplota je menší než 60
Čerpadlo je zapnuto?
Ano
Ne
Méně než 60 °C
Ne
Více než 60 °C
Hodnota teploty#1
Více než 80 °C
Otevře se regulační armatura #1 o 10%
Je otevřena regulační armatura #1 na 100%
Ano
Vypne se čerpadlo
Vysoká teplota čerpadla



Obr. D.2: Algoritmus scénáře pro zpracování odpadních vod

E Přehled kódů položky status

Tab. E.1: Přehled kódů položky status

Název statusu	Kód statusu $[0000_H]$	Definice statusu
Úspěch	00	Služba byla úspěšně provedena zadaným objektem.
Selhání připojení	01	Služba připojení selhala na straně připojovací cesty.
Zdroj není k dispozici	02	Zdroje potřebné k provedení požadované služby pro objekt nebyly k dispozici.
Neplatná hodnota parametru	03	Viz Stavový kód 0x20, což je preferovaná hodnota, která se má použít pro tuto podmínku.
Chyba segmentu cesty	04	Zpracovatelský uzel nerozuměl identifikátor segmentu cesty ani syntaxi segmentu. Zpracování cesty se zastaví, když dojde k chybě segmentu cesty.
Neznámá cesta	05	Cesta odkazuje na objektovou třídu, instanci nebo prvek struktury, která není známa nebo není obsažena v uzlu zpracování. Zpracování cesty se zastaví, jakmile dojde k neznámé chybě cíle cesty.
Částečný přenos	06	Byla přenesena pouze část očekávaných dat.
Ztráta připojení	07	Připojení bylo ztraceno.
Služba není podporována	08	Požadovaná služba nebyla pro tento objekt implementována nebo nebyla definována Třída / Instance.
Neplatná hodnota atributu	09	Byla zjištěna neplatná data atributů.
Chyba v seznamu atributů	0A	Atribut v odpovědi Get Attribute List nebo Set Attribute List má nenulový stav.
Již je v požadovaném stavu/režimu	0B	Objekt je již v režimu / stavu požadovaném službou.
Konflikt stavu objektu	0C	Objekt nemůže provést požadovanou službu v aktuálním režimu / stavu.

Název statusu	Kód statusu $[0000_H]$	Definice statusu
Objekt již existuje	0D	Požadovaná instance objektu, který má být vytvořen, již existuje.
Atribut nelze nastavit	0E	Byl přijat požadavek na úpravu nemodifikovatelného atributu.
Porušení oprávnění	0F	Selhalo ověření oprávnění.
Konflikt stavu zařízení	10	Aktuální režim / stav zařízení zakazuje provádění požadované služby.
Odpověď je příliš velká	11	Data, která mají být vyslána do vyrovnávací paměti odpovědí, jsou větší než přidělená vyrovnávací paměť odpovědi.
Fragmentace primitivní hodnoty	12	Služba specifikovala operaci, která se chystá fragmentovat primitivní datovou hodnotu, tj. Polovinu datového typu REAL.
Nedostatek dat	13	Služba neposkytla dostatek dat k provedení zadané operace.
Atribut není podporován	14	Atribut uvedený v žádosti není podporován.
Příliš mnoho dat	15	Služba poskytla více dat, než se očekávalo.
Objekt neexistuje	16	Zadaný objekt v zařízení neexistuje.
Neprobíhá sekvence fragmentace služeb	17	Fragmentační sekvence pro tuto službu není pro tato data aktuálně aktivní.
Žádná uložená data atributů	18	Data atributů tohoto objektu nebyla uložena.
Selhání operace úložiště	19	Data atributů tohoto objektu nebyla uložena kvůli selhání během ukládání.
Selhání směrování-paket požadavku je příliš velký	1A	Paket požadavku na službu byl příliš velký pro přenos v síti. Směrovací zařízení bylo nuceno službu zrušit.
Selhání směrování-paket odpovědi je příliš velký	1B	Paket servisní odpovědi byl příliš velký pro přenos v síti. Směrovací zařízení bylo nuceno službu zrušit.

Název statusu	Kód statusu [0000_H]	Definice statusu
Chybějící data položky seznamu atributů	1C	Služba neposkytla hodnotu atribut v seznamu atributů, které služba potřebovala k provedení požadovaného chování.
Neplatný seznam hodnot atributů	1D	Služba vrací seznam atributů dodávaných se stavovými informacemi pro ty atributy, které byly neplatné.
Chyba vnitřní služby	1E	Servisní služba vyústila v chybu.
Specifická chyba dodavatele	1F	Pole doplňkový kód chybové odpovědi definuje konkrétní zjištěnou chybu. Použití tohoto obecného kódu chyby by mělo být provedeno pouze tehdy, pokud žádný z kódů chyb uvedených v této tabulce nebo v definici třídy objektu neodráží chybu přesně.
Neplatný parametr	20	Parametr spojený s požadavkem byl neplatný. Tento kód se používá, pokud parametr nesplňuje požadavky této specifikace nebo požadavky definované ve specifikaci aplikačního objektu.
Hodnota jednorázového zápisu nebo již zapsané médium	21	Byl proveden pokus o zápis na zapisovatelné médium (např. Jednotka WORM, PROM), na které již bylo zapsáno, nebo o změnu hodnoty, kterou nelze již provést.
Přijata neplatná odpověď	22	Je přijata neplatná odpověď (např. Kód služby odpovědi neodpovídá kódu služby požadavku nebo odpověď je kratší než minimální očekávaná velikost odpovědi). Tento stavový kód může sloužit pro jiné příčiny neplatných odpovědí.
	23–24	Rezervováno CIP pro budoucí rozšíření

Název statusu	Kód statusu $[0000_H]$	Definice statusu
Selhání klíče	25	Klíčový segment, který byl zahrnut jako první segment do cesty, se neshoduje s cílovým modulem. Stav specifický pro objekt musí označovat, která část kontroly klíče selhala.
Neplatná velikost cesty	26	Velikost cesty, která byla odeslána s požadavkem na službu, není dostatečně velká, aby umožnila směrování požadavku k objektu, nebo bylo zahrnuto příliš mnoho směrovacích dat.
Neočekávaný atribut v seznamu	27	Byl proveden pokus o nastavení atributu, který nelze v tuto chvíli nastavit.
Neplatné ID člena	28	ID člena uvedené v žádosti v zadané třídě / instanci / atributu neexistuje.
Člen není nastavitelný	29	Byl obdržen požadavek na změnu nemodifikovatelného člena.
Obecná chyba na straně serveru	2A	Tento kód chyby může být hlášeny pouze servery DeviceNet 2. skupiny pouze s 4K nebo méně kódovým prostorem a služba není podporována, atribut není podporován a atribut není nastavitelný.
	2B–CF	Vyhrazeno pro budoucí rozšíření.
Vyhrazeno pro třídu objektů a servisní chyby	D0–FF	Tento rozsah kódů chyb se používá k označení chyb specifických pro třídu objektů. Použití tohoto rozsahu by mělo být provedeno pouze tehdy, pokud žádný z chybových kódů uvedených v této tabulce přesně neodráží chybu, ke které došlo.

F Zdrojové kódy simulace

Výpis F.1: Knihovna rpscada.py

```
import json
import os
import socket
import locale
from dialog import Dialog
import time
from datetime import datetime
from datetime import timedelta
from threading import Thread
from cpppo.server.enip import client
from time import sleep
import logging
import sys
import threading
import cpppo
import runpy
logging.basicConfig( **cpppo.log_cfg )
from cpppo.server.enip import poll
import cpppo.server.enip.client
from mollyab import RPI_MACHINE as device
locale.setlocale(locale.LC_ALL, '')
import runpy
from pprint import pprint

setgetLock = threading.Lock()
lck = threading.Lock()
loglck = threading.Lock()
import sys
import warnings
if not sys.warnoptions:
        warnings.filterwarnings('ignore',
                                category=RuntimeWarning,
                                module='runpy')



globalParametters={
    "turbina1"                      : {"type":"BOOL","size":"10"},
    "otackomer2"           : {"type":"DINT","size":"1000"},
    "tlak2"                     : {"type":"REAL","size":"1000"},
    "tlak1"                   : {"type":"REAL","size":"1000"},
    "mereni_vodivosti1"   : {"type":"DINT","size":"1000"},
```

```
    "u_armatura4"                  : {"type":"BOOL","size":"10"},
    "u_armatura3"                  : {"type":"BOOL","size":"10"},
    "u_armatura2"                  : {"type":"BOOL","size":"10"},
    "hladina5"                      : {"type":"BOOL","size":"10"},
    "u_armatura1"                  : {"type":"BOOL","size":"10"},
    "hladina4"                      : {"type":"BOOL","size":"10"},
    "hladina2"                    : {"type":"BOOL","size":"10"},
    "hladina1"                   : {"type":"BOOL","size":"10"},
    "hladina3"                    : {"type":"BOOL","size":"10"},
    "r_armatura1"                   : {"type":"DINT","size":"1000"},
    "cerpadlo1"                   : {"type":"BOOL","size":"10"},
    "teplota1"                      : {"type":"DINT","size":"1000"},
    "otackomer1"                   : {"type":"DINT","size":"1000"}
}


class Gui():
    def __init__(self,title):
        self.d = Dialog(dialog="dialog")
        self.d.set_background_title(title)

    def MainMenu(self,title=""):
        menu = [("(0)","Find Devices"),
                        ("(1)","Load Devices"),
                        ("(2)","Run Scenario"),
                        ("(3)","Run speed test")
                        ]
        code, tag = self.d.menu(title, choices=menu)

        if code == self.d.OK:
            return tag
        else:
            return None

    def ScenariosMenu(self,title=""):
        scenarios = os.listdir("./scenarios")
        tmp_scenarios = []
        for scenario in scenarios:
            if scenario.startswith("SC"):
                tmp_scenarios.append(scenario)
        scenarios = tmp_scenarios
        scenarios = [("{}".format(i),
             "{}".format(scenario)) for i,scenario in enumerate(scenarios) ]
        code, tag = self.d.menu(title,
                                choices= [(a,b[2:]) for a,b in scenarios])

        if code == self.d.OK:
```

```
90              return scenarios[int(tag)]
91          else:
92              return None
93
94      def input(self,title="",init=""):
95          code, string = self.d.inputbox(text=title,init=init)
96
97          if code == self.d.OK:
98              return string
99          else:
100             return None
101
102     def SpeedMenu(self,title=""):
103         m = Master()
104         ip = self.input("IP adresa:","localhost")
105         if not ip:
106             return None
107         tag = self.input("tag","otackomer2")
108         if not tag:
109             return None
110         count = self.input("count","1000")
111         if not count:
112             return None
113         def count_to_int(count):
114             try:
115                 count = int(count)
116                 if count <= 0:
117                     raise ValueError("count is lower then zero")
118             except ValueError as e:
119                 if "count is lower then zero" in str(e):
120                     self.msg("count is lower then one")
121                     return count_to_int(self.input("count","1000"))
122                 else:
123                     self.msg("count is not number")
124                     return count_to_int(self.input("count","1000"))
125             except TypeError as e:
126                 return None
127             return count
128
129         count = count_to_int(count)
130         self.info("test speed running ...")
131         count = int(count)
132         try:
133             measure = m.speedTest(ip,tag,count)
134         except ConnectionRefusedError:
135             self.msg("can't reach slave")
136             return None
```

```
137             except Exception:
138                 self.msg("something is wrong with speedtest try it again later")
139                 return None
140             average = sum(measure) / len(measure)
141             self.msg("{} microseconds\n average from {} repeats of tag {} to
                ↪  slave {}".format(average, count,tag,ip))
142
143
144
145
146     def progressBox(self, filepath):
147         excode = self.d.tailbox(filepath)
148         return excode
149
150
151
152
153     def msg(self,text):
154         self.d.msgbox(text)
155
156     def info(self,text):
157         self.d.infobox(text)
158
159     def discoveryList(self,devices={},title=""):
160         localdevices=[]
161         for i, key in enumerate(list(devices.keys())):
162             localdevices.append(("{}".format(key),
163                                  "{} ({})".format(key,devices[key]['ip'])))
164
165         code, tag = self.d.menu(title, choices=localdevices)
166         if code == self.d.OK:
167             return devices[tag], tag
168         else:
169             return None, None
170
171
172     def setValues(self,device,title="",backtitle=""):
173         blacklist=["product_name","tcpip","identity"]
174         values = device['values']
175         for black in blacklist:
176             values.pop(black, None)
177
178         values = [("{}".format(key),"",values[key]) for key in
                ↪  list(values.keys())]
179
180         code, tags = self.d.checklist(title,
181                                       choices=values,
```

```
                            title="",
                            backtitle=backtitle)
        if code == self.d.OK:
            newdevice = device
            for key in list(newdevice['values'].keys()):
                newdevice['values'][key] = False
            for key in tags:
                newdevice['values'][key] = True

            return newdevice
        else:
            return None

class Master():

    def __init__(self,
                 devicesFile="data/devices.db",
                 dataFile="data/data.db"):
        self.devicesFile = devicesFile
        self.dataFile = dataFile
        self.db={}

    def updateValues(self,params,hostname="localhost",name=None,port=44818):
        if not name:
            name=hostname

        def failure( exc ):
            failure.string.append( str(exc) )
        failure.string                = []

        def process( par, val ):
            process.values[par]        = val


        process.done                = False
        process.values                = {}
        poller                        = threading.Thread(
            target=poll.poll, kwargs={
                'proxy_class':  device,
                'address':      (hostname, port),
                'cycle':        1,
                'timeout':      1,
                'process':      process,
                'failure':      failure,
                'params':       params,
            })
        poller.start()
```

```
229         def readValues():
230             try:
231                 while True:
232                     while process.values:
233                         par,val                 = process.values.popitem()
234                         if val:
235 
236 
237                             lck.acquire()
238                             try:
239                                 with open(self.dataFile,"r") as f:
240                                     try:
241                                         devices=json.load(f)
242                                     except json.decoder.JSONDecodeError:
243                                         devices={}
244                             except FileNotFoundError:
245                                 open(self.dataFile, 'a').close()
246                                 devices={}
247 
248                             with open(self.dataFile,"w") as f:
249                                 values=devices.get(name,{})
250                                 if device.PARAMETERS[par][1]=="BOOL":
251                                     values[par]= True if val[0] == 1 else
                                    ↪  False
252                                 else:
253                                     values[par]=val[0]
254                                 devices[name]=values
255                                 f.write(json.dumps(devices,
256                                                    sort_keys=True,
257                                                    indent=4))
258                             lck.release()
259 
260 
261 
262 
263                     while failure.string:
264                         exc                 = failure.string.pop( 0 )
265 
266 
267                     time.sleep( .0001 )
268             finally:
269                 process.done                = True
270                 poller.join()
271 
272         t = Thread(target=readValues,args=())
273         t.start()
274 
```

```
275     def loadDB(self):
276         try:
277             with open(self.devicesFile, 'r+') as f:
278                 try:
279                     self.db = json.load(f)
280                 except json.decoder.JSONDecodeError:
281                         f.seek(0)
282                         f.write("{}")
283                         f.truncate()
284                         self.db={}
285         except FileNotFoundError:
286             open(self.devicesFile, 'a').close()
287             self.loadDB()
288 
289     def saveDB(self):
290         with open(self.devicesFile, 'w') as outfile:
291             json.dump(self.db, outfile, sort_keys=True, indent=4)
292 
293     def waitForDevs_async(self,time=5):
294         T = Thread(target=self.waitForDevs,args=(time,))
295         T.start()
296 
297     def waitForDevs(self, time=5,gui=None):
298         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
299             sock.settimeout(1)
300             endTime = datetime.now() + timedelta(seconds=time)
301             server_address = ('', 5620)
302             print('starting up on {} port {}'.format(*server_address))
303             sock.bind(server_address)
304             clients = ["Finding devices ..."]
305             gui.info("\n".join(clients))
306             while datetime.now() < endTime:
307                 try:
308                     data, address = sock.recvfrom(1024)
309                     data = json.loads(str(data, 'utf-8'))
310                     values ={}
311                     for value in data["values"]:
312                         values[value]=False
313                     self.db[data['hostname']]={"values":values,
314                                                "ip":address[0]}
315                     if gui is not None:
316                         clients.append("{}) {}".format(len(clients),
317                                                        data['hostname']))
318                         gui.info("\n".join(clients))
319                 except:
320                     continue
321 
```

```
322             return self.db if self.db else None
323
324     def broadcast(self):
325         with socket.socket(socket.AF_INET,
326                            socket.SOCK_DGRAM,
327                            socket.IPPROTO_UDP) as s:
328             s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
329             s.sendto(b'RPSCADA-DISCOVERY', ('255.255.255.255', 5621))
330
331
332     def sendValue(self,hostname,tag, value,debug=False):
333         def write(host,tag="",value="",debug=False):
334             with client.connector( host=host,port=44818 ) as conn:
335                 attr=9
336                 operation = ["{}[{}]=({}){} ".format(tag,
337                                      attr,
338                                      globalParametters[tag]["type"],value)]
339                 for idx,dsc,req,rpy,sts,val in conn.pipeline(
340                             operations=client.parse_operations(operation ,
341                             route_path=[]),
342                             timeout=1):
343                     tag = req["path"]["segment"][0]["symbolic"]
344                     return sts
345         setgetLock.acquire()
346         result = write(hostname,tag,value)
347         setgetLock.release()
348         return result
349
350     def getValue(self,hostname,tag,debug=False):
351         def read(host,tag="",debug=False):
352             with client.connector( host=host,port=44818 ) as conn:
353                 attr=9
354                 operations = client.parse_operations(["{}[{}]".format(tag,
355                                                                       attr)],
356                                                      route_path=[])
357
358                 for idx,dsc,req,rpy,sts,val in conn.pipeline(
359                                                  operations=operations,
360                                                  multiple=20,
361                                                  depth=20,
362                                                  timeout=1):
363                     tag = req["path"]["segment"][0]["symbolic"]
364                     ret = val[0]
365                     if globalParametters[tag]["type"] == "BOOL":
366                         ret = True if ret==1 else False
367                     return ret
368         setgetLock.acquire()
```

```
369         result = read(hostname,tag,debug)
370         setgetLock.release()
371         return result
372 
373     def speedTest(self,hostname,tag,count=1000):
374         speeds=[]
375         for i in range(count):
376             t1 = datetime.utcnow()
377             self.sendValue(hostname,tag,20)
378             t2 = datetime.utcnow()
379             t3 = t2 -t1
380 
381             speeds.append(t3.microseconds)
382         return speeds
383 
384 
385 
386 
387 
388 class Slave():
389     def __init__(self, values=[]):
390         self.values = values
391         self.hostname = socket.gethostname()
392 
393     def waitForDevs(self):
394         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
395             server_address = ('', 5621)
396             print('starting up on {} port {}'.format(*server_address))
397             sock.bind(server_address)
398             while True:
399                 data, address = sock.recvfrom(1024)
400                 print(data,address)
401                 data = str(data, 'utf-8')
402                 if str(data)=="RPSCADA-DISCOVERY":
403                     sock.close()
404                     return address[0]
405 
406     def sendValues(self, ip):
407         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
408             forSend = {"hostname":self.hostname,"values":self.values}
409             forSend = str(json.dumps(forSend))
410             sent = sock.sendto(forSend.encode(), (ip, 5620))
411 
412 class Scenario():
413     def __init__(self, master=None):
414         self.master=master
415 
```

```
416     def print(self, text):
417
418
419         loglck.acquire()
420         with open("logfile","a") as l:
421                     l.write("{}\n".format(text))
422         loglck.release()
423
424     def sV(self,device,variable,value):
425         if self.master.sendValue(self.master.db[device]["ip"],
426                                  variable,value,
427                                  debug=True)==0:
428             self.print("{} is set to {}\n".format(variable,value))
429
430
431     def gV(self,device,variable):
432         return self.master.getValue(self.master.db[device]["ip"],
433                                     tag=variable,
434                                     debug=True)
435
436
437     def gVf(self,device,variable):
438
439
440         lck.acquire()
441         try:
442             with open(self.master.dataFile,"r") as f:
443                 try:
444                     devices=json.load(f)
445                 except json.decoder.JSONDecodeError:
446                     devices={}
447         except FileNotFoundError:
448             open(self.master.dataFile, 'a').close()
449             devices={}
450
451         fdevice=devices.get(device,None)
452         if fdevice:
453             value=fdevice.get(variable,None)
454
455         else:
456             value=None
457
458         if value is None:
459             self.print("{}/{} not found in database trying get it again in 1
                ↪  second".format(device,variable))
460             lck.release()
461             sleep(1)
```

```
462             return self.gV(device,variable)
463
464         lck.release()
465
466         return value
467
468
469 if __name__=="__main__":
470     m = Master()
471     m.sendCmd("192.168.1.51","otackomer1",0,debug=True)
```

Výpis F.2: Skript master.py

```
1  import rpscada
2  import sys
3  import importlib
4  from threading import Thread
5  import time
6
7  def findMenu(devices):
8      device, name = gui.discoveryList(devices,title="Vyberte zařízení:")
9      if device is not None:
10         device = gui.setValues(device,title="vyberte hodnoty:")
11         if device is not None:
12             devices[name]=device
13             master.db = devices
14             master.saveDB()
15             findMenu(devices)
16         else:
17             findMenu(devices)
18
19 def mainMenu():
20     code = gui.MainMenu("Menu")
21     while code is not None:
22         if code == "(0)":
23             master.broadcast()
24             devices = master.waitForDevs(gui = gui)
25             if devices:
26                 findMenu(devices)
27             else:
28                 gui.msg("no devices Found")
29
30         elif code == "(1)":
31             master.loadDB()
32             devices = master.db
```

```
33              if devices:
34                  findMenu(devices)
35              else:
36                  gui.msg("Local database is empty")
37
38          elif code == "(2)":
39              scenario_name = gui.ScenariosMenu("Vyberte scénář:")
40              if scenario_name:
41                  scenario_name = str(scenario_name[1])[:-3]
42                  scenario = importlib.import_module('scenarios')
43                  master.loadDB()
44                  devices = master.db
45                  with open("logfile","w") as f:
46                      f.write("")
47
48                  T = Thread(target=scenario.run_scenario,args=(scenario_name,
49                                                                master,))
50                  T.start()
51                  gui.progressBox("logfile")
52
53          elif code == "(3)":
54              gui.SpeedMenu("SpeedMenu")
55
56          code = gui.MainMenu("Menu")
57
58  if __name__ == "__main__":
59      master = rpscada.Master()
60      gui = rpscada.Gui("CIP simulace")
61      mainMenu()
62
63
```

Výpis F.3: Skript slave.py

```
1  import rpscada
2  import cpppo
3  from mollyab import RPI_MACHINE
4  from threading import Thread
5  import time
6  import os
7  from cpppo.server.enip import client
8  from pprint import pprint
9  import datetime
10 from pprint import pprint
11 import sys
```

```
12
13
14  parametters={
15      "turbina1"          : {"type":"BOOL","size":"10","default":"false"},
16      "otackomer2"        : {"type":"DINT","size":"1000","default":"0"},
17      "tlak2"             : {"type":"REAL","size":"1000","default":"0"},
18      "tlak1"             : {"type":"REAL","size":"1000","default":"0.4"},
19      "mereni_vodivosti1" : {"type":"DINT","size":"1000","default":"330"},
20      "u_armatura4"       : {"type":"BOOL","size":"10","default":"false"},
21      "u_armatura3"       : {"type":"BOOL","size":"10","default":"false"},
22      "u_armatura2"       : {"type":"BOOL","size":"10","default":"false"},
23      "hladina5"          : {"type":"BOOL","size":"10","default":"false"},
24      "u_armatura1"       : {"type":"BOOL","size":"10","default":"false"},
25      "hladina4"          : {"type":"BOOL","size":"10","default":"false"},
26      "hladina2"          : {"type":"BOOL","size":"10","default":"true"},
27      "hladina1"          : {"type":"BOOL","size":"10","default":"false"},
28      "hladina3"          : {"type":"BOOL","size":"10","default":"false"},
29      "r_armatura1"       : {"type":"DINT","size":"1000","default":"0"},
30      "cerpadlo1"         : {"type":"BOOL","size":"10","default":"false"},
31      "teplota1"          : {"type":"DINT","size":"1000","default":"55"},
32      "otackomer1"        : {"type":"DINT","size":"1000","default":"2950"}
33  }
34
35
36
37
38  def discovery():
39      while True:
40          addr = slave.waitForDevs()
41          print("found master at ",addr)
42          slave.sendValues(addr)
43          time.sleep(0.1)
44
45  def read(host,tag="",debug=False):
46      with client.connector( host=host,port=44818 ) as conn:
47          attr=9
48          results={}
49          operations = client.parse_operations( [tag] ,route_path=[])
50          for idx,dsc,req,rpy,sts,val in conn.pipeline(operations=operations,
51                                                       multiple=20,
52                                                       depth=20,
53                                                       timeout=1):
54              tag = req["path"]["segment"][0]["symbolic"]
55              ret = val[0]
56              if parametters[tag]["type"] == "BOOL":
57                  ret = True if ret==1 else False
58              results[req["path"]["segment"][0]["symbolic"]] = ret
```

```
59
60      return results
61
62  def write(host,tag="",value="",debug=False):
63      global parametters
64      with client.connector( host=host,port=44818 ) as conn:
65          attr=9
66          results={}
67          operation = ["{}[{}]=({}){} ".format(tag,attr,
68                                               parametters[tag]["type"],
69                                               value)]
70          for idx,dsc,req,rpy,sts,val in conn.pipeline(
71                          operations=client.parse_operations( operation ,
72                                                              route_path=[]),
73                                                              timeout=1):
74              tag = req["path"]["segment"][0]["symbolic"]
75
76
77  def readAll(host,debug=False):
78      with client.connector( host=host,port=44818 ) as conn:
79          attr=9
80          results={}
81          params= ["{}[{}]".format(key,attr) for key in parametters.keys()]
82          operations = client.parse_operations( params ,route_path=[])
83          for idx,dsc,req,rpy,sts,val in conn.pipeline(operations=operations,
84                                                       multiple=20,
85                                                       depth=20,
86                                                       timeout=1):
87              tag = req["path"]["segment"][0]["symbolic"]
88              ret = val[0]
89              if parametters[tag]["type"] == "BOOL":
90                  ret = True if ret==1 else False
91              results[req["path"]["segment"][0]["symbolic"]] = ret
92
93      return results
94
95
96  def changer():
97      while(True):
98          try:
99              inputed = input("zadej parametr=hodnota: ").split("=")
100             param = inputed[0]
101             val = inputed[1]
102             write("127.0.0.1",param,val)
103         except Exception:
104             continue
105
```

```
106
107 def info():
108     while True:
109         pprint(readAll("127.0.0.1",debug=False))
110         time.sleep(10)
111
112 def define():
113     for param in parametters.keys():
114         write("127.0.0.1",param,parametters[param]["default"])
115
116
117
118 def runServer():
119     global parametters
120     params = " ".join(["{}={}[{}]".format(key,
121                                          parametters[key]['type'],
122                     parametters[key]['size']) for key in parametters.keys()])
123     os.system("python3 RPIsimulator.py {}".format(params))
124
125 slave = rpscada.Slave()
126 slave.values=list(parametters.keys())
127
128 discoveryServerThread=Thread(target=discovery)
129 runServerThread=Thread(target=runServer)
130 changerThread=Thread(target=changer)
131 infoThread=Thread(target=info)
132
133
134 if __name__ == "__main__":
135     runServerThread.start()
136     discoveryServerThread.start()
137     time.sleep(5)
138     define()
139
140     chT = changerThread
141     chT.start()
142     chT.join()
143
144
145
146
147
```

Výpis F.4: Skript RPIsimulator.py

```
from __future__ import absolute_import, print_function
from __future__ import division, unicode_literals
try:
    from future_builtins import zip, map
except ImportError:
    pass
import json
import ast
import errno
import logging
import os
import re
import socket
import sys
import time
import threading

import pytest

if __name__ == "__main__":
    if __package__ is None:
        __package__        = "cpppo.server.enip"
    sys.path.insert(0, os.path.dirname(os.path.dirname(
                                        os.path.dirname(
                                        os.path.dirname(
                                        os.path.abspath(__file__))))))
    from cpppo.automata import log_cfg



from cpppo.dotdict import dotdict
from cpppo.misc import timer, near
from cpppo.modbus_test import nonblocking_command
from cpppo.server import enip, network
from cpppo.server.enip import poll
from mollyab import powerflex, RPI_MACHINE


class UCMM_no_route_path( enip.UCMM ):
    route_path                      = False

class DPI_Parameters( enip.Object ):
    def __init__( self, name=None,  **kwds ):
        super( DPI_Parameters, self ).__init__( name=name,**kwds )

def main( **kwds ):
```

```
47      enip.config_files                   += [ __file__.replace( '.py', '.cfg' ) ]
48      return enip.main( argv=sys.argv[1:], UCMM_class=UCMM_no_route_path )
49
50  if __name__ == "__main__":
51      sys.exit( main() )
```

Výpis F.5: Scénář SCregulaceTurbiny.py

```
1   from threading import Thread
2   from time import sleep
3   import json
4   from rpscada import Scenario as SC
5   from rpscada import Master
6   import time
7
8   class Scenario():
9       def __init__(self,master):
10          self.s = SC(master)
11          self.done=False
12          self.t1 = Thread(name="vetev_turbina", target=self.vetev_turbina,
13                                                  args=(self.s,))
14
15      def start(self):
16          self.t1.start()
17
18      def vetev_turbina(self,s):
19          while not self.done:
20              if s.gV("client51","turbina1") is False:
21                  while s.gV("client51","turbina1") is False:
22                      time.sleep(0.1)
23                  s.print("Turbina najeta")
24
25
26
27
28              hodnota_otacek = s.gV("client51","otackomer2")
29              if hodnota_otacek == 8000:
30                  self.vetev_turbina(s)
31              else:
32                  if hodnota_otacek < 8000:
33                      while s.gV("client51","otackomer2")<8000:
34                          s.sV("client51","otackomer2",s.gV("client51",
35                                                          "otackomer2")+100)
36                  else:
37                      while s.gV("client51","otackomer2")>8000:
```

```
                        s.sV("client51","otackomer2",s.gV("client51",
                                                          "otackomer2")-100)


if __name__ == "__main__":
    master = Master()
    scenario = Scenario(master)
    scenario.start()
```

Výpis F.6: Scénář SCprecerpaniOdpadniVody.py

```
from threading import Thread
from time import sleep
import json
from rpscada import Scenario as SC
from rpscada import Master

class Scenario():
    def __init__(self,master):
        self.s = SC(master)

        self.t1 = Thread(name="vetev_provoz",target=self.vetev_provoz,
                                                          args=(self.s,))
        self.t2 = Thread(name="vetev_otacky",target=self.vetev_otacky,
                                                          args=(self.s,))
        self.t3 = Thread(name="vetev_teplota",target=self.vetev_teplota,
                                                          args=(self.s,))
        self.t4 = Thread(name="vetev_tlak",target=self.vetev_tlak,
                                                          args=(self.s,))
        self.t5 = Thread(name="vetev_vodivost",target=self.vetev_vodivost,
                                                          args=(self.s,))
        self.t6 = Thread(name="vetev_cilova_nadrz",
                                              target=self.vetev_cilova_nadrz,
                                              args=(self.s,))
        self.client = ["CreapyMachine","DESKTOP-4U2AS9R","client51"]
        self.client = self.client[0]

    def start(self):

        while not self.s.gV(self.client,"hladina1"):
            sleep(0.5)

        self.s.sV(self.client,"cerpadlo1",True)
        if self.s.gV(self.client,"cerpadlo1"):
            self.t1.start()
```

```
35              self.t2.start()
36              self.t3.start()
37              self.t4.start()
38              self.t5.start()
39              self.t6.start()
40
41
42
43      def vetev_provoz(self,s):
44          s.print("Zapnulo se cerpadlo.")
45          s.sV(self.client,"u_armatura1",True)
46          s.sV(self.client,"r_armatura1",20)
47          while s.gV(self.client,"hladina2"):
48              sleep(0.5)
49              continue
50          s.sV(self.client,"r_armatura1",0)
51          s.sV(self.client,"u_armatura1",False)
52          s.sV(self.client,"cerpadlo1",False)
53          s.print("Vypnulo se cerpadlo.")
54
55
56      def vetev_otacky(self,s):
57          if s.gV(self.client,"cerpadlo1")==True:
58              if s.gV(self.client,"otackomer1")<2950:
59                  if not s.gV(self.client,"hladina2"):
60                      s.sV(self.client,"otackomer1",2950)
61                      self.vetev_otacky(s)
62              elif s.gV(self.client,"otackomer1")>2950:
63                  s.sV(self.client,"otackomer1",2950)
64                  self.vetev_otacky(s)
65              elif s.gV(self.client,"otackomer1")==2950:
66                  self.vetev_otacky(s)
67
68      def vetev_teplota(self,s):
69          cerpadlo=True
70          while s.gV(self.client,"teplota1")<=60:
71              sleep(0.5)
72              if s.gV(self.client,"cerpadlo1"):
73                  continue
74              else:
75                  cerpadlo=False
76
77          if cerpadlo:
78              while 60 < s.gV(self.client,"teplota1"):
79                  sleep(0.2)
80                  s.sV(self.client,"r_armatura1",s.gV(self.client,
81                                                      "r_armatura1")+10)
```

```
82                 if not cerpadlo: break
83                 if s.gV(self.client,"r_armatura1")==100:
84                     s.sV(self.client,"cerpadlo1",False)
85                     cerpadlo=False
86                     s.print("Vysoka teplota na cerpadle!")
87                     break
88                 elif s.gV(self.client,"r_armatura1")<100:
89                     if s.gV(self.client,"teplota1") > 80:
90                         s.sV(self.client,"cerpadlo1",False)
91                         cerpadlo=False
92                         s.print("Vysoka teplota na cerpadle!")
93                         break
94             if cerpadlo:
95                 self.vetev_teplota(s)
96
97
98
99
100    def vetev_tlak(self,s):
101        cerpadlo = True
102        while s.gV(self.client,"tlak2") <= (s.gV(self.client,
103                                                 "tlak1") + 0.98):
104            sleep(0.5)
105            if s.gV(self.client,"cerpadlo1"):
106                continue
107            else:
108                cerpadlo = False
109                break
110        if cerpadlo:
111            s.sV(self.client,"u_armatura1",False)
112            s.sV(self.client,"cerpadlo1",False)
113            s.print("Vysoky rozdil tlaku!")
114
115
116    def vetev_vodivost(self,s):
117        cerpadlo = True
118        while s.gV(self.client,"mereni_vodivosti1") <= 330 :
119            sleep(0.5)
120            if s.gV(self.client,"cerpadlo1"):
121                continue
122            else:
123                cerpadlo = False
124                break
125
126        if cerpadlo:
127            s.sV(self.client,"u_armatura1",False)
128            s.sV(self.client,"cerpadlo1",False)
```

```
129             s.print("Vysoka vodivost!")
130
131
132     def vetev_cilova_nadrz(self,s):
133         if not s.gV(self.client,"hladina3"):
134             if not s.gV(self.client,"u_armatura2"):
135                 s.sV(self.client,"u_armatura2",True)
136             if s.gV(self.client,"cerpadlo1"):
137                 self.vetev_cilova_nadrz(s)
138         else:
139             s.sV(self.client,"u_armatura2",False)
140             def hladina4_r():
141                 if not s.gV(self.client,"hladina4"):
142                     if not s.gV(self.client,"u_armatura3"):
143                         s.sV(self.client,"u_armatura3",True)
144                     if s.gV(self.client,"cerpadlo1"):
145                         hladina4_r()
146                 else:
147                     s.sV(self.client,"u_armatura3",False)
148                     def hladina5_r():
149                         if not s.gV(self.client,"hladina5"):
150                             if not s.gV(self.client,"u_armatura4"):
151                                 s.sV(self.client,"u_armatura4",True)
152                             if s.gV(self.client,"cerpadlo1"):
153                                 hladina5_r()
154                         else:
155                             s.print("Plne nadrze!")
156                     hladina5_r()
157
158             hladina4_r()
159
160
161
162 if __name__ == "__main__":
163     master = Master()
164     scenario = Scenario(master)
165     scenario.start()
```

Výpis F.7: Soubor _init_.py

```
1 from . import SCprecerpaniOdpadniVody
2 from . import SCregulaceTurbiny
3
4
5
```

```
6
7  def run_scenario(scenario,master):
8      scenarios={
9      'SCprecerpaniOdpadniVody':SCprecerpaniOdpadniVody.Scenario(master),
10     'SCregulaceTurbiny':SCregulaceTurbiny.Scenario(master)
11     }
12     scenarios[str(scenario)].start()
13
14 def get_scenario_dependencies(scenario):
15     scenarios={
16     'SCprecerpaniOdpadniVody':["client51"],
17      'SCregulaceTurbiny':["client51"]
18     }
19     return scenarios[str(scenario)
```

G Obsah přiloženého CD

- / ..... kořenový adresář přiloženého CD
  - Dokumentace ..... dokumentace k diplomové práci
    - Diplomová_práce.zip
  - Program ..... program výsledné simulace
    - __pycache__
    - bin
    - data
    - scenarios
    - venv
    - logfile
    - master.py
    - mollyab.py
    - pyvenv.cfg
    - README.md
    - requirements.txt
    - RPIsimulator.py
    - rpscada.py
    - slave.py
    - testSlave.py
    - values.json
  - Příloha A ..... přehled předdefinovaných kódů
    - Přehled_předdefinovaných_kódů.pdf
  - Příloha B ..... konfigurace pro obousměrnou komunikaci (vyčítání z konzole)
    - ioclient.py
    - pollserver.py
    - START.sh
  - Příloha C ..... konfigurace pro obousměrnou komunikaci v reálném čase
    - controller.py
    - Výpis_na_konzoly_pro_sběr_informací.sh
    - Výpis_na_konzoly_řízeného_zařízení.sh
  - Příloha D ..... algoritmy scénářů
    - Algoritmus_scénáře_pro_regulaci_turbíny.pdf
    - Algoritmus_scénáře_pro_zpracování_odpadních_vod.pdf
  - Příloha E ..... přehled kódů položky status
    - Přehled_kódů_položky_status.pdf