Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

EtherNet/IP Softwarová emulace PLC

William Groman

Česká zemědělská univerzita v Praze
Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	William Groman
Studijní program:	Informatika
Vedoucí práce:	Ing. Dana Vynikarová, Ph.D.
Garantující pracoviště:	Katedra informačního inženýrství
Jazyk práce:	Čeština
Název práce:	EtherNet/IP softwarová emulace PLC
Název anglicky:	EtherNet/IP software emulation of PLC
Cíle práce:	Cílem teoretické části bakalářské práce je popis tvorby software a ethernetové softwarové emulace PLC. Dílčím cílem je analýza možných technologií a nástrojů pro tvorbu EtherNet/IP softwaru. Na základě těchto analýz je cílem vybrat vhodné řešení pro tvorbu software. Cílem praktické části bakalářské práce je návrh a tvorba emulace PLC s funkcí EtherNet/IP za pomoci vybraných řešení a nástrojů. Vytvořený software bude následně řádně otestován.
Metodika:	Teoretická část bude vytvořena na základě studia odborné literatury, dokumentací včetně webových stránek souvisejících s používanými nástroji a průzkumu problematiky. V praktické části bude za pomoci průzkumů a analýz provedených v teoretické části bakalářské práce vytvořen software, který je schopný s EtherNet/IP zařízením navázat cyklickou komunikaci s různými cyklickými časy. Výstupem práce bude již zmíněný software, závěrem pak bude zhodnocení celého projektu.

Doporučený rozsah práce: 30-60 stran

Klíčová slova: PLC, emulace, software, Ethernet, IP, Python

Doporučené zdroje informací:

1. MARTIN, Robert C. Clean code: a handbook of agile software craftmanship. Upper Saddle River, NJ: Prentice Hall, c2009. Robert C. Martin series. ISBN 978-0132350884.

2. RINALDI, S. John a Perry MARSHALL. Industrial Ethernet. Third Edition. ISA, 2016. ISBN 978-1945541049.

3. RINALDI, S. John. EtherNet/IP: The Everyman's Guide to The Most Widely Used Manufacturing Protocol. 2018. ISBN 978-1726662567.

Předběžný termín obhajoby: 2022/23 LS - PEF

Elektronicky schváleno: 23. 11. 2021
Ing. Martin Pelikán, Ph.D.
Vedoucí katedry

Elektronicky schváleno: 29. 11. 2021
Ing. Martin Pelikán, Ph.D.
Děkan

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci “EtherNet/IP softwarová emulace PLC” jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Daně Vynikarové, Ph.D. a Ing. Alexanderu Zhigunovovi Ph.D. za cenné rady při tvorbě bakalářské práce.
Speciálně bych poté rád poděkoval Ing. Lukáši Kvardovi za dobré vedení a cennou pomoc při tvorbě softwaru.

EtherNet/IP softwarová emulace PLC

Abstrakt

Tato práce řešila softwarovou emulaci PLC za použití EtherNet/IP protokolu. Cílem této práce bylo zjistit, zda by nebylo možné vytvořit alternativu k hmotnému PLC podporující EtherNet/IP, které je, jak velmi složité získat, tak má řadu dalších nevýhod pro účel testování široce používaného průmyslového hardwaru. První z úkolů bylo jak seznámení se s PLC pro jeho přesné napodobení, tak s nástroji, které používáme a odůvodnit jejich použití. Druhým úkolem byla již samotná snaha o tvorbu emulace za využití těchto nástrojů. Dále popis jejich využití, popis komplexity problémů, které se vyskytly při tvorbě a jejich originálních řešení. V závěru bylo zjištěno, že je skutečně možné vytvořit softwarovou verzi PLC vytvořit napodobující přesné chování jeho hmotného podkladu. Také jsou popsány aktuální testy softwarové verze stejně jako její limitace a plány do budoucna.

Klíčová slova: PLC, emulace, software, EtherNet/IP, Python

EtherNet/IP softwarová emulace PLC

Abstract

This work dealt with the software emulation of a PLC using EtherNet/IP protocol. The aim of this work was to find out whether it would be possible to create an alternative to a physical PLC supporting EtherNet/IP, which, as it is very difficult to obtain, has a number of other disadvantages for the purpose of testing widely used industrial hardware. The first task was to familiarize ourselves with the PLC for its exact imitation, as well as with the tools we use and to justify their use. The second task was the very effort to create an emulation using these tools. Furthermore, a description of their use, a description of the complexity of the problems that occurred during creation and their original solutions. In conclusion, it was found that it is indeed possible to create a software version of a PLC that mimics the exact behaviour of its material base. Current tests of the software version as well as its limitations and plans for the future are also described.

Keywords: PLC, emulation, software, EtherNet/IP, Python

Obsah

Seznam obrázků

Seznam tabulek

2. Úvod

V této práci bude popsán proces tvorby softwarové emulace PLC pro testování kompatibility široce používaného průmyslového zařízení, za účelem zajistit jejich bezchybný běh za jakýchkoli podmínek. PLC je řídící jednotka používaná v automatické a důvodem snahy o jeho emulaci je nemožnost torby testů pro případy, jako například zaslání špatného packetu do zařízení. Vyjma tohoto důvod jsou hmotná PLC velmi drahá a těžko dostupná v mnoha případech s čekacími lhůtami až půl roku. Další nevýhoda je, že v některých případech není možná bez manuální konfigurace změna projektu za běhu používaného PLC. Všechny tyto problémy bude softwarová emulace jednoduše řešit. Postupem práce bude seznámit se s PLC, dále vyhodnotit nejlepší nástroje pro tvorbu jeho emulace a odůvodnit proč tyto nástroje byly vybrány. Poté již samotná tvorba softwaru a její popis.

3. Cíl práce a metodika

3.1. Cíl práce

Cílem teoretické části bakalářské práce je popis tvorby software a ethernetové softwarové emulace PLC. Dílčím cílem je analýza možných technologií a nástrojů pro tvorbu EtherNet/IP softwaru. Na základě těchto analýz je cílem vybrat vhodné řešení pro tvorbu software.

Cílem praktické části bakalářské práce je návrh a tvorba emulace PLC s funkcí EtherNet/IP za pomoci vybraných řešení a nástrojů. Vytvořený software bude následně řádně otestován.

3.2. Metodika

Teoretická část bude vytvořena na základě studia odborné literatury, dokumentací včetně webových stránek souvisejících s používanými nástroji a průzkumu problematiky.

V praktické části bude za pomoci průzkumů a analýz provedených v teoretické části bakalářské práce vytvořen software, který je schopný s EtherNet/IP zařízením navázat cyklickou komunikaci s různými cyklickými časy.

Výstupem práce bude již zmíněný software, závěrem pak bude zhodnocení celého projektu.

4. Teoretická východiska

4.1. PLC

Jelikož bude tvořena emulace PLC je nutné zjisti základní informace o tomto hardwaru jako proč byl vytvořen, co PLC je, jak se s ním zachází a jaký je jeho účel v automatizované výrobě, pro účel jeho co nejpřesnějšího napodobení.

4.1.1. Co je PLC

[1] Programovatelný logický kontrolér (PLC) nebo programovatelný kontrolér je průmyslový počítač, který byl přizpůsobený a zrobustněný pro řízení výrobních procesů, jako jsou montážní linky, stroje, robotická zařízení nebo jakákoli činnost vyžadující vysokou spolehlivost, snadné programování a diagnostiky chyb procesu.

PLC existují ve formách od malých modulárních zařízení s desítkami vstupů a výstupů (I/O), v pouzdře integrovaném s procesorem, až po velká modulární zařízení montovaná do racku s tisíci I/O, která jsou často propojena s jinými PLC a SCADA systémy.

I/O (vstup/výstup)
[2] Vyslovované „eye-oh“, popisuje jakoukoli operaci, program nebo zařízení, které přenáší data do nebo z počítače.

Supervisory control and data acquisition (SCADA)
[3] Je architektura řídicího systému zahrnující počítače, síťovou datovou komunikaci a grafická uživatelská rozhraní pro dohled nad stroji a procesy na vysoké úrovni. Zahrnuje také senzory a další zařízení, jako jsou programovatelné logické automaty, které jsou propojeny s procesem nebo strojním zařízením.

[3]PLC mohou být zaměřené na různé účely, například pro ovládaní velkého množství digitálních a analogových I/O; rozšířené teplotní rozsahy; odolnost vůči elektrickému šumu. odolnost vůči vibracím a nárazům, anebo plnění všech těchto požadavku najednou. Programy pro řízení provozu stroje jsou obvykle uloženy v bateriově zálohované nebo energeticky nezávislé paměti.

[3]PLC je příkladem systému tvrdého reálného času, protože výstupní výsledky musí být vytvořeny v reakci na vstupní podmínky v omezeném čase, jinak dojde k nezamýšlené operaci.

4.1.2. Historie PLC

[4] PLC byly poprvé vyvinuty v automobilovém průmyslu, aby poskytovaly flexibilní, odolné a snadno programovatelné řídicí jednotky, které nahradily pevně zapojené reléové logické

systémy. Od té doby byly široce přijímány jako vysoce spolehlivé automatizační řídicí jednotky vhodné pro drsná prostředí.

[5]Dick Morley je považován za otce PLC, protože vynalezl první PLC, Modicon 084, pro General Motors v roce 1968. Ochranná známka „Modicon“ pokračuje i v dnešní době.

Dříve se řídicí logika pro výrobu skládala hlavně z relé, vačkových časovačů, bubnových sekvencerů a vyhrazených ovladačů s uzavřenou smyčkou. Pevně propojená povaha ztěžovala konstruktérům změnit proces automatizace. Když vznikly počítače pro všeobecné použití, byly brzy použity pro řídicí logiku v průmyslových procesech. Tyto rané počítače byly nespolehlivé a vyžadovaly specializované programátory a přísnou kontrolu pracovních podmínek, jako je teplota, čistota a kvalita napájení.

[6]PLC vzniklo jako odpověď na tyto výzvy a omezeni. PLC poskytovalo několik výhod oproti dřívějším automatizačním systémům. Snášelo průmyslové prostředí lépe než počítače a bylo spolehlivější, kompaktnější a vyžadovalo méně údržby než reléové systémy. Bylo snadno rozšiřitelné o další I/O moduly, zatímco reléové systémy vyžadovaly komplikované hardwarové změny v případě rekonfigurace. To umožnilo snadnější opakování návrhu výrobního procesu. S jednoduchým programovacím jazykem zaměřeným na logiku a přepínací operace bylo uživatelsky přívětivější než počítače používající univerzální programovací jazyky. PLC umožnilo také sledování provozu jeho operací. Dřívější PLC byly naprogramovány v žebříkové logice, která silně připomínala schematický diagram reléové logiky.

4.1.3. PLC komunikace

[7] Komunikační protokol je způsob trans-příjmu dat se sadou pravidel, pro odesílání nebo přijímání mezi dvěma nebo více zařízeními. Komunikační protokol je médium nebo kanál mezi dvěma nebo více komunikujícími zařízeními. Pomocí komunikačních protokolů se mohou dvě zařízení propojit a komunikovat spolu.

[7]Různé typy dostupných komunikačních protokolů pomáhají rozšiřovat síť zařízení vzájemným propojením. Existuje řada starších komunikačních protokolů, které se stále používají v praxi a stále mají své místo, protože umožňují rychlé a snadné spojení mezi produkty jednoho dodavatele. Mnoho dodavatelů PLC stále podporuje tato starší fyzická, kabelová připojení

[7]Mezi nejrozšířenější komunikační protokoly používané pro automatizaci PLC procesů v dnešní době můžeme vyjmenovat: nejrozšířenější komunikační protokoly používané pro automatizaci PLC procesů v dnešní době můžeme vyjmenovat:

1. EtherNet/IP
2. profibus
3. modbus
4. Interbus
5. ProfiNet
6. ControlNet
7. DeviceNet
8. DirectNet
9. CompoNet

toto jsou nejpřednější komunikační protokoly používané pro PLC a další síťová připojení. Tyto protokoly jsou podporovány nejen různými PLC, ale také dalšími síťovém zařízeni. Ve světě PLC správně zvolena komunikace hraje důležitou roli. Použitý protokol může být odlišný v případě komunikaci s hardwarovými I/O a pro výměny dat různými způsoby. hardwarovými I/O a pro výměny dat různými způsoby.

Když jsou PLC moduly připojeny do sítě, jsou použity standardní komunikační protokoly.

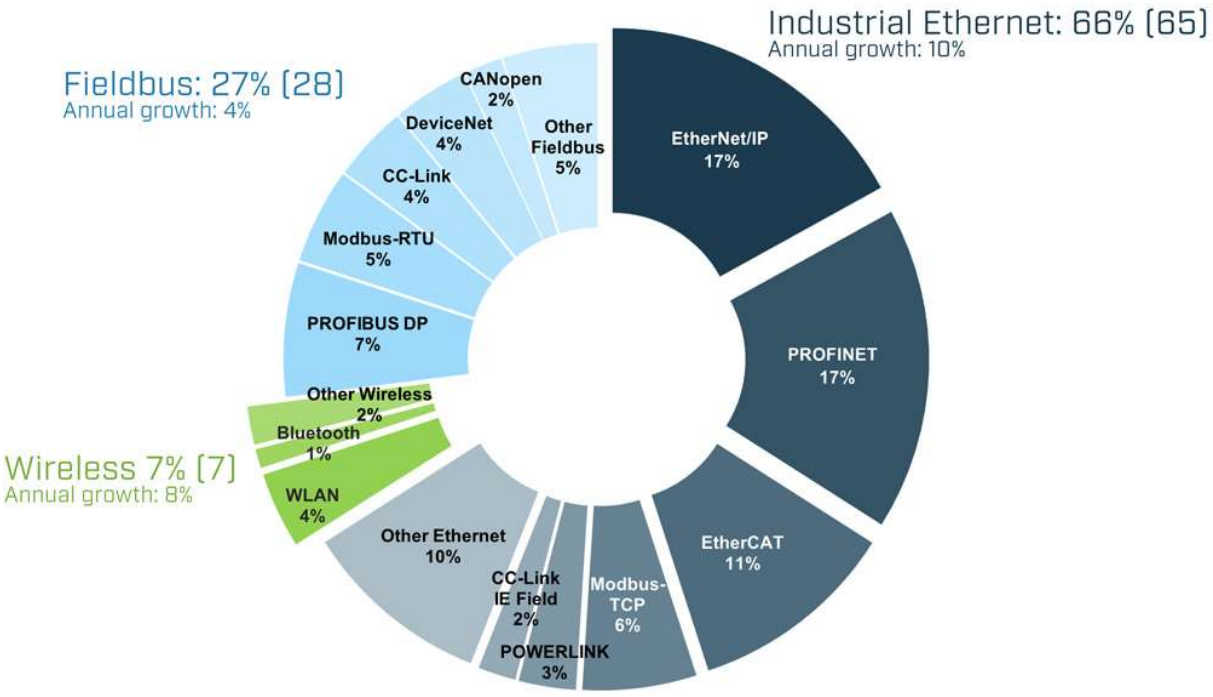
[7]Standardní komunikační protokoly mají určité vlastnosti, jako, například, podporovaná přenosová rychlost (baud rate), maximální vzdálenost (segment lenght) a počet připojovacích zařízení (nodes). V Tabulce č.1 uvedeny parametry pro vybrané komunikační protokoly.

	Protocol	přenosová rychlost	vzdálenost	zařízení
1	Ethernet	100 Mb/s	pár km	255
2	Profibus	5-12 Mb/s	15Km	127
3	MPI	19,2-38,4 Kb/s	50 m	32
4	PPI	187,5 Kb/s	500 m	1
5	DH	230,4 Kb/s	3,048 m	64
6	Device Net	500 Kb/s	0,487 m	64

Tabulka č. 1 - porovnání komunikačních protokolů [7]

Z uvedené tabulky lze vidět, že ethernet vede jak v přenosové rychlosti, tak v počtu zařízení, které lze připojit.

Rozložení trhu s komunikačními protokoly



Obrázek č. 1 - Rozložení komunikačního trhu [8]

Na obrázku č. 1 lze vidět rozložení nejpoužívanějších komunikačních protokolů z května roku 2022 kde silně převahují protokoly EtherNet/IP a PROFINET

Při vyhodnocování komunikačních protokolu ze zjištěných informací lze usoudit, že pro nás nejideálnější bude tvořit emulaci v EtherNet/IP jelikož se jedná o jeden z výkonnějších a také nejpopulárnějších protokolů na trhu.

4.2. Úvod do PLC programování

[9] Programování PLC je nezbytným krokem při návrhu a implementaci řídicí aplikace v závislosti na potřebách zákazníka. Program PLC se skládá ze sady instrukcí buď v textové nebo grafické formě, které představují logiku, která má být implementována pro konkrétní průmyslové aplikace.

Vyhrazený programovací software pochází z hardwaru konkrétního PLC. Software umožňuje zadávání podporovaných instrukci a vývoj kódu, odpovídající potřebám uživatelské aplikace. Na konci vývoje kód v binární formě lze stáhnout do paměti PLC. V určitých případech tento software také zajišťuje rozhraní, zvané Human Machine Interface (HMI), které je vytvořeno jako grafické znázornění proměnných a probíhajících procesu. Jakmile se tento program stáhne do PLC, to potom může být uvedeno do režimu Provoz („Run“), pak nepřetržitě pracuje podle programu.

4.2.1. uživatelský program

Jedná se o kombinaci různých funkcí a logice, které jsou potřebné pro zpracování navrženého úkolu. Některé z úkolů uživatelského programu zahrnují:

- Zajištění podmínek pro spuštění zadané úlohy
- Čtení a vyhodnocování všech binárních a analogových vstupních signálů
- Určení výstupních signálů pro binární a analogové výstupní signály

Provádění přerušení a zpracování chyb

[10] V době skenování PLC je **přerušení** signálem do PLC indikujícím událost, která vyžaduje okamžitou pozornost. Přerušení upozorní PLC na stav s vysokou prioritou vyžadující přerušení aktuálního kódu, který PLC provádí.

V současném sektoru průmyslové automatizace existuje několik předních výrobců PLC, kteří vyvíjejí PLC od jednoduchých jednoúčelových až po multi-modulové a multi-účelové PLC. Většina výrobců PLC má svůj vlastní vyhrazený software pro programování a konfiguraci hardwaru PLC. Programovací jazyk PLC se také liší v závislosti na výrobci. Někteří výrobci používají standardizované programovací jazyky a někteří mají své vlastni. Standardní programovací jazyky pro PLC jsou v zásadě dvou typů, textové jazyky a grafické jazyky, které se dále dělí na několik podtypů (viz. kapitoly 2.2.1 a 2.2.2).

4.2.2. Textové jazyky

Textové jazyky nedisponují žádným uživatelským rozhraní, PLC ovládají pomocí jednoduchých programovacích jazyků skládající se z klíčových slov, hodnot a proměnných. Mezi textové jazyky patří structured text (ST) a instruction list (IL)

4.2.2.1. Structured text

[11] Strukturovaný text, zkráceně ST nebo STX, je jedním z pěti jazyků podporovaných standardem IEC 61131-3 určeným pro programovatelné logické automaty (PLC). Jedná se o vysokoúrovňový jazyk, který je blokově strukturován a syntakticky připomíná Pascal, na kterém je založen. Proměnné a volání funkcí jsou definovány společnými prvky, takže vjednom programu mohou být použity různé jazyky v rámci normy IEC 61131-3.

[12] Vzhledem k tomu, že strukturovaný text je podobný tradičním programovacím jazykům na vysoké úrovni, může být pro mnoho lidí, kteří nemusí mít zkušenosti s programováním PLC, ale mají zkušenosti s tradičním kódováním, poměrně snadné učit se a vyvíjet a rozšiřovat projekty PLC.

4.2.2.2. Instruction list

[13]IL se nejvíce podobají programování v assembleru. Jak název napovídá, program je řada instrukcí, které jsou uvedeny v podstatě stejným způsobem jako program sestavení. Takže, například, některé běžné operace jsou matematické, jako je sčítání, odečítání, násobení a dělení hodnot. Další operace mohou zahrnovat skok na štítek programu, stejně jako volání nebo návrat ze samostatných funkcí.

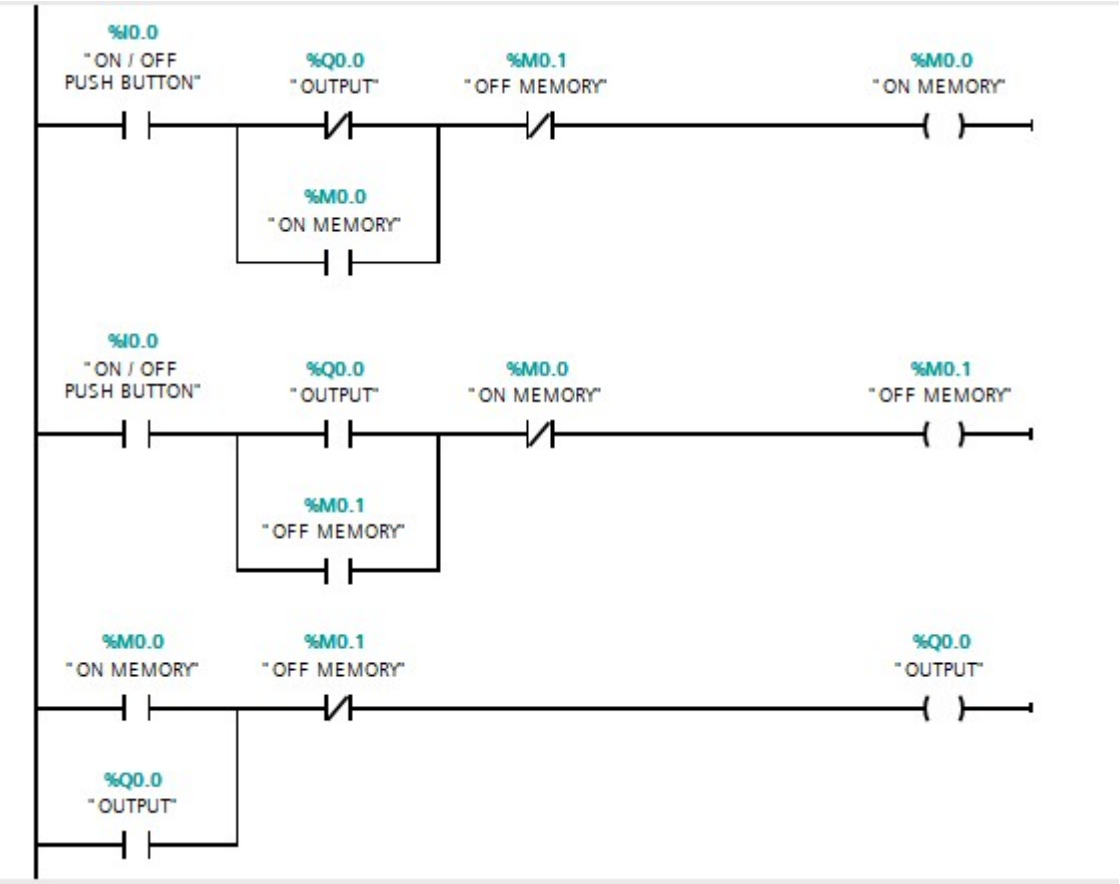
4.2.3. Grafické jazyky

Narozdíl od textových jazyků, jak již název napovídá grafické jazyk disponují grafickým rozhraní, které uživatel musí používat. grafické jazyky vznikly za účelem zjednodušení ovládání PLC pro uživatele který není tolik obeznámen se strukturou programovacích jazyků, a tudíž pro něj převedený do grafické formy.

4.2.3.1. Ladder diagram

[14] Nejběžnějším jazykem používaným k programování PLC je Ladder Diagram (LD), také známý jako Relay Ladder Logic (RLL).
Toto je grafický jazyk zobrazující logické vztahy mezi vstupy a výstupy, jako by to byly kontakty a cívky v pevně zapojeném elektromechanickém reléovém obvodu.

Tento jazyk byl vynalezen pro výslovný účel, aby programování PLC působilo „přirozeně“ pro elektrikáře, kteří jsou obeznámeni s logickými a řídicími obvody založenými na relé. Zatímco programování žebříkových diagramů má mnoho nedostatků, zůstává extrémně populární v průmyslové automatizaci. Následující Obrázek č. 2 ukazuje screenshot typického Ladder Diagram programu.



Obrázek č. 2 - ukázka žebříčkového diagramu

4.2.3.2. Sequential function charts

[15]Klíčovými koncepty, na nichž jsou SFC založeny, jsou kroky a přechody. Krok je v podstatě nějaká funkce v rámci celkového systému, jako je individuální strojový proces. Přechod je právě to, změna z jednoho kroku do dalšího kroku nebo stavu. Kromě základů mohou programy SFC také zahrnovat standardní techniky logického programování, jako

jsou zpětnovazební smyčky a větvení (buď paralelní nebo alternativní větve). SFC lze také navrhovat pomocí stavových diagramů.

4.2.3.3. Function block diagram

[16] Funkční blokový diagram je grafický jazyk pro návrh programovatelného logického regulátoru, který dokáže popsat funkci mezi vstupními proměnnými a výstupními proměnnými. Funkce je popsána jako sada elementárních bloků. Vstupní a výstupní proměnné jsou připojeny k blokům spojovacími linkami.

Závěr PLC programování

Jelikož PLC emulace bude vytvořena v programovacím jazyce Python a pro uživatele kteří jsou již seznámeni s tradičním textovým programování, bude pro naše účely nejvýhodnější napodobovat textové PLC programování, zejména structured text, který se svým chování i podobá programování v Python a dalších objektových jazycích.

4.3. EtherNet/IP

EtherNet/IP je jeden z nejrozšířenějších komunikačních protokolů v automatizované výrobě. To a další byly důvody proč byl vybrán zrovna tento protokol na výrobu emulace. V této kapitole budou popsány jeho funkce a struktura.

4.3.1. Odva

[17] ODVA (dříve Open DeviceNet Vendors Association, Inc.) byla založena v roce 1995 a je celosvětovou asociací, jejíž členové zahrnují přední světové společnosti v oblasti automatizace. Poslání ODVA je podporovat otevřené, interoperabilní informace a komunikaci technologie v průmyslové automatizaci. ODVA uznává svá média, nezávislý síťový protokol, Common Industrial Protocol nebo „CIP“ – a síťové adaptace CIP – EtherNet/IP, DeviceNet, CompoNet a ControlNet – jako jeho základní technologie a primární společný zájem svého členství. Pro budoucí interoperabilitu výrobních systémů a integrace výrobních systémů s ostatnímy systémy, ODVA zahrnuje přijetí commercial-off-the-shelf (COTS) a standardní, neupravené technologie internetu a Ethernetu jako vůdčím principem, kdekoli je to možné. Tento princip je ilustrován např EtherNet/IP – světová jednička v průmyslové síti Ethernet.

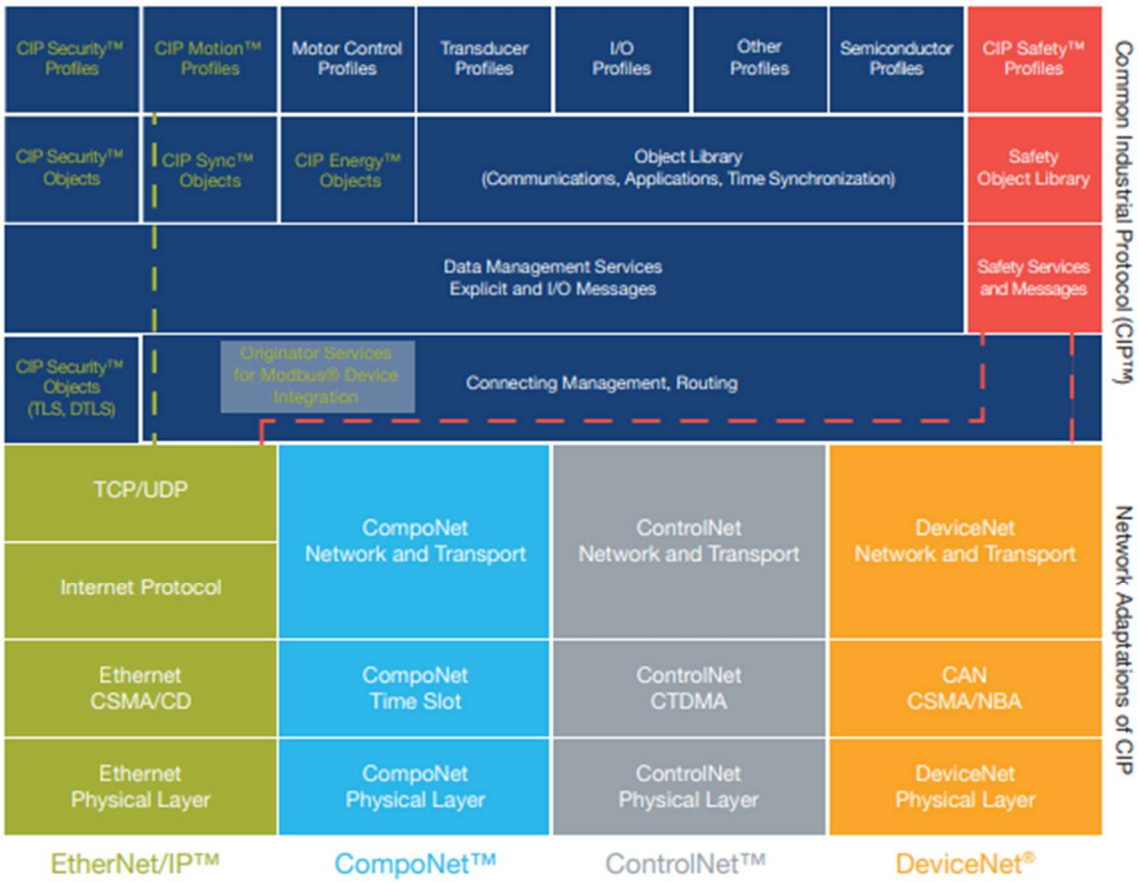
4.3.2. Odva úvod do EIP

[18] EtherNet/IP™ byl představen v roce 2001 a dnes je nejvíce vyvinuté, osvědčené a kompletní průmyslové Ethernet síťové řešení pro automatizaci výroby a procesů. EtherNet/IP je členem rodiny sítí, které implementuje společný průmyslový protokol (CIP™)

na své vrchní vrstvě. CIP zahrnuje komplexní sadu zpráv a služeb pro řadu výrobních a proces automatizačních aplikací, zahrnující řízení, bezpečnosti, zabezpečení, energie, synchronizace, pohyb, konfigurace a informace. Jako protokol skutečně nezávislý na médiích, který je podporovaný stovkami prodejců po celém světě, CIP poskytuje uživatelům jednotnou komunikační architekturu v celém výrobním podniku.

S nezávislostí médií přichází i možnost volby síť CIP, která je užitečná pro každou aplikaci. Jedna z těchto možných voleb je EtherNet/IP, která přizpůsobuje CIP Technologie Ethernet. Proč přizpůsobit CIP Ethernetu? Ethernet a TCP/IP – standard Ethernet – jsou stejné síťová technologie používaná ve většině architektur lokálních sítí (LAN) a rozlehlých sítí (WAN), které se nacházejí v komerčních aplikacích po celém světě. Tyto architektury propojují počítače a periferní zařízení, obchodní operace s podnikem, poskytují uživatelům přístup k webovým aplikacím a mají instalovanou základnu čítající miliónů uzlů. Díky využití úspor z rozsahu v této osvědčené komerční technologii poskytuje EtherNet/IP uživatelům nástroje pro nasazení standardní ethernetové technologie pro výrobní a procesní aplikace, zlepšuje konektivitu mezi lidmi, partnery a procesy, zařízeními, odděleními a systémy v průmyslových aplikacích a otevírá nové příležitosti pro produktivitu, efektivitu a flexibilitu.

4.3.3. Co je EIP



Obrázek č. 3 - protokoly EtherNet/IP [19]

[19] EtherNet/IP, stejně jako ostatní sítě CIP, se řídí modelem Open Systems Interconnection (OSI), který definuje rámec pro implementaci síťových protokolů v sedmi vrstvách: fyzická, datová linka, síť, přenos, relace, prezentace a aplikace. Sítě, které se řídí tímto modelem, definují kompletní sadu síťových funkcí od fyzické implementace přes aplikační vrstvu nebo vrstvu uživatelského rozhraní. Stejně jako u všech sítí CIP implementuje EtherNet/IP CIP na vrstvě Session a výše a přizpůsobuje CIP konkrétní technologii EtherNet/IP na vrstvě Transport a níže. Tato síťová architektura je znázorněna na obrázku č. 3.

[19]Ethernet má jedinečnou vlastnost, že jde o síť s aktivní infrastrukturou. Proto na rozdíl od typických průmyslových sítí – které mají obecně pasivní infrastrukturu, která omezuje počet zařízení, která lze připojit, a způsob jejich připojení – síťová infrastruktura EtherNet/IP může pojmout prakticky neomezený počet uzlů typu point-to-point. a s technologií vestavěných přepínačů mohou také podporovat lineární a kruhové topologie, které uživatelům poskytují nepřekonatelnou flexibilitu při navrhování sítí, které vyhovují jejich současným požadavkům a zároveň umožňují snadné a nákladově efektivní rozšíření v budoucnu.

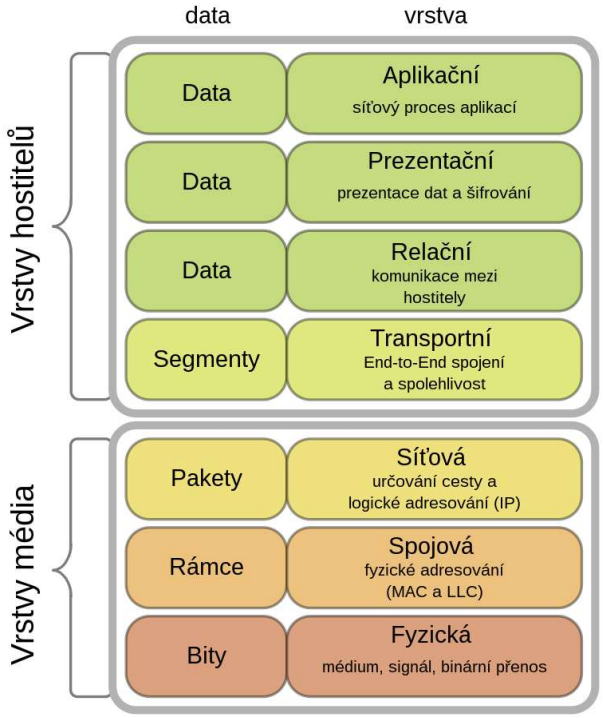
4.3.4. Fyzická vrstva

[19]EtherNet/IP využívá standardní technologii IEEE 802.3 Fyzické a data link vrstvy. Tato norma poskytuje a specifikace pro fyzická média, definuje společný rámcový formát pro přesouvání paketů dat mezi zařízeními a poskytuje sadu pravidel pro určování toho, jak síťová zařízení mají reagovat, když se dvě zařízení pokusí použít datový kanál zároveň. Toto je známé jako CSMA/CD (Carrier Sense Multiple Access/Collision Detection).
Jako síť s aktivní infrastrukturou je EtherNet/IP typicky konfigurován pomocí řady síťových segmentů vytvořených z dvoubodových spojení v hvězdicové konfiguraci. Jádrem této topologie sítě je propojení přepínačů Ethernet Layer 2 a Layer 3, které, jak již bylo zmíněno, mohou pojmout neomezený počet uzlů typu point-to-point. Sítě EtherNet/IP však mohou také implementovat lineární větvení a kruhovou topologii tolerantní vůči jediné chybě využitím technologie vestavěných přepínačů a technologie Device Level Ring (DLR™). Tyto alternativní topologie lze kombinovat pro optimalizaci vedení kabelů a uspořádání komunikace stroje.

4.3.5. Data link vrstva

[19]Specifikace IEEE 802.3 je také standard používaný pro přenos paketů dat ze zařízení do zařízení na vrstvě EtherNet/IP Data Link Layer. Ethernet využívá mechanismus přístupu k médiím CSMA/CD, který určuje, jak síťová zařízení sdílejí společnou sběrnici (tj. kabel) a jak detekují a reagují na kolize dat.

Původně Ethernet fungoval v polovičním duplexním režimu, což znamená, že uzel mohl odesílat nebo přijímat data, ale nemohl dělat obojí současně. To způsobilo zácpy datového provozu, které jsou v časově kritických řídicích aplikacích nepřijatelné. Nyní, s plně duplexním Ethernetem (dnešní de facto standard), mohou síťová zařízení odesílat a přijímat pakety ethernetových dat současně. Díky tomu, spolu s pokrokem v technologii přepínání, je Ethernet vhodný pro použití v celé šířce výrobních nebo procesních aplikací. Obrázek č. 4 je přenosový řád.

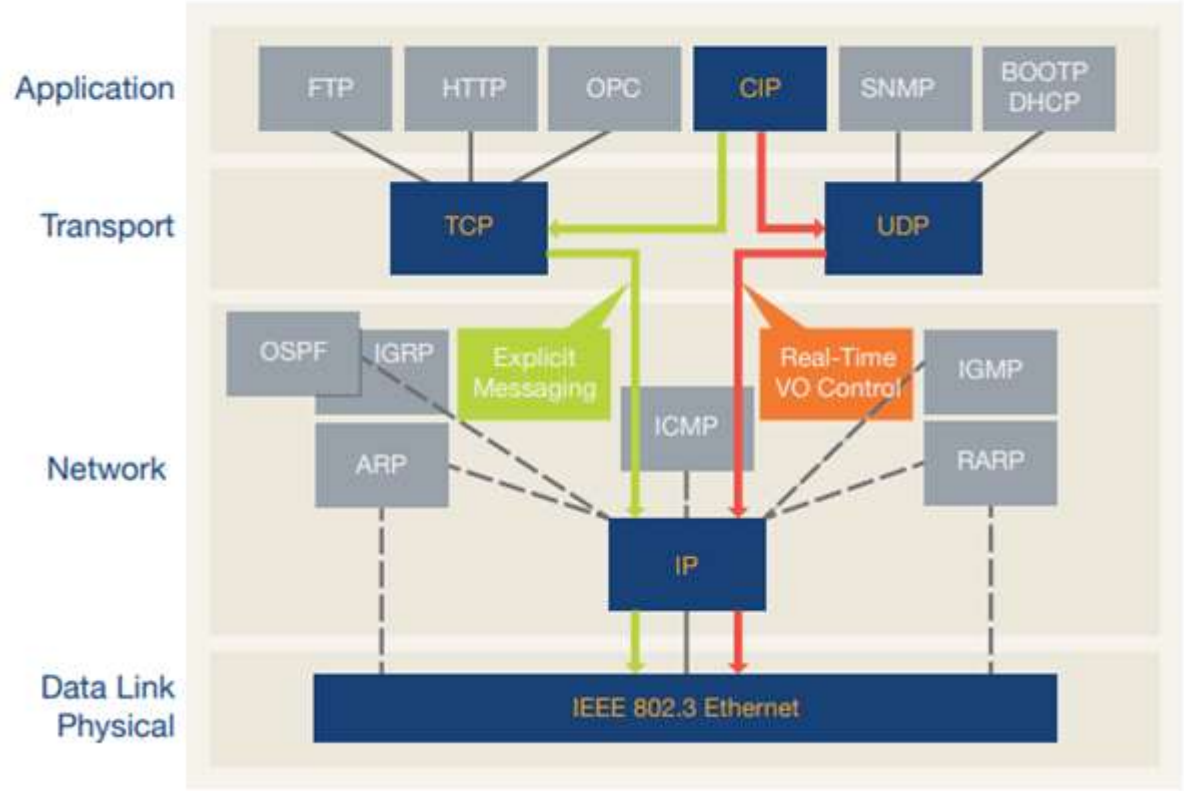


Obrázek č. 4 - Postup přenosu [20]

[19]Protokol Media Access Control (MAC) specifikace IEEE 802.3 ve skutečnosti umožňuje zařízením „mluvit“ v síti Ethernet. Každé zařízení má jedinečnou MAC adresu složenou z 6bajtového čísla, které je regulováno IEEE a výrobcem produktu, aby byla zachována jedinečnost. Tato MAC adresa se používá v poli zdrojové adresy (SA) rámce k označení, který uzel poslal rámec, a používá se v poli cílové adresy (DA) k označení cíle rámce. Nastavení prvního bitu na „1“ v poli DA označuje paket dat pro více cílů a umožňuje ethernetovému uzlu přenášet jeden datový paket do jednoho nebo více cílových uzlů.

4.3.6. Network a transport vrstvy

[19]V síťové a transportní vrstvě využívá EtherNet/IP standardní TCP/IP (Transmission Control Protocol/Internet Protocol) k odesílání zpráv mezi jedním nebo více zařízeními. TCP/IP se používá v kancelářských aplikacích po desetiletí a má širokou znalost a podporu. Poskytuje nezbytné vlastnosti komunikačního protokolu potřebné k implementaci plně funkčních sítí (tj. schéma adresování a mechanismy pro navázání spojení se zařízením a výměnu dat). Na obrázku č. 5 lze vidět kompatibilitu CIP s široce používanými internet protokoly.



Obrázek č. 5 - komptabilita s internetovými protokoly [19]

Na těchto vrstvách jsou také zapouzdřeny standardní zprávy CIP používané všemi sítěmi CIP. Toto zapouzdření TCP/IP umožňuje uzlu v síti vložit zprávu CIP do zprávy Ethernet a odeslat ji do jiného uzlu v síti pomocí standardního protokolu TCP/IP.

4.3.6.1. TCP

[19]část protokolu TCP/IP je spojově orientovaný transportní mechanismus typu point-to-point (unicast), který zajišťuje řízení toku dat, opětovné sestavení fragmentace a potvrzení zpráv. Uzly obdrží každou zprávu a potvrdí odesílateli, že byla přijata. Pokud je zpráva fragmentována do více rámců, odesílatel odešle další fragment, který je potvrzen. Toto se opakuje, dokud není přijata celá zpráva. V té době přijímající uzel zpracuje data a podle toho bude jednat. Protože TCP je ideální pro spolehlivý přenos velkého a malého množství dat, používá EtherNet/IP TCP/IP k zapouzdření explicitních zpráv CIP, které se obecně používají k přenosu konfiguračních a diagnostických dat a také k vytvoření v reálném čase (implicitní) datové přenosy mezi zařízeními.

4.3.6.2. IP

[19]Část protokolu TCP/IP je mechanismus, který zajišťuje směrování paketů přes více možných cest. Základem internetového protokolu je schopnost posílat zprávy na jejich místa určení, i když je primární cesta narušena. Protože EtherNet/IP používá standardní IP, stejný typ směrování se používá k udržení správného oddělení řídicích prvků na úrovni továrny a dalších výrobních systémů pomocí standardní infrastruktury, jako jsou spravované přepínače a směrovače vrstvy 3.

4.3.6.3. UDP

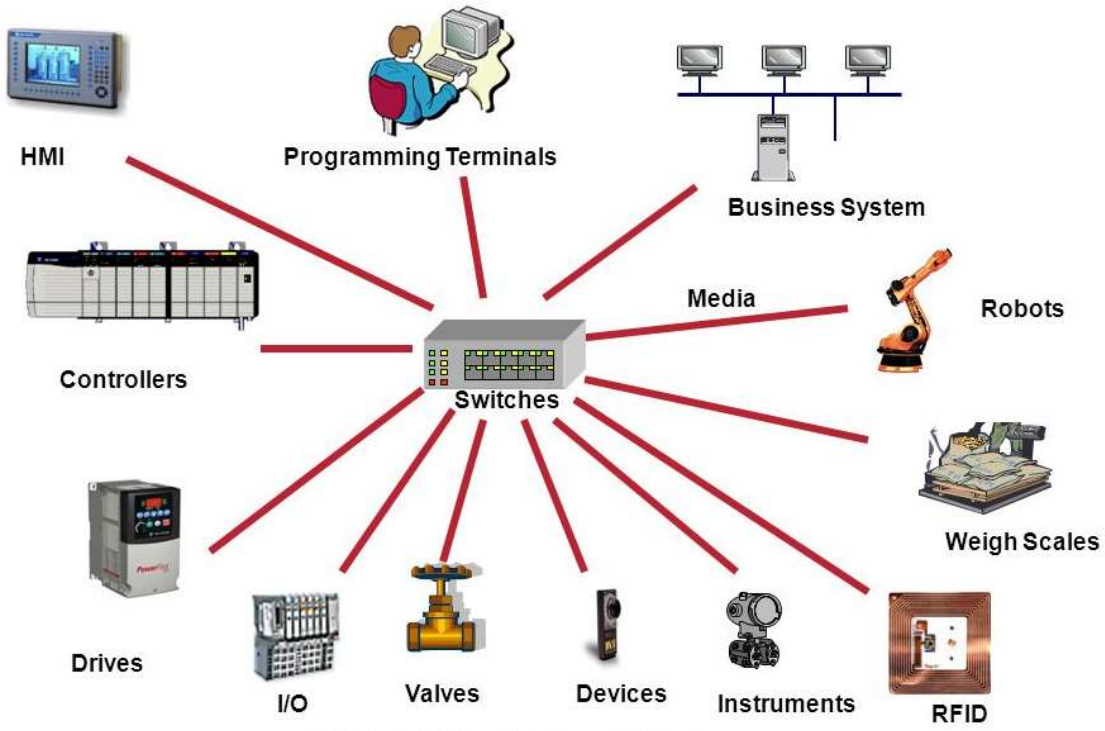
[19]Pro přenos dat v reálném čase využívá EtherNet/IP také UDP přes IP k přenosu I/O zpráv, které obsahují časově kritická řídicí data. UDP je mechanismus přenosu bez připojení, který má nízkou protokolovou velikost režii, poskytuje menší paketů a umožňuje vícesměrné vysílání do více než jednoho cíle. Menší pakety a podpora vícesměrného vysílání nabízí model Producer/Consumer s EtherNet/IP a zajišťuje efektivní tok datového systému, zatímco mechanismus CIP Connection poskytuje mechanismy časového limitu, které mohou odhalit problémy s doručováním dat. Z těchto důvodů je UDP výhodné pro přenos implicitních dat (tj. I/O) v reálném čase na EtherNet/IP.

[19]Proces otevírání spojení se nazývá Connection Origination a uzel, který iniciuje požadavek na navázání spojení, se nazývá Connection Originator nebo jen Originator. Naopak uzel, který odpovídá na žádost o vytvoření, se nazývá cíl připojení nebo cíl.

EtherNet/IP má dva typy připojení zpráv:

- **Explicitní spojení** zasílání zpráv jsou vztahy point-to-point, které jsou vytvořeny pro usnadnění transakcí typu request-response mezi dvěma uzly. Tato připojení mají obecný účel a obvykle se používají pro časté požadavky mezi dvěma uzly. Lze je použít k dosažení všech položek v zařízení dostupných v síti. Explicitní spojení pro zasílání zpráv využívají služby TCP/IP k přenosu zpráv přes Ethernet.
- **Implicitní (I/O data)** připojení jsou vytvořena pro přesun dat I/O specifických pro aplikaci v pravidelných intervalech. Tato připojení lze nastavit jako vztahy jeden k jednomu nebo jako jeden k mnoha, aby bylo možné plně využít výhod modelu multicast producent-spotřebitel. Implicitní zasílání zpráv využívá zdroje UDP/IP, aby se multicastové datové přenosy přes Ethernet staly realitou.

4.3.7. Dostupné EtherNet/IP produkty



Obrázek č. 6 - Dostupné EtherNet/IP produkty [21]

Obrázek č. 6 ukazuje EtherNet/IP zařízení na trhu a jakým způsobem se zapojují do sítě.

4.3.7.1. Produkty třídy zasílání zpráv

[19]podporují zasílání explicitních zpráv (připojených nebo nepřipojených), které jsou odesílány nebo přijímány ze všech ostatních tříd produktů. Produkty Messaging Class jsou cílem explicitních požadavků na připojení zpráv a mohou být také původci těchto požadavků, ale neodesílají ani nepřijímají I/O data v reálném čase.
Příklady produktů v této třídě zahrnují:

- Zařízení, která provádějí konfiguraci a programování produktů HMI, robotů a PLC;
- Zařízení s aplikacemi, které poskytují operátora rozhraní k řídicím systémům (tj. produkty HMI);
- Softwarové aplikace, které nevyžadují I/O v reálném čase odezva (např. aplikace MIS);
- Konfigurace sítě a diagnostické nástroje.

4.3.7.2. Produkty třídy adaptérů

[19]Jsou cílem požadavků na datové připojení I/O v reálném čase z produktů třídy skenerů. Nemohou odesílat ani přijímat I/O data v reálném čase, pokud je o to nepožádá skener, a neukládají ani nevytvářejí parametry datové komunikace potřebné k navázání spojení.

Produkty třídy adaptérů přijímají explicitní požadavky na zprávy (připojené a/nebo nepřipojené) od všech ostatních tříd produktů. Mohou si také vyměňovat (rovnocenná) data pomocí explicitních zpráv s jakoukoli třídou zařízení, ale obvykle nemohou vytvářet takové vztahy.
Příklady produktů v této třídě zahrnují:

- I/O zařízení, jako jsou I/O bloky nebo stojany I/O modulů, které produkují a spotřebovávají I/O data v reálném čase;
- Vážící váhy, svářečky, pohony a roboty, které odesílají a přijímají data v reálném čase na žádost PLC a dalších řídicích jednotek;
- Produkty HMI, které odesílají nebo přijímají explicitní nebo v reálném čase I/O data do/z PLC nebo jiných řídicích jednotek.

4.3.7.3. Produkty třídy skenerů

[19]Jsou původci požadavků na datové připojení I/O k produktům třídy adaptérů a také k dalším produktům třídy skenerů, které podporují funkce třídy adaptérů (tj. explicitní peer-to-peer nebo I/O data). Tyto produkty jsou obvykle také původci nebo cíli explicitních požadavků na připojení k jiným třídám produktů a od nich a mohou také odesílat nebo přijímat explicitní zprávy do nebo ze všech ostatních tříd produktů.
Příklady produktů v této třídě zahrnují:

- PLC, ovládací prvky na bázi PC, další řídicí jednotky a roboty, které odesílají a přijímají data v reálném čase do a z I/O zařízení, PLC, ovládací prvky na bázi PC, pohony, roboty, váhy, svářečky a produkty HMI;
- PLC, řídicí jednotky a roboty, které odesílají a přijímají data explicitních zpráv do a z jiných PLC, robotů, vah, řídicích systémů na bázi PC, svářeček a produktů HMI.

4.3.8. Vrchní vrstva

[19]EtherNet/IP využívá na vyšších vrstvách Common Industrial Protocol (CIP), což je objektově orientovaný protokol. Každý objekt CIP má dobře definované atributy (data), služby (příkazy) a chování (reakce na události). Komunikační model CIP výrobce-spotřebitel poskytuje efektivnější využití síťových zdrojů než čistý model zdroj-cíl tím, že umožňuje výměnu aplikačních informací mezi odesílajícím zařízením (např. výrobcem) a mnoha přijímajícími zařízeními (např. spotřebiteli) bez nutnosti dat být přenášen vícekrát jedním zdrojem do každého jednotlivého cíle. V sítích producent-spotřebitel je zpráva identifikována svým ID připojení, nikoli svou cílovou adresou (jako je tomu u sítí zdroj-cíl). V EtherNet/IP je to realizováno pomocí kombinace ID připojení a skupinové adresy IP Multicast.

[19]CIP také zahrnuje „typy zařízení“, pro které existují Profily zařízení. Pro daný typ zařízení bude Profil zařízení specifikovat sadu objektů CIP, které musí být implementovány, možnosti konfigurace a formáty I/O dat. Tato konzistence v implementaci objektu pro daný typ zařízení poskytuje další jasnou výhodu pro uživatele CIP sítí tím, že podporuje společné aplikační rozhraní pro daný typ zařízení a interoperabilitu v sítích složených ze zařízení od více dodavatelů. Pro aplikace, kde je vyžadována jedinečná funkčnost, je také možné, aby prodejce EtherNet/IP definoval další objekty specifické pro dodavatele v produktu kompatibilním s EtherNet/IP, aby podpořil funkční požadavky konkrétních aplikací, které jsou pro daného dodavatele jedinečné.

4.4. Softwarové prvky

4.4.1. Python

[22] Python je univerzální programovací jazyk na vysoké úrovni. Jeho filozofie designu klade důraz na čitelnost kódu s použitím výrazného odsazení.

Python podporuje více programovacích paradigmat, včetně strukturovaného (zejména procedurálního), objektově orientovaného a funkčního programování. Díky své komplexní standardní knihovně je často popisován jako multifunkční.

[23] Guido van Rossum začal pracovat na Pythonu na konci 80. let jako nástupce programovacího jazyka ABC a poprvé jej vydal v roce 1991 jako Python 0.9.0. Python 2.0 byl vydán v roce 2000 a zavedl nové funkce, jako je porozumění seznamům, cyklické shromažďování odpadků, počítání odkazů a podpora Unicode. Python 3.0, vydaný v roce 2008, byl hlavní revizí, která není zcela zpětně kompatibilní s dřívějšími verzemi. Python 2 byl ukončen s verzí 2.7.18 v roce 2020.

[24]V roce 2022 byly Python 3.10.4 a 3.9.12 urychleny a 3.8.13 a 3.7.13 kvůli mnoha bezpečnostním problémům. Když byl v květnu 2022 vydán Python 3.9.13, bylo oznámeno, že řada 3.9 (připojující se ke starším sériím 3.8 a 3.7) bude v budoucnu dostávat pouze bezpečnostní opravy. 7. září 2022 byly kvůli potenciálnímu útoku odmítnutí služby vydány čtyři nové verze: 3.10.7, 3.9.14, 3.8.14 a 3.7.14.

4.4.2. Pycomm3

[25] pycomm3 je veřejná knihovna, která rozšiřuje python o funkce na komunikaci s CIP zařízení za pomoci Ethernet IP, proto je tato knihovna důležitou částí projektu. Dále je stručně vypsána historie a nejužitečnější schopnosti této knihovny.

pycomm3 začal jako odvětví Pythonu 3 pycomm, což je knihovna Pythonu 2 pro komunikaci s PLC Allen-Bradley pomocí EtherNet/IP. Počáteční port Python 3 byl proveden v této vidlici a byl použit jako základ pro pycomm3. Od té doby byla knihovna téměř celá přepsána a API již není kompatibilní s pycomm. Bez tvrdé práce, kterou odvedli původní vývojáři pycomm, by pycomm3 neexistoval. Tato knihovna se snaží rozšířit jejich skvělé dílo.

4.4.2.1. CIPDriver

[25]Tento ovladač je základním ovladačem pro knihovnu, zpracovává běžné služby CIP používané ostatními ovladači. Věci jako otevření/uzavření připojení, registrace/zrušení registrace relací, předávání služeb otevírání/zavírání, zjišťování zařízení a obecné zasílání zpráv. Lze jej použít pro připojení k jakémukoli zařízení EtherNet/IP, jako jsou: pohony, přepínače, měřiče a další zařízení bez PLC.

4.4.2.2. LogixDriver

[25]Tento ovladač podporuje služby specifické pro PLC ControlLogix, CompactLogix a Micro800. Služby jako čtení/zápis tagů, nahrávání seznamu tagů a získávání/nastavení času PLC.

4.4.2.3. SLCDriver

[25]Tento ovladač podporuje základní čtení/zápis datových souborů v PLC SLC500 nebo MicroLogix. Je to port SlcDriveru od pycomm s minimálními změnami, aby se API podobalo ostatním ovladačům. V současné době je tento ovladač považován za starší a jeho vývoj bude omezený.

4.4.2.4. Práce s EDS soubory

[25]Pycomm3 také podporuje práci s EDS soubory. Na oficiálních dokumentací jsou návody pro práci s nimi.

4.4.3. EDS soubory

[26] Soubor EDS je strukturovaný textový soubor ASCII, který obsahuje popis síťově konfigurovatelných parametrů uvnitř zařízení chytrého produktu. Soubor umožňuje síťovým konfiguračním nástrojům interpretovat data přenášená ze zařízení a konfigurovat různé parametry zařízení.

4.4.4. WireShark

[27]Wireshark je bezplatný a open-source analyzátor paketů. Používá se pro řešení problémů se sítí, analýzu, vývoj softwaru a komunikačních protokolů a vzdělávání. Projekt se původně jmenoval Ethereal a v květnu 2006 byl kvůli problémům s ochrannou známkou přejmenován na Wireshark.

[27]Wireshark umožňuje uživateli přepnout řadiče síťového rozhraní do promiskuitního režimu (pokud to řadič síťového rozhraní podporuje), takže mohou vidět veškerý provoz viditelný na tomto rozhraní, včetně provozu unicast neodesílaného na MAC adresu řadiče síťového rozhraní. Při zachycování pomocí analyzátoru paketů v promiskuitním režimu na portu na síťovém přepínači však není veškerý provoz přes přepínač nutně odeslán do portu, kde se zachycování provádí, takže zachycování v promiskuitním režimu nemusí nutně stačit k zobrazení veškerého síťového provozu.

5. Vlastní práce

Cíle této kapitoly je tvorba cyklické komunikace mezi počítačem a připojeným zařízení, kde bude možné měnit stav daného zařízení. To je hlavní a zásadní funkce PLC. Je třeba přesně napodobit komunikaci, která by proběhla při použití hmotného PLC. Vše bude probíhat na protokolu Ethernet\IP a sledováno pomocí softwaru Wireshark. k ovládání bude použit programovací jazyk Python.

5.1. Použitá zařízení a jejich konfigurace

Jako testovací hardware jsou využívány dva moduly a to: SIMATIC ET 200eco PN DQ 8x24VDC 0,5A a SIMATIC ET 200eco PN DI 8x24VDC

Tyto zařízení umožňují rychlé přepínání vstupů a výstupů, což to je hlavním předmětem testování. Signály dokážou zpracovávat a odesílat v minimálním limitu 200 ms až maximálním 20 s kdy základní hodnota je nastavena na 1 s

Obě jsou postaveny za použití stejné architektury. Každé disponuje dvanácti porty, dva zdrojové, dva komunikační, a osm portů, které mají na starost odesílat či přijímat signál. Po straně modulů je umístěno 22 led diod, ty nás ujišťují o správném chodu. Šest je rezervovaných pro komunikační a zdrojové kabely, zbytek na určování stavu vstupů či výstupů. Všechny porty a diody jsou řádně popsány z důvodu přehlednosti.

Jedním z důvodů, proč tyto zařízení byla vybrána je, jelikož obě jsou multifieldbusové, což znamená, že kromě jejich základního komunikačního protokolu Profinet, disponují schopností komunikovat i dalšími, jako je například Modbus, nebo také právě námi vybraný EtherNet/IP.

Fieldbus – je označení pro rodinu komunikačních protokolů pro průmyslovou aplikaci.

Jak bylo již řečeno EtherNet/IP pro tyto zařízení není nastaveno jako základní, proto je třeba toto nastavení změnit. Na to byl použit nástroj MFCT, který je vyvíjen přímo firmou SIEMENS a používá se na konfiguraci firmou vyvíjeného hardwaru.

Další důvod čítá, že je zapotřebí otestovat zařízení, která jsou již delší dobu na volném trhu, ale je nutné ujistit se, že i přes tuto skutečnost jsou schopna díky novému firmwaru navázat komunikaci i v dalších fieldbusech jako je například právě námi testovaný EtherNet/IP.

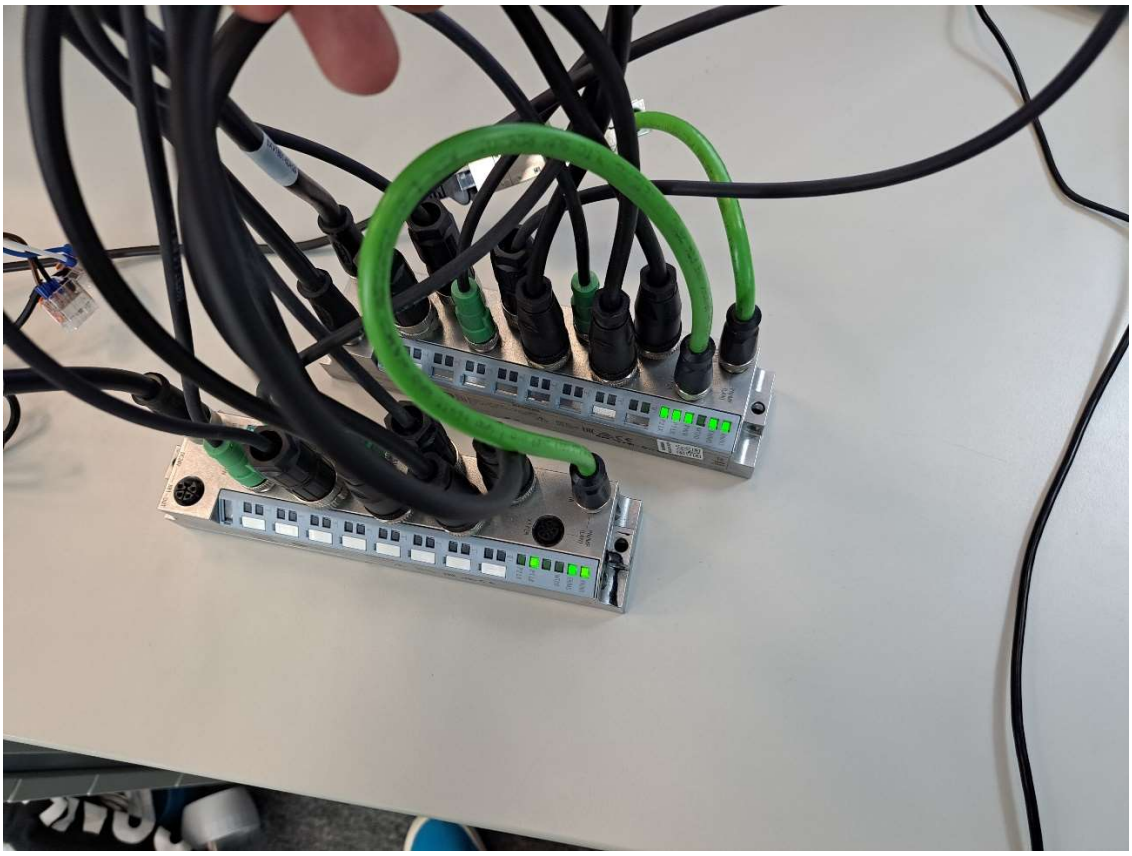
Zařízení je dále odolné proti řadě živelných podmínek jako jsou mráz, voda, či vysoká teplota. Je také velmi odolné i proti otřesům, což z nich dělá ideální nástroj pro využití přímo ve výrobě co nejblíž k přístrojům, u kterých jsou potřeba.

Modul DI neboli vstupní zařízení, které reaguje pouze pokud získává signál zvenčí a nelze přímo nastavit jeho stav. O přijetí signálu nás informuje rozsvěcením a zhasínáním led diod.

Naopak účel DQ modulu je odesílat signály z portů ven a opět nás o tomto ujistí rozsvěcením led diod vztahující se k danému portu. Na rozdíl od DI modulu, tento je schopný přijímat nastavení odeslaná zvenčí, za pomoci UDP zpráv.

Pro naše účely bylo potřeba zařízení napojit do dvaceti čtyř voltového zdroje elektřiny. Toho bylo docíleno úpravou notebookové nabíječky. použitím zdrojového kabelu bylo připojeno do napájení i druhé zařízení. Poté již napájená zařízení byla propojena mezi sebou a počítačem ethernetovým kabelem (zelený kabel). Jelikož máme jak vstupní, tak výstupní zařízení, pro otestování všech portů bylo je zapotřebí mezi sebou propojit.

Ukázku modulů za pomoci kterých bude práce vyhotovena uvidíte na obrázku č. 7



Obrázek č. 7 – DI a DQ moduly

5.2. Navázání spojení se zařízením

Prvním krokem při tvoření cyklické komunikace je navázání spojí se zařízením, ke kterému byl počítač připojen za pomoci EtherNet/IP. Toto spojení probíhá za pomoci packetů, které do zařízení odesíláme a následně zařízení odešle odpověď na naše dotazy. Každý z těchto

packetů obsahuje řadu informací, které jsou v hexadecimální podobě neboli v bytech. Muže jít od booleovské typy po slova a číslice. Tyto informace jsou nadále rozděleny do daných šablon podle typu packetu, který byl přijat nebo odeslán.

Našim cílem v této části je odesílat ony packety s informacemi ve správné podobě a pořadí, jinak je zařízení odmítne, či selže při rozřazování dat, která do zařízení odesíláme. V tomto případě zařízení odmítne vykonat tento příkaz a sdělí nám proč se tomu tak stalo.

Pro zapisování této komunikace byl zvolen Wireshark, který umožnuje zachycovat veškerou odchozí a příchozí komunikaci a poskytuje uživatelské rozhraní pro průzkum této komunikace. Wireshark nám poskytuje základní informace sloužící k jednoduššímu zařazení packetu a zjištění jeho účelu.

Number – popisující pořadí odeslaných či přijmutých packetu.
Time – doba odeslání či přijmutí packetu od začátku naslouchání komunikace.
Source a destination – popisující z jaké IP a do které IP byl packet odeslán.
Protokol – protokol který byl použit.
Lenght – délka packetu uvedená v bytech.
Info – poskytující informaci o který typ packetu šlo.

No.	Time	Source	Destination	Protocol	Length	Info
1	*REF*	HP_6d:8e:cf	LLDP_Multicast	LLDP	172	LA/evc00088nb LA/port-001 20 SysN=EVC00088NB SysD=HP HP Z
2	2.019122	Siemens_d7:e5:e7	LLDP_Multicast	LLDP	269	LA/dq8-05a LA/port-001 20 SysN=dq8-05a SysD=Siemens, SIMA
3	2.617974	192.168.0.100	192.168.0.67	TCP	66	55760 → 44818 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
4	2.618412	Siemens_d7:e5:e6	Broadcast	ARP	60	Who has 192.168.0.100? Tell 192.168.0.67
5	2.618444	HP_6d:8e:cf	Siemens_d7:e5:e6	ARP	42	192.168.0.100 is at c8:5a:cf:6d:8e:cf
6	2.618617	192.168.0.67	192.168.0.100	TCP	60	44818 → 55760 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1
7	2.618798	192.168.0.100	192.168.0.67	TCP	54	55760 → 44818 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	2.619304	192.168.0.100	192.168.0.67	ENIP	82	Register Session (Req), Session: 0x00000000
9	2.621394	192.168.0.67	192.168.0.100	ENIP	82	Register Session (Rsp), Session: 0x00000001
10	2.622467	192.168.0.100	192.168.0.67	CIP	106	Assembly - Get Attribute Single
11	2.641224	192.168.0.67	192.168.0.100	CIP	118	Success: Assembly - Get Attribute Single
12	2.641883	192.168.0.100	192.168.0.67	CIP	106	Assembly - Get Attribute Single
13	2.661249	192.168.0.67	192.168.0.100	CIP	114	Success: Assembly - Get Attribute Single
14	2.662114	192.168.0.100	192.168.0.67	CIP	106	Assembly - Get Attribute Single
15	2.681248	192.168.0.67	192.168.0.100	CIP	99	Success: Assembly - Get Attribute Single
16	2.683398	192.168.0.100	192.168.0.67	CIP CM	150	Connection Manager - Forward Open (Assembly)
17	2.701425	192.168.0.67	192.168.0.100	CIP CM	164	Success: Connection Manager - Forward Open (Assembly)
18	2.701607	192.168.0.67	192.168.0.100	CIP I/O	82	Connection: ID=0xAA0000AA, SEQ=0000000000, T->O
19	2.710758	192.168.0.100	192.168.0.67	CIP I/O	82	Connection: ID=0x8A5E0001, SEQ=0000000000, O->T
20	2.741782	192.168.0.100	192.168.0.67	TCP	54	55760 → 44818 [ACK] Seq=281 Ack=308 Win=63933 Len=0

Obrázek č. 8 - Wireshark ukázka packetů

Na obrazku č. 8 je ukázáno, jak vypadá standardní záznam komunikace ve Wireshark.

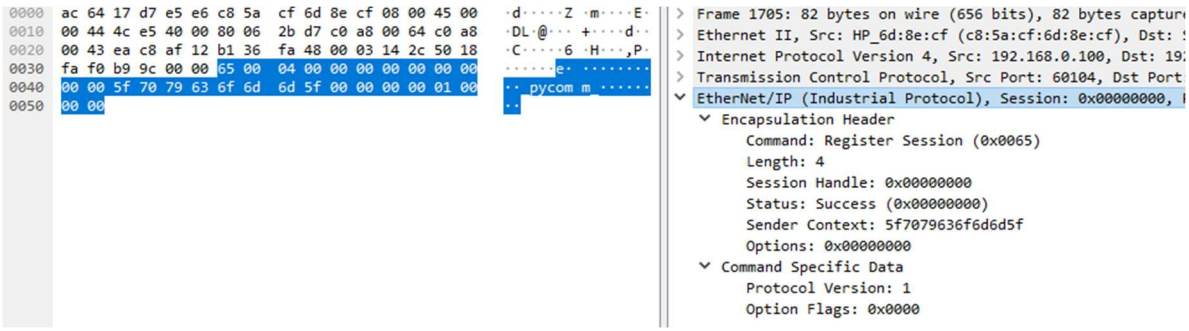
5.2.1. Typy procesů potřebných pro navázání komunikace

Každé navázání komunikace musí obsahovat řadu kroků nutné pro její úspěch, zde je jejich výčet:

- **Register Session (Req)**
- **Register Session (Rsp)**
- **Assembly – Get Attribute Single 0x300**
- **Assembly – Get Attribute Single 0x301**
- **Assembly – Get Attribute Single 0x307**
- **Success: Assembly – Get Attribute Single**
- **Connection Manager – Forward Open**
- **Success: Connection Manager – Forward Open**

5.2.1.1. Register Session

Register Session, který proběhne jako žádost (request) a odpověď (responce) vždy na úplném začátku komunikace. V tomto packetu, jak název napovídá, se registruje náš pokus o navázání spojení. Tomuto spojení se také říká handshake, a jde u něj o to, zda si dvě různá zařízení rozumí. V pythonu spojení spravuje funkce _registred_session, která je součástí CIP_driver kihovny pycomm3 a spouští se v rámci otevírání spojení. Tato funkce má na starost zjistit, zda už bylo spojení navázáno, pokud ano vrátí toto spojení ve formě integeru pokud ne vytvoří nové spojení a navrátí to také ve formě integeru.



Obrázek č. 9 – Ukázka Register session packet

Na obrázku č. 9 lze vidět rozdělení tohoto packetu. Nás vždy zajímá hlavně EtherNet/IP část packetu.

EtherNet/IP se rozděluje na dva pododstavce encapsulation Header což je hlavička naší zprávy a command specific data.

Na začátku encapsulation header Lze vidět, že je Command (příkaz) který určuje, o co se v daném packetu pokoušíme. Je o velikosti 2 bytu a registred session má kód 65 00. Každý příkaz má svůj bytové číslo a další příkazy, které se dají poslat jsou například 66 00 – unregistred session anebo 6F 00 což je send RR data. Následuje délka, která je složená ze

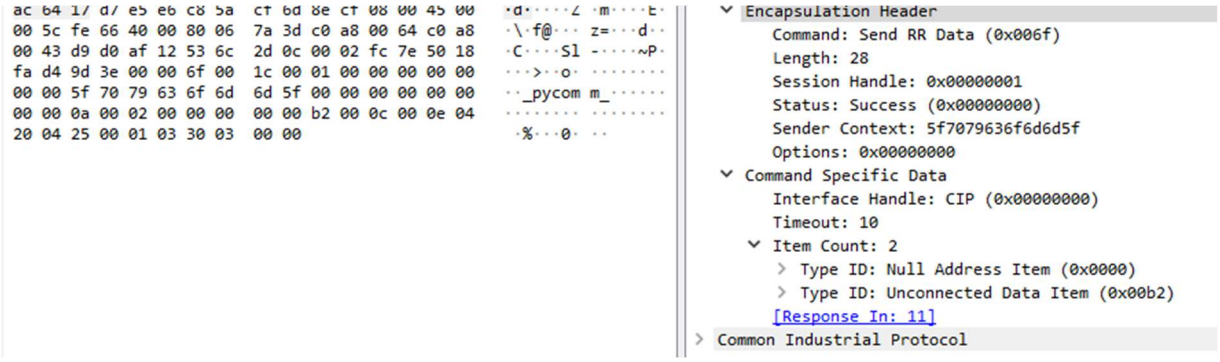
dvou bytů. Poté následuje session handle tvořený čtyřmi byty. Pokud navázání spojení proběhne bezproblémově session handl se v odpovědi na náš dotaz změní na číslo jedna. Dále následuje status, který je také určený čtyřmi byty a vrací nulu při úspěchu či jedna při neúspěchu. Sender Context, zde odesílatel může poslat krátkou zprávu. V naší zprávě je bytový kód pro slovo _pycomm_ jelikož bylo navázáno spojení pomocí knihovny pycomm3. Poslední atribut encapsulations headeru je options který je tvořen čtyřmi byty a v našem případě je prázdný.

Na konci tohoto packetu je Command specific data, kde se nachází po dvou bytech rozdělená protocol version a option flags. U nás je protokolová verze jedna a volitelné flagy nastavené na nulu.

5.2.1.2. Assembly – Get Attribute Single

Tento packet se volá, pokud potřebujeme vyčíst jeden z atributů komunikace sám o sobě. V pythonu jí voláme pomocí funkce generic_message, která se nachází v pycomm3 v CIPDriver skrze assembler s pomocí daných instancí, které mají hexadecimální kód 0x0300, 0x301 a 0x307. Funkce slouží k vyčtení dat, na která jsou zařízení nastavená.

Jak již bylo řečeno každá z instancí má svůj hexadecimální kód. 0x300 má na starosti výstup dat, pokud je zařízení výstupní (DQ) změny v této instanci odesílá ven. 0x301 je input dat, pokud je zařízení vstupní (DI), pouze přijímá nastavení z venčí neboli z DQ zařízení. Poslední je 0x307 což je konfigurace zařízení. Požadavek pouze informace vyčítá, žádným způsobem s nimi nepracuje.

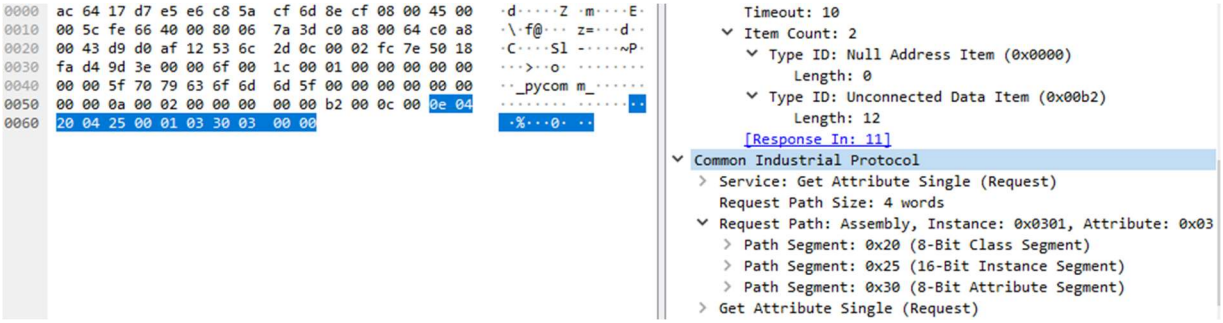


Obrázek č. 10 – Ukázka Assembly Get Attribute Single

Packet Assembly má stejný Encapsulation Header jako Registred Session s jedinou změnou, obsahuje již zmíněny kód pro Send RR Data. Vyjma malé změny v zapouzdření hlavičky se nám změnily atributy v specifické data příkazu a přibyl další odstavec nazvaný Common Industrial Protocol neboli CIP. Ukázka je na obrázku č. 10.

Comand Specific Data nyní obsahuje Inteface Handle, který obsahuje prázdných čtyři byty což je kód pro CIP. Další v řadě je dvěma byty Timeout, který určuje, po jaké době bude příkazu dojde čas a bude zrušen. Dále máme v řadě Item Count určený dvěma byty a v něm nacházející se další dva atributy Type ID, které se oba skládají z čtyř dalších bytů. První type

id je prázdný Null Address Item (první dva byty) s délkou 0(druhé dva byty). Druhý má stejné složení, avšak type id zde je kódem pro Unconnected Data Item (0x00B2) s délkou 12(0x000C). Response In na konci nás informuje, na jakém řádku ve Wireshark může být nalezena odpověď.

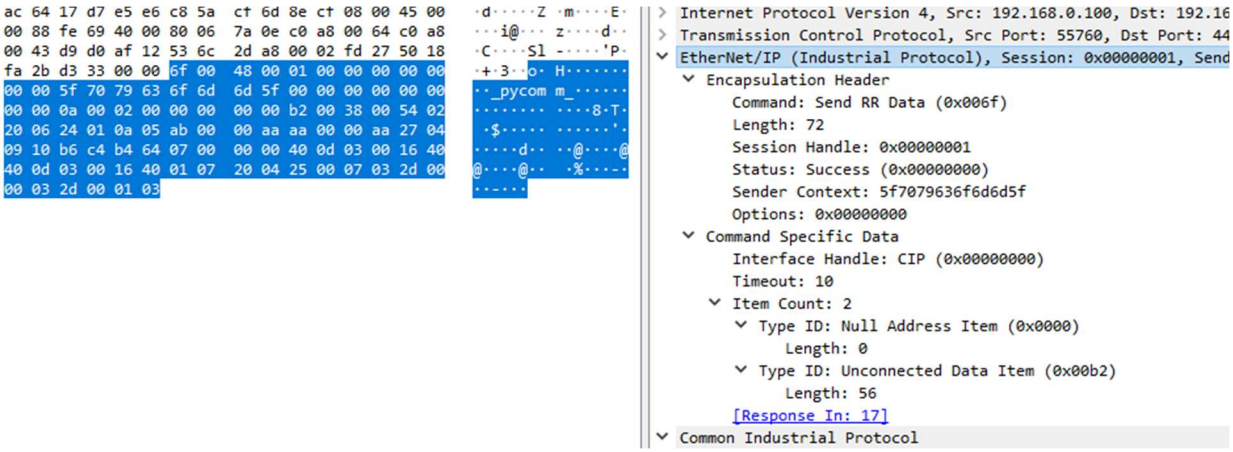


Obrázek č. 11 – Ukázka Assembly Get Attribute Single druhá část

Common Industrial Protocol na obrázku č. 11 začíná servisem, ten je určený jedním bytem a jde v podstatě o typ požadavku, který byl odeslán. To následuje velikost cesty požadavku což jsou čtyři slova neboli osm bytu. Další je Request Path Assembly, která skládá úplnou cestu Assembly instance dohromady. nyní se přesuneme k zmíněnému pro nás velmi důležitému Request Path: Assembly Instance. V prefixu a sufixu každé instance, kterou potřebujeme vyčíst jsou byty které doplňují pro ně úplnou cestu oba po dvou bytech, segment třídy a segment atributu. Uprostřed, mezi nimi je potom vždy jedna ze tří z našich instancí. Jako poslední máme Get Attribute single, který je vyplněný prázdnými byty.

5.2.1.3. Connection Mannager Forward Open

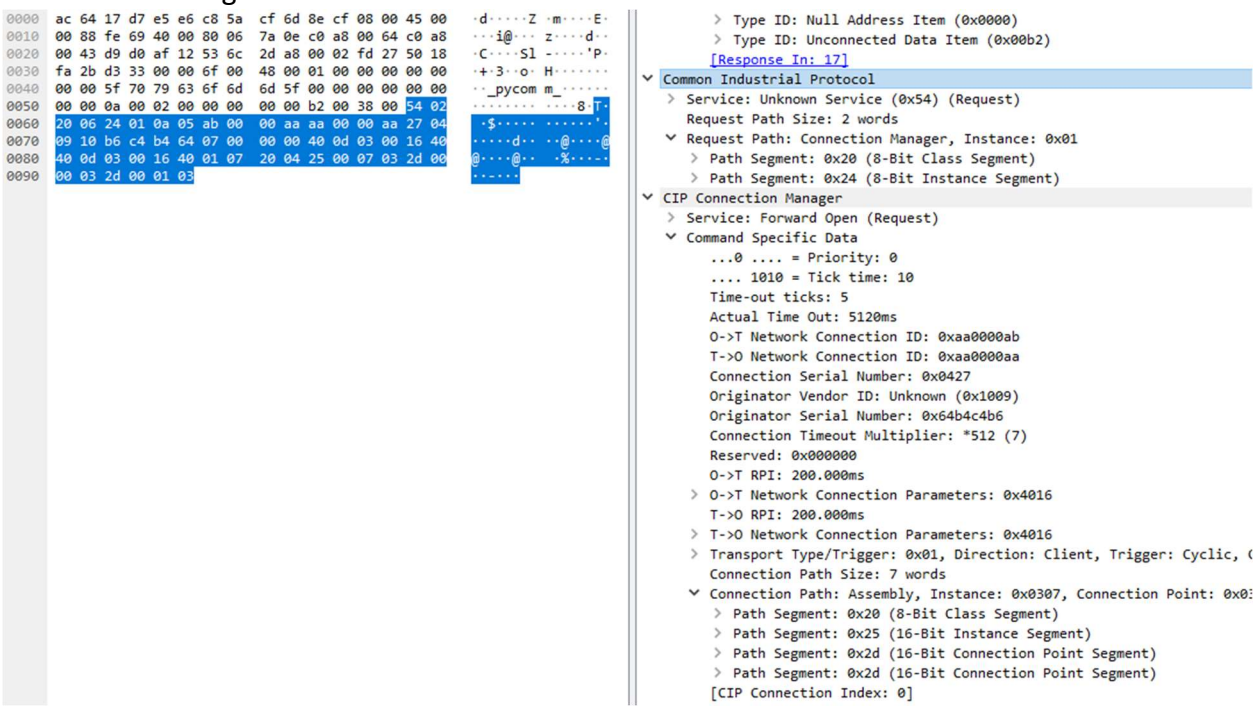
Forward open, ukázka na obrázku č. 12 a 13, je další z velmi důležitých packetů pro kompletní cyklickou komunikaci se zařízením připojeným k našemu počítači. Forward Open otevírá cestu pro možnost zapisovat data do našeho zařízení. Bez tohoto packetu by náš pokus o zapisování dat byl kompletně ignorován. Zde jsme narazili na celou řadu problému. Jelikož v CIP driveru od pycomm3 je příkaz forward open vytvořen výhradně na posílání packetů za pomoci protokolu TCP, tato funkce pro nás byla z většiny nepoužitelná a bylo jí třeba do hloubky předělat. Pro náš účel bylo třeba změnit velkou řadu dat, které forward open obsahoval. Forward open také posílá důležitá data na zpět ve formě odpovědi, proto bylo třeba tuto odpověď zachytit a tyto data vyjmout.



Obrázek č. 12 - Forward open packet část 1

Opět jako každý packet má na vrcholu EtherNet/IP hlavičku a informace o příkazu.

Hlavička a informace o příkazu jsou kromě délek daného packetu, stejná jako v Assembly Get Attribute Single.



Obrázek č. 13 – Forward Open packet část 2

V CIP a CIP connection manažer je velké množství informací, proto budou popsány jen ty, které pro nás mají větší význam. Celá CIP část protokolu pouze určuje specifikace cesty našeho příkazu. Ta je popsaná v osmi bytech. CIP connection manager již obsahuje námi nastavená data. Zde máme opět definováno, o jaký příkaz jde nyní jedním bytem 0x54 a za tímto následují command specific data. Pro nás velmi důležité je O->T a T->O network connection ID kde nastavujeme identifikační číslo naší zprávy. Tyto číslo jsou důležité z důvodu, že díky nim zařízení ví, že se snažíme komunikovat zrovna s ním. My se pokoušíme o nastavení těchto čísel, bohužel v této části je číslo odmítnuto. Další důležitá informace O->T a T->O RPI, číslo určuje, v jakém intervalu bude ono s námi a mi s ním

komunikovat. My tuto hodnotu nastavujeme na 200 milisekund, takže po každých 200 milisekundách nám pošle svůj stav, my ho zase v minimálním intervalu 200 milisekund můžeme nastavit. Na konci tohoto packetu jsou přidány informace o 3 instancích Assembly objektu (0x300, 0x301, 0x307)

T->O – označení zprávy odeslané z počítače do zařízení O->T je potom opak

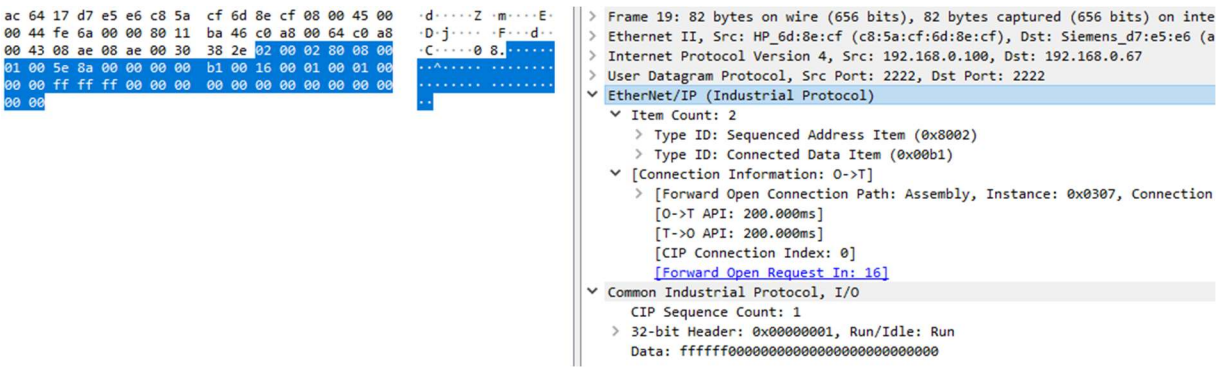
Odpověď přijde ve formě Succes: Connection Manager – Forward Open. V odpovědi je nám náš požadavek z většiny zopakován je v něm ale jedna změna. jak bylo již řečeno naše O->T Network Connection je odmítnuto a systém sám nám přidělil nové číslo(čtyři byty), které je třeba použít při jakékoli další komunikaci se zařízením.

5.2.1.4. CIP Connection O->T a T->O

Connection T->O je náš pokus o změnu dat v zařízení, což dále změní jeho nastavení. V našem případě je počítač připojen ke dvěma zařízení výstupní DQ a vstupní DI, jelikož DI od nás nebude přijímat žádné nastavení o změnu se pokoušíme se o změnu v DQ modulu, které odešle naše nastavení ven a DI ho příjme. Při tvorbě tohoto packetu bylo opět zjištěna řada problému.

CIP Driver opět nebyl uzpůsoben, pro odesílání těchto zpráv proto bylo třeba tentokrát pozměnit téměř celý CPI Driver a přizpůsobit ho našim účelům. V první řadě bylo třeba opět navázat místo TCP UDP spojení, Jelikož tentokrát se snažíme o změnu nastavení, tento úkol byl mnohem komplexnější a bylo třeba upravit mimo samotný CIP driver i řadu tříd, které CIP driver používá na odesílání zpráv. To byly například součásti knihovny pycomm3 nazvané base a socket_. Base se stará o sestavení požadavku, který odesíláme a socket_ se stará o TCP připojení.

Další problém, na který bylo naráženo je spojen s identifikačním číslem. Zpráva se bez tohoto čísla nespojí s naším zařízení, tuto skutečnost bylo třeba najít v packetu Forward Open stejně jako skutečnost, že ve Forward Open toto identifikační nelze nastavit a je třeba použít to které nám sám poskytne.



Obrázek č. 14 – Ukázka Connection I/O

Zde je výsledný packet (obrázek č. 14), většinu informací packet sbírá z již předešlého Forward Open. Lze je najít v pododstavci Connection Information. Forward Open jsou informace spojené s assembly objektem, ty se zde používají již přímo k nastavení zařízení a dále API což je jen nastavené RPI. V CIP I/O jsou již námi poslaná data. Sequence count počítá kolikrát byl odeslán příkaz, header je statický a je vždy nastaven na jedna. Data zde představují konfiguraci výstupů daného zařízení.

5.2.2. Použití python

Bylo rozhodnuto, že jazyk, který použijeme pro naše řešení, bude z řady důvodů Python. Mezi důvody patří například uživatelsky přívětivá syntaxe tohoto jazyku, široká variace uživatelských knihoven tvořených komunitou, jejich lehká instalace a použití, a neposlední řadě jeho již zaběhlé používání.

V souladu s pythonem byla také použita komunitou vyvíjena knihovna pycomm3. Jelikož zaměření této knihovny je odesílání TCP zpráv do PLC bylo však zapotřebí implementovat úpravy, které nám dovolili lépe napodobit komunikaci zařízení a PLC. Ta probíhá výhradně za pomoci UDP zprav zejména z důvodu vyšší rychlosti.

Za pomoci upraveného kódu bylo umožněno odesílání UDP zpráv a tím pádem i za použití správných bytu určit typ procesu který bude použit.

Třídy jsou z většiny pojmenovány stejně jako packety, které odesílají pro jednoduší orientaci v kódu, až na případy, kdy by název nedával smysl.

5.2.2.1. Vyčítání Assembly objektu

Funkce read_instances (obrázek č. 15) je určena k vyčtení dat z jedné instance (0x300, 0x301, 0x307). Je to funkce, patřící do třídy CIPDriverUDP a má jeden argument.

Na počátku se do vypíše číslo vložené instance do konzole pythonu.

Následující krok je vytvoření objektové proměnné request. Tento objekt v sobě ukrývá řadu hodnot používaných k nastavení funkce generic_message a již popsaných v Assembly -Get Attribute Single.

Connect_to_module kontroluje, zda je počítač připojen k TCP, pokud ne, připojí nás. Dále také spustí pycomm3 funkci register_session, ta registruje naší komunikaci.

Poté již používáme metodu generic_message převzatou z knihovny pycomm3 kde použijeme námi nastavené hodnoty. generic_message jak již název napovídá je podklad pro obecnou TCP zprávu a dále určujeme její zaměření vloženými hodnotami. Při odesílání dostaneme odpověď, která je uzavřena do proměnné odpověď (response)

V dalším kroku se odpojíme TCP, pokud je to vyžadováno a navrátíme odpověď.

```
15      def read_instances(self, instance):
16          print("read_instances = {}".format(instance))
17
18          request = ModuleRequestInfoObject(b"\x0E", b"\x04", instance, 0x03,
19                                            name="INSTANCE instance READ")
20          self.connect_to_module()
21
22          response = self.cip_driver.generic_message(
23              service=request.service_code,
24              class_code=request.class_code,
25              instance=request.instance,
26              attribute=request.attribute,
27              data_type=None,
28              connected=False,
29              name=request.name,
30              return_response_packet=True
31          )
32
33          self.is_requested_to_disconnect(False)
34          return response
```

Obrázek č. 15 – read instances Python kód

5.2.2.2. Forward Open

Forward open (obrázky č 16, 17 a 18) pro nás otevírá komunikaci a nastavuje řádu námi požadovaných specifikací. Zde je kód delší proto bude rozdělena do 3 častí. Funkce se nachází ve třídě CIPDriverUDP.

```
53      def udp_forward_open(self, instances=None, rpi=200000):
54          if self._target_is_connected:
55              return True
56          if self._session == 0:
57              raise CommError("A session must be registered before a Forward Open")
58
59          service = (
60              ConnectionManagerServices.forward_open
61          )
```

Obrázek č. 16 – udp_forward_open Python kód část 1.

Na obrázku forward open první část lze vidět, že obsahuje dva argumenty instance a rpi přednastavené na 200000.

V prvních dvou řádcích se testuje, zda již forward open neproběhl, pokud ano navrátí pravdu.

V dalších dvou řádcích bylo třeba otestovat, zda je naše komunikace registrována, pokud ne navrací chybový kód.

Poté se definuje proměnná service, která udržuje Connection Manager Forward Open.

```
        forward_open_msg = [
            PRIORITY,
            b"\x05",
            b"\xab\x00\x00\xaa",
            b"\xaa\x00\x00\xaa",
            self._cfg["csn"],
            self._cfg["vid"],
            self._cfg["vsn"],
            b'\x05',  # TIMEOUT_MULTIPLIER,
            b"\x00\x00\x00",  # reserved
            convert_hex(rpi),  # O->T RPI in microseconds
            create_network_parameters(),
            convert_hex(rpi),
            create_network_parameters(),
            b"\x01",
            self.use_required_instances(instances),
        ]
```

Obrázek č. 17 - udp_forward_open Python kód část 2.

V druhé části forward open byl vytvořen list se s hodnotami následně odesílanými jako specifikace nastavení. Tyto hodnoty jsou již vysvětleny v předešlé kapitole.

```
        response = self.generic_message(
            service=service,
            class_code=ClassCode.connection_manager,
            instance=ConnectionManagerInstances.open_request,
            request_data=b"".join(forward_open_msg),
            route_path=False,
            connected=False,
            name="forward_open",
        )

        if response:
            self._target_cid = response.value[:4]
            self._target_is_connected = True
            return True
        return False
```

Obrázek č. 18 - udp_forward_open Python kód část 1

V třetí části je znova použita Funkce generic_message s našimi daty.

V posledním bloku je zjištěno, zda byla získána odpověď, pokud ano, bylo zapsáno target_cid, což je naše Network Connection ID z předešlé kapitoly. Dále jen zapíšeme, že již připojení proběhlo a navrátíme pravdu.

Pokud se nedostaneme do tohoto bloku jen navrátíme nepravdu, to znamená, že někde nastala chyba.

5.2.2.3. Odesílání UDP zpráv

Tato funkce slouží k odesílání zpráv z počítače do zařízení. Ukázka je na obrázku č. 19. Při její vytváření bylo zjištěno, že pycomm3 nepodporuje odesílání zpráv v UDP formátu. Zde

se bylo třeba připojit do UDP soketu a také vytvořit zcela nový formát který touto schopností disponovat bude proto bylo třeba pozměnit i Forward Open. Také bylo třeba zjistit jaké data nejsou pro naše účely potřeba z důvodu odlišnosti s TCP.

Funkce se nachází v třídě EmulatorControlObject, obsahuje dva argumenty, v obou očekává byty, přičemž druhý argument má přednastavenou pevnou hodnotu a teda není třeba při použití vyplnit.

V první řádce proběhne kontrola připojení.

Druhá řádka získává Connection_ID, které se používá ke správnému identifikování používané komunikace. Tyto identifikační byty získáváme z odpovědi na Forward Open a jsou zapsána do proměnné.

Řádek tři inicializuje objektovou proměnnou request_packet s jedinou počáteční hodnotou get_sequence která navrací číslo v intervalu 1–65535.

Pak následuje inkrementace proměnné, která má na starost počítání odeslaných zpráv. V dalším kroku jí jen převedeme do hexadecimální podoby.

Na šestém řádku se potom námi zadané a zpracované hodnoty přidají do proměnné request_packet a v následném řádku se celá proměnná již odešle. Toto nám navrátí informaci, zda odeslání proběhlo úspěšně.

V posledním řádku jen posíláme dal informaci o stavu odeslání.

```
36      def send_udp_message(self, output_data: bytes, header: bytes = b"\x01\x00\x00\x00"):
37          self.connect_to_module()
38          new_connection_id = self.cip_driver.get_target_cid()
39          request_packet = SendUnitDataRequestPacketIO(self.cip_driver.get_sequence())
40
41          self.o_t_counter += 1
42          o_t_bytes = int(self.o_t_counter).to_bytes(2, "little")
43          request_packet.add(b"".join([o_t_bytes, header, output_data]))
44          send_status = self.cip_driver.send_udp_packet(new_connection_id, request_packet)
45          return send_status
```

Obrázek č. 19 - send_udp_message Python kód

5.2.2.4. Čtení UDP zpráv

Funkce Receive_udp_message (obrázek č. 20) se nachází ve třídě SocketUDP a má jeden argument v základní formě nastaven na 0.

Celá funkce je zaobalena v try except, což znamená, že pokud se kdekoli v try bloku vyskytne chyba, další krok bude přesunut do except bloku, který nás informuje o výskytu chyby. Pokud se toto stane může jít jen o jedinou chybu, a to odpojení od UDP socketu.

Druhá řádka je if blok. To znamená, že kód se posune do tohoto bloku pouze pokud byla splněna podmínka. V tomto případě podmínka je, pokud proměnná timeout není nula. V tom případě bude nastaven socket na námi určený časový limit.

Účel řádku 59, pro nás čtvrtý v pořadí je již naslouchání zprávy přicházející z UDP socketu. Tyto data jsou ve formě bytové podobě a v posledním kroku této funkce pouze navracíme získaná data.

```
55      def receive_udp_message(self, timeout=0):
56          try:
57              if timeout != 0:
58                  self.sock.settimeout(timeout)
59              data = self.sock.recv(1024)
60              print(data)
61              return data
62          except socket.error as err:
63              raise CommError("socket connection broken") from err
```

Obrázek č. 20 - receive_udp_message Python kód

5.2.2.5. Funkce run

(Obrázek č. 21)
Funkce run má na starosti v cyklu naslouchat UDP zprávám a odesílat UDP zpraví paralelně s naším hlavním kódem. Nachází se ve třídě EthernetIPCyclicCommunication.

V prvním kroku se nastaví objektová proměnná start na true. Tato proměnná se dále používá ve funkcích _set a _get.

Následně proměnná sending_rpi. Ta nastavuje interval, ve kterém budeme odesílat a získávat zprávy.

Dále je použita funkce open_udp_connection, která spustí Forward Open.

Ve čtvrtém kroku je dáno námi požadované nastavení ve formě bytu do proměnné.

Řádek 35 pro nás pátý je objkekt Thread. Ten zapne paralelně běžící kód v tomto případě _set, což je zaobalená funkce send_udp_message a jako argumenty do ní bude vložen naše proměnná out_data.

Na dalším řádku se kontroluje, zda Thread je již aktivní, pokud ne, spustí se.

Poté opět je vytvořen objekt Thread nyní s funkci _get. Ta je zaobalení funkce receive_udp_message a opět tuto funkci paralelně spustí.

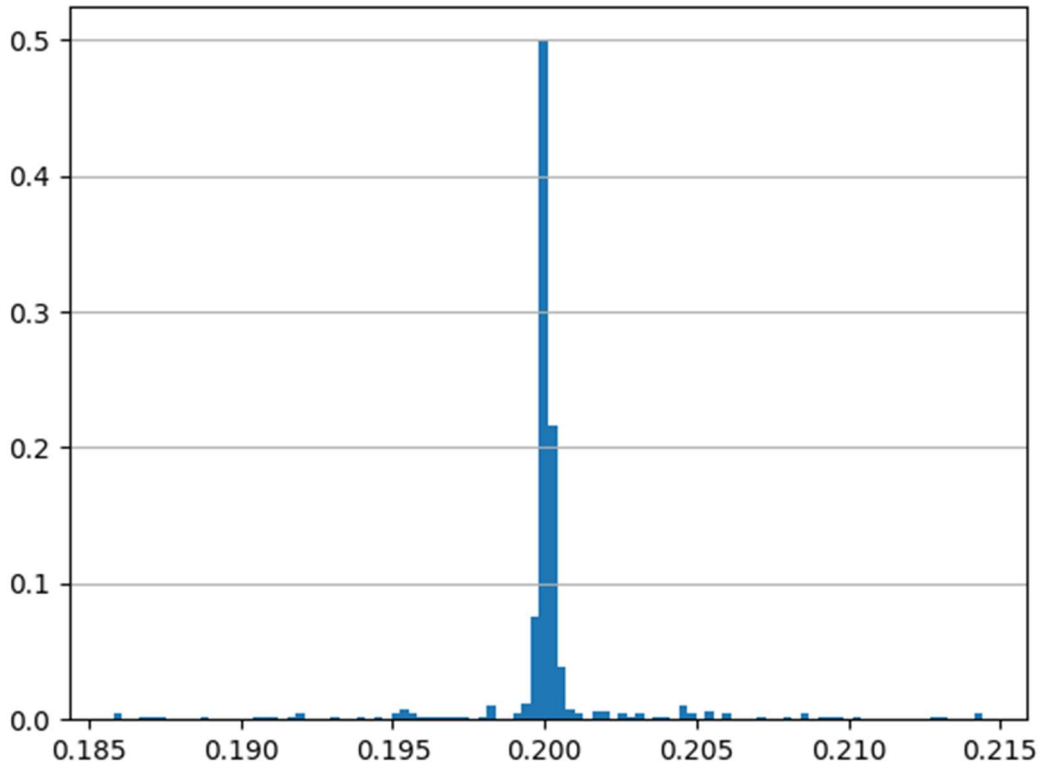
```
26      def run(self):
27          self.start = True
28
29          self.sending_rpi = 200
30          open_status = self.eip_emulator.cip_driver_udp.open_udp_connection(rpi=self.sending_rpi*100000)
31
32          out_data = bytes([0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
33                            0x00, 0x00, 0x00, 0x00])
34
35          self.thread_set = Thread(target=self._set, args=(out_data,), daemon=True, name='mtcp_set')
36          if self.thread_set.is_alive() is False:
37              self.thread_set.start()
38
39          self.thread_get = Thread(target=self._get, daemon=True, name='mtcp_get')
40          self.thread_get.start()
```

Obrázek č. 21 – multithreading run Python kód

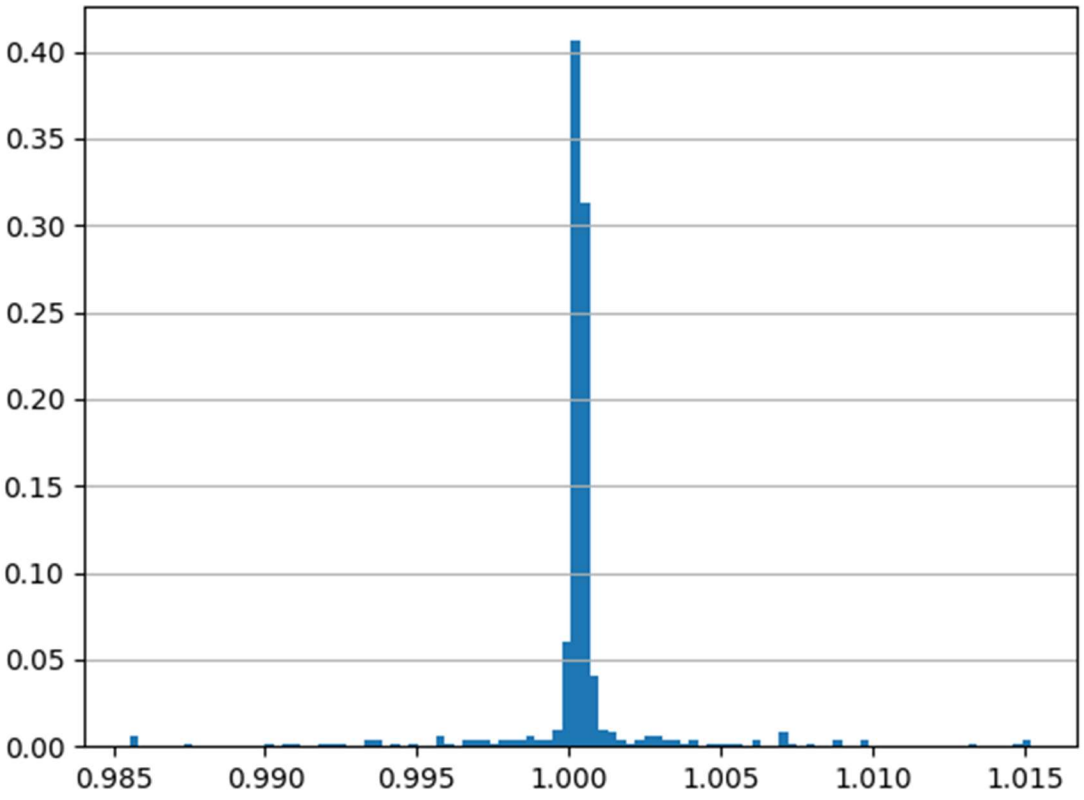
6. Výsledky a diskuse

6.1. Výsledné testování

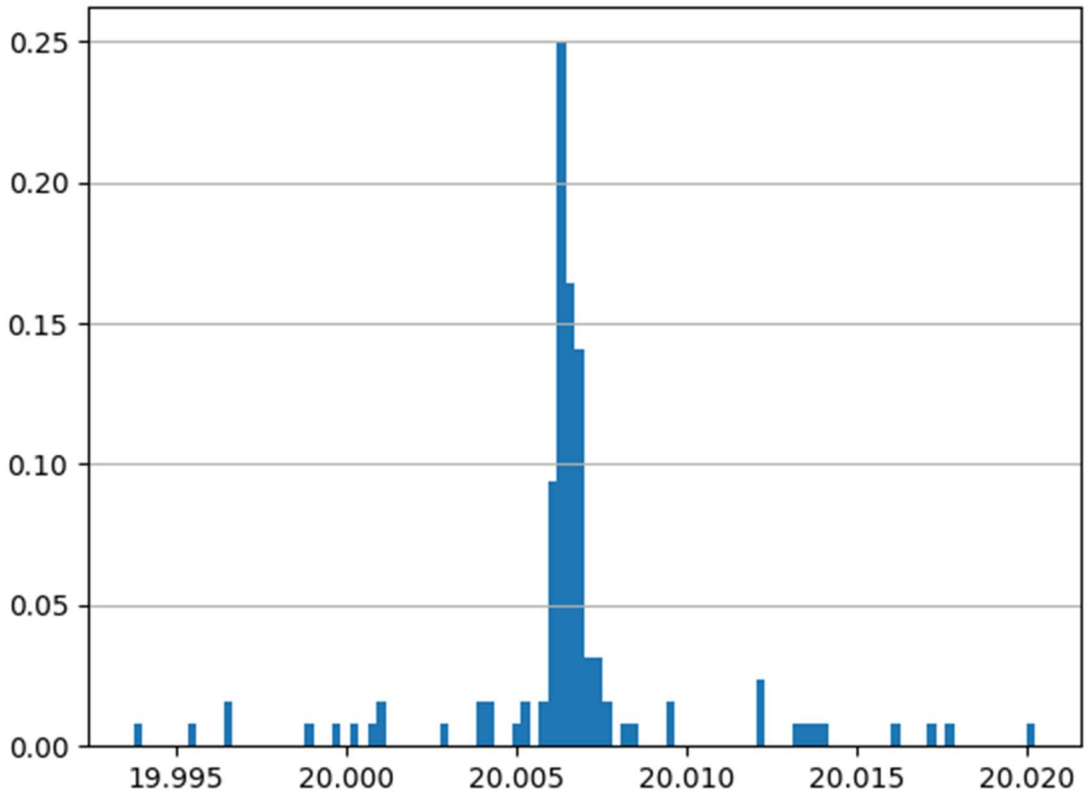
Výsledné testování proběhne pomocí zachycené komunikace v programu Wireshark, ten nám poskytne záznam, následně využijeme knihovnu rio_pca, která je zaměřena na vyhodnocování záznamů z Wireshark. Testy proběhli na DI a DQ zařízení a byly testovány jak minimální, tak maximální hranice odesílání a přijímání zpráv tak průměr.



Obrázek č. 22 – testování odesílání zpráv 0.2 s



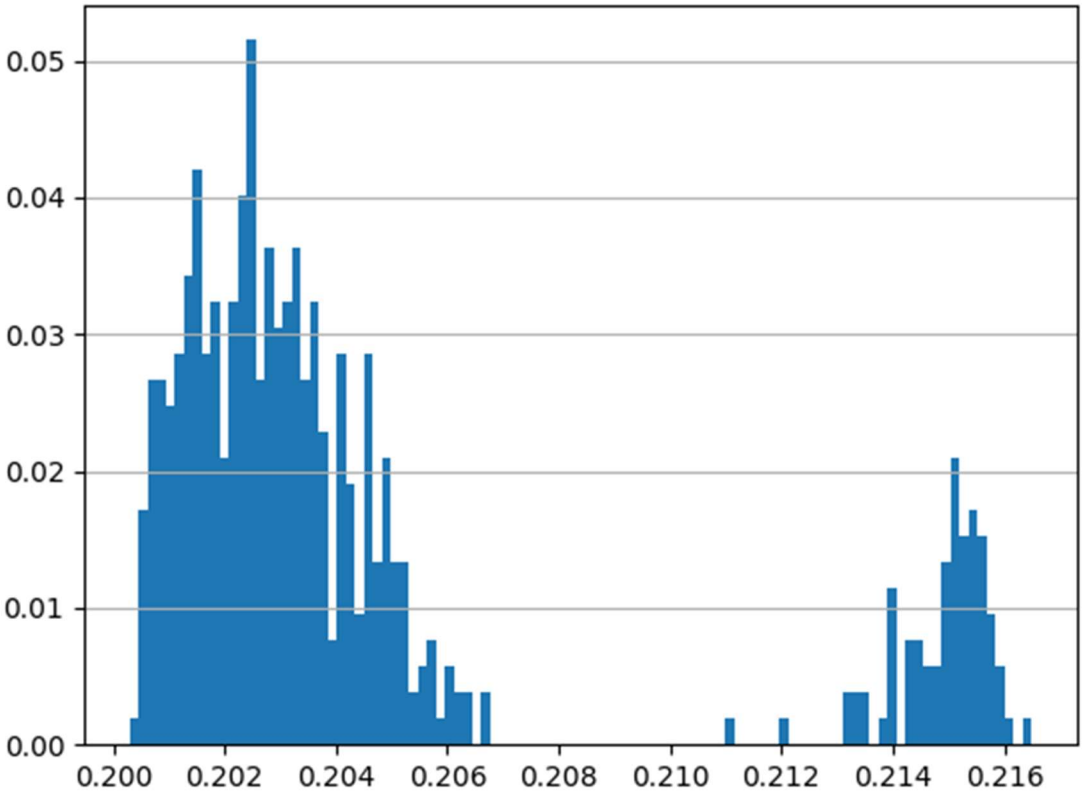
Obrázek č. 23 – testování odesílání zpráv 1 s



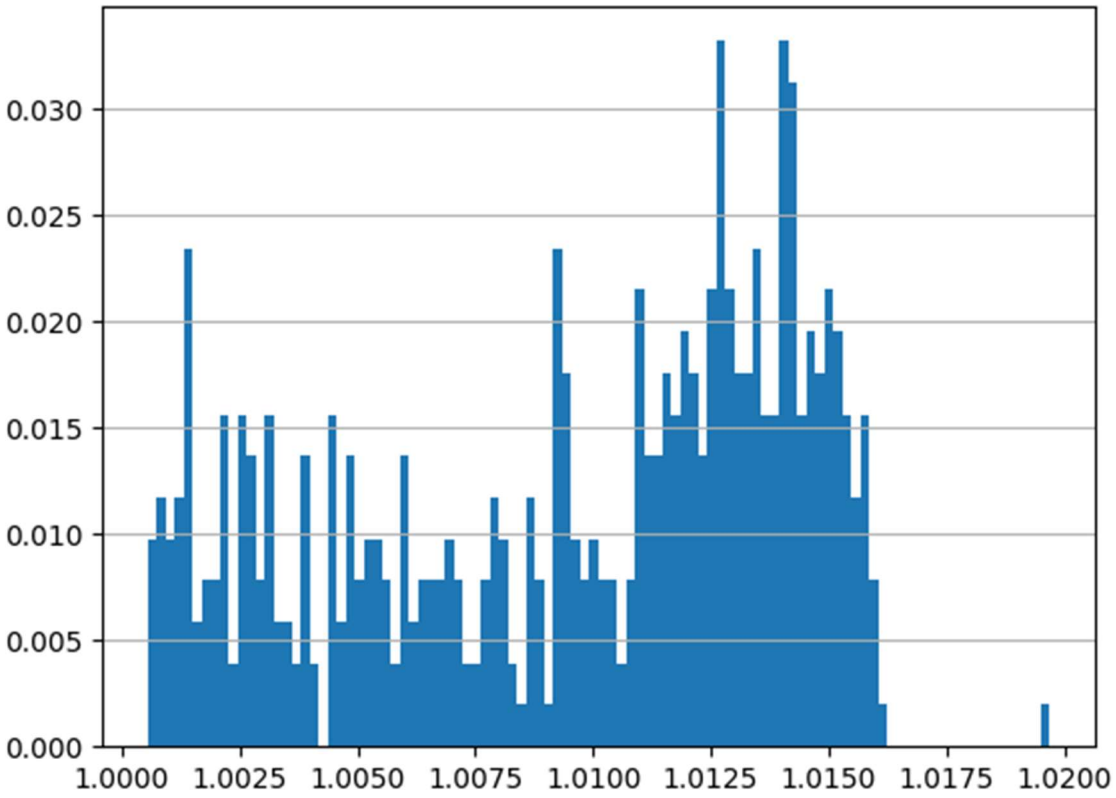
Obrázek č. 24 – testování odesílání zpráv 20 s

V obrázku č. 22, 23, a 24 lze vidět zprávy odeslané z počítače do modulů. Výsledky jsou ve všech případech velmi podobné, kdy až na pár packetů většina přišla v rozhraní -5 ms až +5 ms od našeho určeného cyklického času. Také lze ale vidět, že s rostoucím intervalem roste i zpoždění. Stále jde ale o zpoždění v rámci 10 ms.

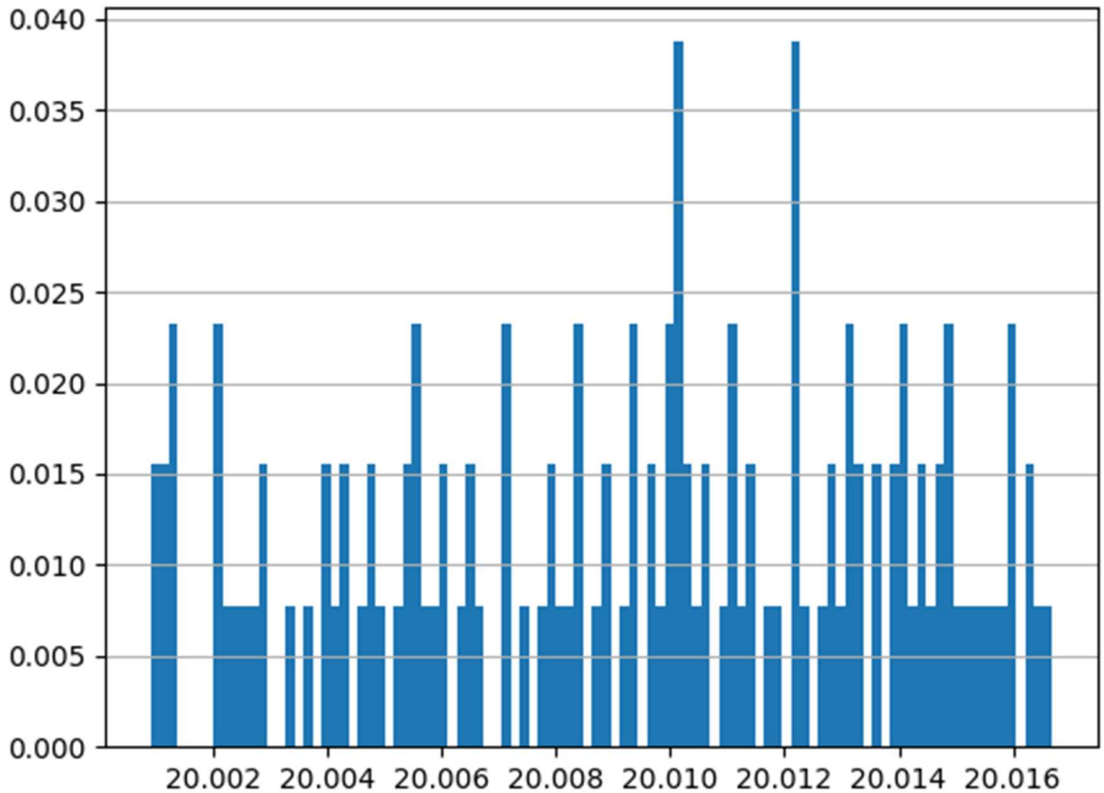
Ot 0.2s



Obrázek č. 25 – testování přijímání zpráv 0.2 s



Obrázek č. 26 – testování přijímání zpráv 1 s



Obrázek č. 27 – testování přijímání zpráv 20 s

Na obrázku č. 25, 26 a 27 lze vidět zprávy odeslané ze zařízení do počítače. Opět jde vidět změny s rostoucím intervalem, tentokrát ale v rovnoměrnosti zastoupení. U všech jde o zpoždění pohybující se okolo +16 ms.

6.2. Plány do budoucna

Hlavně z důvodu limitace časem nebylo možné emulaci dovést do dokonalosti a je zapotřebí ošetřit řadu případů, kdy by uživatel mohl zadat špatná data. Program má také své limitace, ke kterým nebyl nalezen jejich důvod, například aktuální maximální počet cyklu, který je schopný provést a to je 512. Bohužel k tomuto v den odevzdání práce nebyl problém stále vyřešen, je ale pravděpodobné, že tento limit je určen knihovnou pycomm3.

Emulaci je možné i v aktuálním stavu rozšířit o mnoho užitečných funkcí které nebylo možné implementovat opět z důvodu časové limitace. Ty čítá přijímání a vyčítání nastavení z EDS souborů, dále je třeba zapisovat vyčtené UDP zprávy do python vytvořené databáze. Dále vytvoření jak negativních, tak pozitivních testů, mezi které patří například odesílání nevalidních dat jako je RPI, špatná Connection Size a další.

7. Závěr

Tento projekt je nedílnou součásti většího softwarového řešení. Úspěšně byla navázána cyklická komunikace na protokolu EtherNet/IP. Program je schopný, jak cyklicky odesílat UDP zprávy, které mění podle požadavku nastavení zařízení, tak přijímat zprávy o jeho stavu ve zvoleném intervalu. Při tvorbě byla zjištěna řada problémů. Většina těchto problémů se stahovala k limitaci námi vybrané knihovny, která nebyla uzpůsobena pro náš účel, proto jí bylo třeba z většiny pozměnit. Další problém bylo odesílání packetů ve správné formě z důvodu jejich komplexnosti. I přes tyto obtíže se však podařilo projekt dokončit, avšak, hlavně z důvodu časového omezení je daleko od dokonalosti a je třeba dokončit řadu podfunkcí, které by emulátor měl obsahovat pro přesné napodobení, či dokonce v jistých ohledech předčení hmotného PLC.

8. Bibliografie

[1] S. P. Tubbs, Programmable Logic Controller (PLC) Tutorial, Siemens Simatic S7-1200., publicis mcd werbeagentur gmbh, 2018.

[2] anonymous, „input/output (I/O),“ [Online]. Available: https://www.techtarget.com/whatis/definition/input-output-I-O. [Přístup získán 2 3 2023].

[3] inductiveautomation, „What is SCADA?,“ 2018. [Online]. Available: https://inductiveautomation.com/resources/article/what-is-scada. [Přístup získán 10 1 2023].

[4] B. Wayand, „What Is a PLC (Programmable Logic Controller)?,“ 23 2 2020. [Online]. Available: https://www.mroelectric.com/blog/what-is-a-plc/. [Přístup získán 10 12 2022].

[5] E. A. Parr, "Computers and industrial control". Industrial Control Handbook, Industrial Press Inc., 1998.

[6] W. Bolton, Programmable Logic Controllers, Newnes, 2015.

[7] D. Chaudhari, „What is Communication Protocol?,“ [Online]. Available: https://dipslab.com/plc-communication-protocols-used-industry/. [Přístup získán 20 12 2022].

[8] T. Carlsson, „Industrial networks keep growing despite challenging times,“ 2022. [Online]. Available: https://www.hms-networks.com/news-and-insights/news-from-hms/2022/05/02/industrial-networks-keep-growing-despite-challenging-times. [Přístup získán 12 3 2023].

[9] Watelectronics, „Step by Step Procedure of PLC Programming in Industries,“ 2019. [Online]. Available: https://www.watelectronics.com/how-to-program-the-programmable-logic-controllers/. [Přístup získán 10 12 2022].

[10] „The interrupt functions of DVP series PLCs,“ [Online]. Available: https://filecenter.deltaww.com/Products/download/06/060302/Application%20Note/DELTA_IA-PLC_Interrupt_AN_EN_20150706.pdf. [Přístup získán 10 12 2022].

[11] M. Bacidore, „Should I limit programming to ladder logic or use all standards within IEC 61131?,“ 2018. [Online]. Available: https://www.controldesign.com/control/control-software/article/11310337/should-i-limit-programming-to-ladder-logic-or-use-all-standards-within-iec-61131. [Přístup získán 12 12 2022].

[12] „Basics of Structured Text,“ [Online]. Available: https://realpars.com/structured-text/. [Přístup získán 15 1 2023].

[13] M. BUDIMIR, „What are Instruction Lists (ILs) for PLC programming?,“ 2017. [Online]. Available: https://www.motioncontroltips.com/instruction-lists-ils-plc-programming/. [Přístup získán 10 12 2022].

[14] „What is Ladder Diagram Programing,“ [Online]. Available: https://instrumentationtools.com/ladder-diagram-programming/. [Přístup získán 25 12 2022].

[15] M. Budimir, „What are sequential function charts,“ 2017. [Online]. Available: https://www.motioncontroltips.com/sequential-function-charts-sfcs-plcs/. [Přístup získán 15 12 2022].

[16] R. W. Lewis, Modelling Distributed Control Systems Using, 2001.

[17] K. Gomez, „ODVA forms group for the DeviceNet of Things," 2014. [Online]. Available: https://pacetoday.com.au/odva-forms-group-for-the-devicenet-of-things/. [Přístup získán 28 12 2022].

[18] Odva, „Ethernet/IP," 2021. [Online]. Available: https://www.odva.org/technology-standards/key-technologies/ethernet-ip/. [Přístup získán 20 12 2022].

[19] ODVA, „Ethernet IP Description," 2021. [Online]. Available: https://www.odva.org/wp-content/uploads/2021/05/PUB00138R7_Tech-Series-EtherNetIP.pdf. [Přístup získán 20 12 2022].

[20] Wikipedia, „Referenční model ISO/OSI," [Online]. Available: https://cs.wikipedia.org/wiki/Referen%C4%8Dn%C3%AD_model_ISO/OSI. [Přístup získán 10 3 2023].

[21] D. Haley, „EtherNet/IP Overview.," 2014. [Online]. Available: https://slideplayer.com/slide/276977/. [Přístup získán 16 12 2022].

[22] K. Dave, „A Python Book: Beginning Python, Advanced Python, and Python Exercises," 2012. [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html. [Přístup získán 16 12 2022].

[23] B. Peterson, „Python 2.7.18, the last release of Python 2," 2020. [Online]. Available: https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html. [Přístup získán 16 12 2022].

[24] Ł. Langa, „Python Insider: Python 3.10.3, 3.9.11, 3.8.13, and 3.7.13 are now available with security content," 2022. [Online]. Available: https://pythoninsider.blogspot.com/2022/03/python-3103-3911-3813-and-3713-are-now.html. [Přístup získán 15 12 2022].

[25] pycomm3, „pycomm3 1.2.11," 2023. [Online]. Available: https://pypi.org/project/pycomm3/. [Přístup získán 10 12 2022].

[26] swagelok, „smart products EDS Files," [Online]. Available: https://www.swagelok.com/en/toolbox/software/smart-products-eds-files. [Přístup získán 15 12 2022].

[27] wireshark, „Wireshark FAQ," 2011. [Online]. Available: https://www.wireshark.org/faq.html#q1.2. [Přístup získán 20 2 2023].

[28] R. C. Martin, Clean code: handbook of agile software craftsmanship, NJ: Prentice Hall, 2009.

[29] M. Rostan, „industrial ethernet technologies," 2014. [Online]. Available: https://www.ethercat.org/download/documents/Industrial_Ethernet_Technologies.pdf.

[30] A. Chaudhari, „16 Advantages and Disadvantages of Ethernet | With Its Characteristics," 2018. [Online]. Available: https://www.csestack.org/advantages-and-disadvantages-of-ethernet-characteristics-types/.