
# 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


# SYSTÉM PRO DETEKCI ZAŘÍZENÍ V SÍTI A ROZPOZNÁNÍ POUŽITÝCH PROTOKOLŮ

SYSTEM FOR NETWORK DEVICE DETECTION AND RECOGNITION OF USED PROTOCOLS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS


AUTOR PRÁCE — Bc. Libor Sasák

AUTHOR

VEDOUCÍ PRÁCE — Ing. Eva Holasová

SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

***Student:*** Bc. Libor Sasák ***ID:*** 203175

***Ročník:*** 2 ***Akademický rok:*** 2021/22

**NÁZEV TÉMATU:**

## Systém pro detekci zařízení v síti a rozpoznání použitých protokolů

**POKYNY PRO VYPRACOVÁNÍ:**

Hlavním cílem diplomové práce je vytvořit systém pro automatické rozpoznávání jednotlivých zařízení a používaných protokolů v komunikačních sítích. Systém bude schopen práce s SNMP, Network Performance Monitoring a samotnými využívanými protokoly v síti. Dále bude systém schopen modularity z pohledu definování parametrů dalšího protokolu, který bude systémem rozpoznán. V rámci teoretické části práce student provede rozbor a porovnání přístupů (strojové učení, neuronové sítě atp.) pro identifikaci použitých protokolů v síti, jednotlivých klientů a dat potřebných k identifikaci protokolu a zařízení. V praktické části práce student navrhne a zprovozní experimentální síť, v rámci které bude provádět detekci klientů a použitých protokolů. Získané parametry budou ukládány do databáze. Pro praktickou evaluaci výsledného řešení bude použit vlastní vygenerovaný síťový provoz a také volně dostupné datové sady. K rozpoznání dat budou využity jak běžné protokoly, tak vybrané průmyslové protokoly (např. Modbus/TCP, Profinet, DNP3).

**DOPORUČENÁ LITERATURA:**

[1] CONTI, Mauro, Denis DONADEL a Federico TURRIN. A Survey on Industrial Control System Testbeds and Datasets for Security Research [online]. 1-46. ISSN 1553-877X. Dostupné z: doi:10.1109/COMST.2021.3094360.

[2] FENG, Wenbo, Zheng HONG, Lifa WU, Menglin FU, Yihao LI a Peihong LIN. Network protocol recognition based on convolutional neural network. China Communications [online]. 2020, 17(4), 125-139. ISSN 1673-5447. Dostupné z: doi:10.23919/JCC.2020.04.012.

***Termín zadání:*** 7.2.2022 ***Termín odevzdání:*** 24.5.2022

***Vedoucí práce:*** Ing. Eva Holasová

**doc. Ing. Jan Hajný, Ph.D.**
předseda rady studijního programu

## ABSTRAKT

Diplomová práca sa zaoberá rozpoznávaním použitých protokolov v sieti za pomoci strojového učenia a tvorbou systému pre tento účel. Zameriava sa na najpoužívanejšie priemyselné a bežné aplikačné protokoly a popisuje vybrané osvedčené techniky strojového učenia na ich rozpoznávanie. Prioritne sa však venuje umelým neurónovým sieťam. V skratke popisuje databázy a konkrétnu implementáciu SQLite3 použitú v konečnom riešení. V rámci práce je taktiež vytvorené a popísané virtuálne prostredie na simulovanie vybraných protokolov Modbus/TCP, DNP3, HTTPS a FTP. Časť práce je venovaná zberu, analýze a spracovaniu dát potrebných na rozpoznávanie použitých protokolov. Ďalej sa zaoberá tvorbou a testovaním modelov strojového učenia pre dané protokoly. V neposlednom rade sa práca venuje návrhu rozpoznávacieho systému a jeho implementácie s grafickým užívateľským rozhraním. Súčasťou je aj jeho testovanie a zhodnotenie predností a nedostatkov.



## KĽÚČOVÉ SLOVÁ

Modbus/TCP, Python, rozpoznávanie protokolov, SNMP, SQLite3, strojové učenie, umelé neurónové siete



## ABSTRACT

This master's thesis deals with the recognition of used protocols in a network using machine learning and the creation of a system for this purpose. It focuses on the most widely used industrial and common application protocols and describes selected well-proven machine learning techniques for their recognition. However, priority is given to artificial neural networks. It briefly describes databases and the specific implementation SQLite3 used in the final system implementation. A virtual environment for simulating selected Modbus/TCP, DNP3, HTTPS and FTP protocols is also created and described. Part of the thesis is devoted to the collection, analysis and processing of the data needed to recognize the protocols. Furthermore, it covers the creation and testing of machine learning models for the given protocols. Last but not least, the thesis is devoted to the design of the recognition system and its implementation with a graphical user interface. It also includes testing and evaluation of its advantages and limitations.



## KEYWORDS

Modbus/TCP, Python, protocol recognition, SNMP, SQLite3, machine learning, artificial neural networks

SASÁK, Libor. *Systém pro detekci zařízení v síti a rozpoznání použitých protokolů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 69 s. Diplomová práca. Vedúci práce: Ing. Eva Holasová,

# Vyhlásenie autora o pôvodnosti diela

| | |
|---|---|
| **Meno a priezvisko autora:** | Bc. Libor Sasák |
| **VUT ID autora:** | 203175 |
| **Typ práce:** | Diplomová práca |
| **Akademický rok:** | 2021/2022 |
| **Téma záverečnej práce:** | Systém pro detekci zařízení v síti a rozpoznání použitých protokolů |

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ..... .....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

# POĎAKOVANIE

Rád by som týmto poďakoval vedúcej diplomové práce Ing. Eve Holasovej za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci. Taktiež by som chcel poďakovať pánovi Ing. Karlovi Kuchařovi za užitočné pripomienky v priebehu konzultácií.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Zoznam výpisov

# Úvod

Strojové učenie v dnešnej dobe konštantne naberá na popularite, za čo je možné vďačiť najmä jeho prínosu k možnostiam, akými zariadenia vykonávajú rôznorodé činnosti, ktoré doteraz vykonával predovšetkým človek. A mnohokrát tieto činnosti zvládajú lepšie ako človek sám. Využitie v medicíne, doprave či trhovej analýze je len čerešničkou na torte celého potenciálu, ktorý sa pod pojmom „strojové učenie" skrýva. Dôležitou vlastnosťou strojového učenia však je, že algoritmy patriace to tejto kategórie nemusia byť explicitne naprogramované na to, aby vykonávali požadovanú činnosť, a teda aby vytvorili model, ktorý bude schopný dostatočne presne predpovedať vopred stanovené skutočnosti. Spravidla im na to postačí „iba" enormné množstvo kvalitných dát, vhodná konfigurácia a trocha náhody.

Ciele tejto práce mimo iné zahŕňajú vytvorenie systému na detekciu zariadení ako aj rozpoznanie použitých protokolov v sieti práve za pomoci strojového učenia. To sa týka ako priemyselných protokolov, tak aj bežných aplikačných protokolov. Takýto systém vie byť užitočným doplnením zabezpečenia priemyselnej siete, nakoľko okrem neodmysliteľného pasívneho zabezpečenia zariadení, ich komunikácie a prístupu k nim, je vhodné monitorovať každú anomáliu v tomto značne nepremennom prostredí.

Prvá kapitola sa zaoberá rozličnými priemyselnými a bežnými aplikačnými protokolmi a ich základnými vlastnosťami a parametrami. Jedným z aplikačných protokolov je protokol SNMP (Simple Network Management Protokol), ktorý je neskôr využitý na identifikáciu známych zariadení v sieti. V druhej kapitole je rozobraná podstata algoritmov strojového učenia, ktoré boli vybrané na základe vysokej úspešnosti pri rozpoznávaní sieťového prenosu. Jednak ide o algoritmus C4.5 na vytváranie rozhodovacích stromov či o všeobecne známe metódy podporných vektorov. Z dôvodu všestrannosti a zložitosti sa však najviac zaoberá technikou umelých neurónových sietí a jej hlavnými stavebnými blokmi. Tretia kapitola sa zaoberá obecnou teóriou databáz a vybranou implementáciou relačnej databázy. Štvrtá kapitola popisuje experimentálne pracovisko vytvorené pre účely tejto práce vrátane simulácií vybraných protokolov. Piata kapitola sa zaoberá zberom dát, analýzou dostupných dát vzhľadom k rozpoznávaniu použitého protokolu a komunikujúcich zariadení, spracovaním dát, ktoré dané techniky využívajú na učenie, a praktickým využitím vybraných techník strojového učenia na trénovanie modelov rozpoznávať protokol Modbus/TCP. Modely sú na záver vhodne otestované, porovnané a zhodnotené. Následne sú natrénované finálne modely pre vybrané protokoly s použitím zvoleného algoritmu. Šiesta kapitola obsahuje návrh a popis implementácie výsledného systému, vrátane popisu možností grafického užívateľského rozhrania a testovania systému ako celku.

# 1 Sieťové protokoly

Prvá kapitola je venovaná výhradne sieťovým protokolom, ktorými sa zároveň zaoberá aj celá práca. Sieťový protokol je stanovený súbor pravidiel, ktoré určujú, ako sa prenášajú dáta medzi rôznymi zariadeniami v rámci siete. Umožňuje pripojeným zariadeniam vzájomne komunikovať bez ohľadu na akékoľvek rozdiely v ich vnútorných procesoch, štruktúre alebo dizajne. Existuje mnoho rozličných sieťových protokolov, avšak vzhľadom k cieľu práce je kapitola zameraná výhradne na priemyselné a bežné aplikačné protokoly vyskytujúce sa v TCP/IP (Transmission Control Protocol/Internet Protocol) sieťach.

## 1.1 Priemyselné protokoly

Priemyselné protokoly sú spravidla využívané medzi rôznymi typmi zariadení, ktoré môžu dáta merať/zberať, požadovať ich od iných zariadení alebo ich naopak zasielať iným zariadeniam, ktoré ďalej tieto dáta spracúvajú. Naviac môžu byť tieto protokoly riadiacimi jednotkami využívané na riadenie či kontrolu zariadení v teréne či na ich konfigurovanie správcovským zariadením. Priemyselné protokoly môžu definovať spôsob komunikácie za pomoci rôznych technológií, avšak táto práca sa zaoberá protokolmi využívajúcimi sadu protokolov TCP/IP. Týmito protokolmi sú Modbus/TCP, DNP3 a Profinet.

### 1.1.1 Modbus/TCP

Modbus je žiadosť-odpoveď protokol poskytujúci vzájomnú komunikáciu typu klient-server (master-slave) pomocou dátových správ medzi zariadeniami prepojenými rôznymi typmi zberníc alebo sietí. Modbus sa od roku 1979, kedy bol vytvorený a predstavený spoločnosťou Modicon (teraz Schneider Electric), de facto stal jedným z najrozšírenejších komunikačných štandardov pre priemyslovú automatizáciu. Štandardizuje tiež protokol pre sériovú linku na výmenu Modbus požiadavkou medzi masterom a jedným alebo niekoľkými slave zariadeniami. Modbus/TCP je protokol aplikačnej vrstvy a variant protokolu Modbus využívajúci transportný protokol TCP. Server spravidla naslúcha na vyhradenom TCP porte 502. Systémy komunikujúce pomocou Modbus/TCP môžu zahŕňať nasledujúce typy zariadení:

- klientov a servery pripojených do TCP/IP siete,
- sieťové zariadenia ako bridge, router či gateway prepájajúce sieť TCP/IP a podsieť sériovej linky a tým pádom umožňujúce pripojenie klientov a servery komunikujúce na sériovej linke.

Modbus/TCP klient vie na server zasielať požiadavky s určitým kódom funkcie (viď tabuľka 1.1) a taktiež vie zasielať príkazy. U protokolu Modbus/TCP však príkazy môžu byť zasielané aj ostatnými zariadeniam, t.j. servermi. Príkazom je možné dať Modbus/TCP zariadeniu pokyn na čítanie hodnôt, zmenu hodnôt alebo ich spätné zaslanie. Príkaz obsahuje adresu zariadenia, pre ktoré je určený, zariadenie následne na príkaz reaguje. Pre viac informácií o protokole Modbus či jeho variante Modbus/TCP viď [1] a [2].

Modbus/TCP definuje jednoduchú PDU (Protocol Data Unit) nezávislú od nižších komunikačných vrstiev. Tá sa skladá z:

- Function code – označenie konkrétnej služby protokolu (pre základné kódy funkcií viď tabuľka 1.1),
- Data – prenášané dáta.

PDU je zapuzdrená v ADU (Application Data Unit), viď obrázok 1.1. Tá k PDU pridáva MBAP hlavičku (Modbus Application Protocol Header) obsahujúcu:

- Transaction ID – na synchronizáciu medzi správami klienta a serveru,
- Protocol ID – identifikátor protokolu (0 pre Modbus/TCP),
- Length – počet zvyšných bajtov v danom rámci,
- Unit ID – adresa servera (255 ak nie je použité).

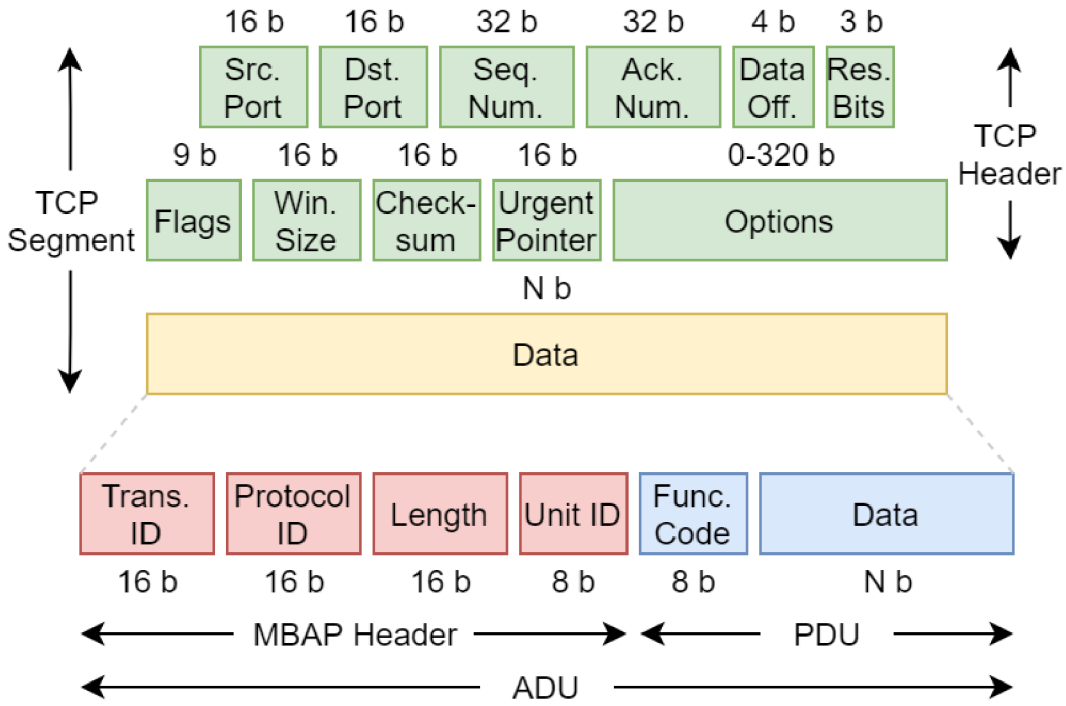
Tab. 1.1: Modbus kódy funkcií.

| **Kód** | **Názov funkcie** | **Popis** |
|---|---|---|
| 01 | Read Coils | Čítanie read-write bitov |
| 02 | Read Discrete Inputs | Čítanie read-only bitov |
| 03 | Read Holding Registers | Čítanie 16-bitových read-write registrov |
| 04 | Read Input Registers | Čítanie 16-bitových read-only registrov |
| 05 | Write Single Coil | Zápis jedného bitu |
| 06 | Write Single Register | Zápis jedného 16-bitového registru |
| 15 | Write Multiple Coils | Zápis viacerých bitov |
| 16 | Write Multiple Registers | Zápis viacerých 16bitových registrov |

### Dostupné nástroje a knižnice

Pre simuláciu komunikácie protokolom Modbus/TCP sú dostupné napr. nástroje:

- ModRSsim2 [3] – Modbus master simulátor pre komunikáciu po RS-232 zbernici a pomocou TCP/IP,
- EasyModbus [4] – Modbus master/slave simulátory a .NET knižnica pre TCP, UDP (User Datagram Protocol) a RTU (Remote Terminal Unit) komunikáciu,
- pymodbus [5] – úplná implementácia protokolu Modbus v jazyku Python.




Obr. 1.1: Modbus ADU/PDU.

### 1.1.2 Distributed Network Protocol 3

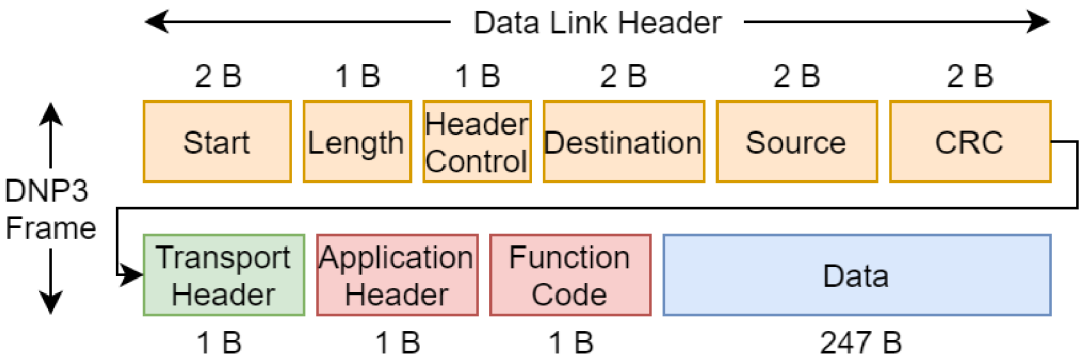
DNP3 (Distributed Network Protocol 3) je súbor komunikačných protokolov používaný zariadeniami v systémoch procesovej automatizácie. Je využívaný najmä v elektrárenských a vodárenských spoločnostiach. Bol vyvinutý na komunikáciu medzi riadiacimi zariadeniami a rôznymi typmi zariadení na zber dát. Zohráva kľúčovú úlohu v systémoch SCADA (Supervisory Control And Data Acquisition), kde je využívaný najmä na komunikáciu medzi riadiacim centrom a vzdialenými koncovými jednotkami alebo inteligentnými elektronickými zariadeniami. DNP3 používa výraz „outstation" na označenie vzdialených zariadení v teréne a výraz „master" na označenie zariadení v riadiacich centrách. Poskytuje pravidlá na vzájomnú výmenu dát a riadiacich príkazov. Pre viac informácií o protokole DNP3 viď [6].

DNP3 definuje tri komunikačné vrstvy:

- aplikačnú
  - prijíma správu od užívateľskej aplikácie pozostávajúcu z funkčného kódu a dátových objektov slúžiacich na získanie správnych informácií od vzdialenej stanice,
  - správu rozdelí na fragmenty o maximálnej veľkosti 2048 bajtov,
  - ku fragmentu pridá hlavičku o veľkosti 1 bajtu a kód funkcie taktiež o veľkosti 1 bajtu,

- pseudo-transportnú,
  - fragment rozdelí na segmenty o maximálnej veľkosti 249 bajtov,
  - ku segmentu pridá hlavičku o veľkosti 1 bajtu,
- linkovú vrstvu,
  - ku segmentu pridá hlavičku o veľkosti 10 bajtov a kontrolný súčet o veľkosti 32 bajtov,
  - výsledkom je rámec s maximálnou veľkosťou 292 bajtov prenesený na nižšiu vrstvu.

DNP3 je primárne určený pre sériové linky, avšak je použiteľný aj v sieťach IP, kde je zabalený do TCP alebo UDP paketov. To znamená, že sa v takom prípade všetky tri spomenuté vrstvy prakticky nachádzajú nad transportnou vrstvou TCP/IP modelu. Na obrázku 1.2 je možné vidieť jednoduchú štruktúru jedného DNP3 rámca obsahujúceho prvý segment fragmentu. Zapuzdrenie v rámci TCP segmentu je rovnaké ako pri Modbus/TCP (viď obrázok 1.1). Pre podrobnejší rozbor DNP3 rámca viď [7].




Obr. 1.2: DNP3 rámec, prvý segment maximálnej veľkosti.

### Dostupné nástroje a knižnice

Pre simuláciu komunikácie protokolom DNP3 sú dostupné napr. nástroje:

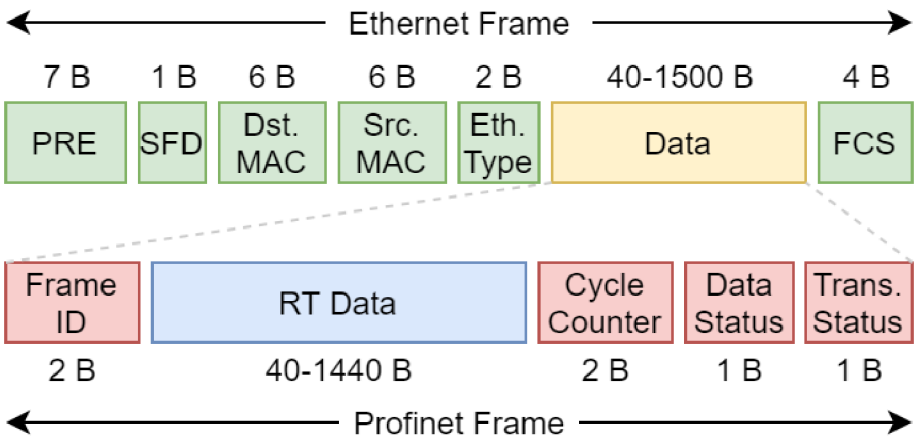
- DNP3 Protocol Development Bundle [8] – outstation/master simulátor a knižnica pre komunikáciu seriálovou zbernicou, protokolom TCP a UDP pre Windows a Linux,
- pydnp3 [9] – Python wrapper väčšiny z tried knižnice opendnp3 [10], ktorá je prenosnou a škálovateľnou implementáciou protokolu DNP3 v jazyku C++,
- CDOAN-DNP3 [11] – Windows testovací nástroj a simulátor protokolu DNP3.

### 1.1.3 Profinet

Profinet je komunikačný štandard pre automatizáciu zastrešený organizáciou PROFIBUS & PROFINET International, ktorý pôvodne vychádzal z historicky staršieho štandardu Profibus. Je navrhnutý na zber údajov a na ovládanie zariadení v priemyselných systémoch využívajúc priemyselný Ethernet. Protokol Profinet je určený na rýchlu výmenu údajov v reálnom čase a riadi sa modelom „provider-consumer" (poskytovateľ-spotrebiteľ). Definuje jednak výmenu údajov medzi riadiacimi jednotkami „IO-Controllers" a zariadeniami „IO-Devices", ako aj nastavenie parametrov a diagnostiku, ktorú vykonáva tzv. „IO-Supervisor".

Profinet používa transportný protokol TCP (prípadne UDP) tam, kde prenos dát nie je časovo kritický, napr. na konfiguráciu, parametrizáciu či diagnostiku. Využíva porty 34962, 34963 a 34964 v závislosti od toho, či ide o unicast, multicast alebo prenos kontextového managera, ktorý vytvára a spravuje aplikačné a komunikačné vzťahy medzi IO-Controllers a IO-Supervisors. Na prenos dát v reálnom čase však Profinet nepoužíva IP a teda ani TCP či UDP, dáta sú adresované len za pomoci fyzických adries a protokol tak operuje priamo nad Ethernetom (EtherType je v takom prípade hodnota 34962). Pre viac informácií o protokole Profinet viď [12]. Na obrázku 1.3 je možné vidieť Profinet rámec prenosu dát v reálnom čase. Rámec je zapuzdrený priamo v Ethernet rámci a pozostáva z:

- Frame ID – identifikácia rámca,
- Real-Time Data – dátová časť rámca,
- Cycle Counter – inkrementovaná hodnota na určenie rýchlosti aktualizácie dát,
- Data Status, Transfer Status – informácie týkajúce sa celkovej platnosti údajov, redundancie a hodnotenia diagnostického stavu.




Obr. 1.3: Profinet rámec.

## 1.2 Aplikačné protokoly

Aplikačné protokoly sú protokoly aplikačnej vrstvy TCP/IP modelu, ktoré využívajú transportné protokoly nižšej vrstvy (najmä TCP a UDP) a prenášajú už konkrétne dáta aplikácie. Pre účel rozpoznávania protokolov bolo vhodné vybrať bežné aplikačné protokoly, ktorých správanie sa podobá správaniu spomenutých priemyselných protokolov na sieti. Táto kapitola je preto zameraná na niekoľko spojovo orientovaných príkaz-odpoveď protokolov založených na modeli klient-server. Jedna sekcia je špeciálne venovaná protokolu SNMP (Simple Network Management Protocol), nakoľko bol využitý v konečnom riešení.

### 1.2.1 Hypertext Transfer Protocol

HTTP (Hypertext Transfer Protocol) je štandardný sieťový protokol, ktorý je najdôležitejšou súčasťou dátovej komunikácie pre World Wide Web, kde hypertextové dokumenty obsahujú odkazy na iné zdroje s jednoduchým prístup pre užívateľa. Komunikácia prebieha tak, že klient odošle serveru správu s HTTP požiadavkom a server poskytujúci zdroje či vykonávajúci iné funkcie vráti klientovi správu s odpoveďou obsahujúcou kód stavu o dokončení požiadavku a prípadný požadovaný obsah. Kódy stavu môžu byť informačné (100-199), označujúce úspech (200-299), presmerovanie (300-399) alebo chybu klienta (400-499) či serveru (500-599).Vzhľadom k jeho úlohe využíva spoľahlivý transportný protokol TCP, avšak vo špecifických prípadoch môže využívať aj UDP. UDP rovnako využíva aj nadchádzajúca hlavná verzia protokolu, HTTP/3, kde spoľahlivosť zabezpečuje protokol QUIC (Quick UDP Internet Connections).

Vo verzii 1.0 protokolu HTTP každá požiadavka vyžadovala vlastné TCP spojenie, čo predstavovalo problém v opozdení, kvôli vykonávaniu TCP handshaku pred každou novou požiadavkou. Vo verzii 1.1 bol tento problém čiastočne vyriešený možnosťou zaslať viacero požiadavkov vrámci jedného spojenia. Vo verzii 2, momentálne najrozšírenejšej verzii protokolu HTTP, sú využívané obojsmerné toky, pričom súbežné toky s jedným klientom predstavujú len jediné TCP spojenie. Umiestnenie zdroja, nad ktorými klient žiada vykonanie určitej akcie v HTTP požiadavku, je jednoznačne špecifikované pomocou URL (Uniform Resource Locator). Je to reťazec znakov, ktorý bežne obsahuje protokol, domény a cestu s názvom požadovaného zdroja. Najvyužívanejším rozšírením protokolu HTTP, ktoré umožňuje vytváranie šifrovaných spojení, je HTTPS (Hypertext Transfer Protocol Secure). Spojenie je šifrované pomocou kryptografického protokolu TLS (Transport Layer Security) alebo jeho predchodcu SSL (Secure Sockets Layer). HTTP podporuje implicitnú konfiguráciu TLS/SSL, používa port 80 pre spojenia HTTP a port 443 pre HTTPS.

HTTP v požiadavkoch využíva tzv. metódy. Metódou sa nazýva označenie akcie, ktorá má byť serverom vykonaná. Je zasielaná v HTTP požiadavku s príslušnými údajmi, napr. s URL. Štandardné metódy je možné vidieť v tabuľke 1.2. Pre viac informácií o protokole HTTP viď [13].

Tab. 1.2: HTTP metódy.

| **Metóda** | **Popis** |
|---|---|
| GET | používa sa na načítanie zdroja zo servera |
| POST | používa sa na odoslanie údajov na server na vytvorenie nového zdroja, údaje sú vo forme textu a nie sú viditeľné v adrese URL |
| PUT | podobná ako POST, akurát s cieľom kompletne aktualizovať už existujúci zdroj |
| DELETE | používa sa na odstránenie zdroja |
| TRACE | požaduje spätné zaslanie prijatej požiadavky (na ladenie) |
| OPTIONS | umožňuje klientovi zistiť metódy HTTP (prípadne ďalšie funkcie) podporované webovým serverom |
| CONNECT | vytvára obojsmernú komunikáciu so serverom, napr. TCP tunel na vytvorenie bezpečného SSL/TLS spojenia (HTTPS) |
| PATCH | podobná ako PUT, avšak je používaná len na čiastočnú aktualizáciu zdroja |

### 1.2.2 File Transfer Protocol

FTP (File Transfer Protocol) je štandardný sieťový protokol používaný na prenos počítačových súborov medzi klientom a serverom. FTP používa samostatné riadiace TCP spojenie, pre ktoré server naslúcha na porte 21. Pre dátové TCP spojenie je využívaný port 20. FTP taktiež využíva URL syntax na špecifikovanie umiestnenia zdroja a využíva návratové hodnoty podobne ako HTTP. Návratové hodnoty majú však odlišný význam. Môžu byť predbežné pozitívne o iniciovaní akcie (100-199), pozitívne o dokončení akcie (200-299), pozitívne požadujúce ďalšie informácie (300-399), dočasné negatívne (400-499) a permanentné negatívne (500-599), ktoré majú odrádzať klienta od opätovného požadovania akcie. Taktiež sú definované aj chránené odpovede (600-699), ktoré po správnom dekódovaní spadajú do už spomenutých kategórií.

Spojenie s FTP serverom môže byť inicializované v dvoch režimoch:

- pasívny – riadiace aj dátové spojenie je inicializované klientom z náhodných portov,

- aktívny – klient inicializuje len riadiace spojenie s náhodným portom *N* a dátové spojenie inicializuje server na port *N+1*, na ktorom klient naslúcha,
  - môže byť problémom pre firewall a NAT (Network Address Translation), keďže spojenie nie je inicializované od klienta, je preto len málo používaný.

FTPS (File Transfer Protocol Secure) variant protokolu FTP rozšíreného o šifrovanie podporuje dve SSL/TLS konfigurácie:

- explicitnú – kedy sú používané rovnaké porty ako pri nešifrovanom spojení, šifrovanie si v tom prípade musí klient explicitne vyžiadať,
- implicitnú – je využívaný port 990 pre riadiace spojenie a port 989 pre dátové spojenie, je považovaná za zastaranú a využíva sa len ojedinele.

FTP definuje 70 štandardizovaných príkazov, ktoré je možné zaslať na server, avšak v praxi sú často využívané príkazy neštandardizované. Niektoré z bežne využívaných príkazov je možné vidieť v tabuľke 1.3. Pre viac informácií o protokole FTP viď [14].

Tab. 1.3: FTP príkazy.

| **Príkaz** | **Popis** |
|---|---|
| GET | prenos súboru zo servera |
| MGET | prenos viacerých súborov zo servera |
| PUT | prenos súboru na server |
| MPUT | prenos viacerých súborov na server |
| DELE | zmazanie súboru |
| LIST | žiadosť o zoznam súborov |
| USER | zadanie mena používateľa |
| PASS | zadanie hesla užívateľa |
| QUIT | ukončenie spojenia |

### 1.2.3 Simple Mail Transfer Protocol

SMTP (Simple Mail Transfer Protocol) je štandardný sieťový protokol používaný na odosielanie a prijímanie elektronických správ, emailov. Bežne sa používa len na zasielanie pošty na server, na získanie pošty zo serveru sa zvykne používať POP3 (Post Office Protocol 3) alebo novší IMAP (Internet Message Access Protocol) či iné proprietárne protokoly v závislosti od poskytovateľa služby. Na transportnej vrstve sa vzhľadom k potrebe spoľahlivého prenosu používa protokol TCP, avšak porty 25 a 587 využívané týmto protokolom sú alokované aj pre protokol UDP. Použitie UDP však nie je bežné. Port 25 je dnes využívaný prednostne na komunikáciu medzi servermi a z dôvodu zamedzenia spamu je často blokovaný firewallmi. Port 587 sa

používa na explicitne šifrovanú SSL/TLS komunikáciu klienta so serverom, rovnako ako aj napr. neoficiálny port 2525, či iné porty. Oficiálny port 465 môže byť taktiež použitý (s implicitným šifrovaním), avšak je považovaný za zastaraný. SMTP používa rovnaké rozsahy návratových kódov ako FTP (viď podkapitola 1.2.2), okrem predbežných pozitívnych a chránených odpovedí. Najpoužívanejšie príkazy protokolu SMTP vrátane príkazov jeho rozšírenia, tzv. ESMTP (Extended Simple Mail Transfer Protocol), je možné vidieť v tabuľke 1.4. Pre viac informácií o protokole SMTP viď [15].

Tab. 1.4: SMTP príkazy.

| **Príkaz** | **Popis** |
|---|---|
| HELO | inicializácia konverzácie so serverom uvedením jeho domény |
| EHLO | obdoba HELO ale pre ESMTP |
| AUTH | autentizácia klienta na serveri zadaním prihlasovacích údajov |
| MAIL (FROM) | zadanie zdrojovej emailovej adresy, začiatok prenosu správy |
| RCPT (TO) | zadanie emailovej adresy príjemcu |
| DATA | žiadosť serveru o zaslanie obsahu správy, po prijatí povolenia klient riadok po riadku zasiela dátum, hlavičku odosielateľa, predmet, hlavičku príjemcu, prílohy a text správy, zadávanie ukončí zaslaním znaku bodky |
| VRFY | žiadosť o overenie existencie adresáta na serveri |
| EXPN | žiadosť o overenie existencie zoznamu adresátov na serveri |
| TURN | obrátenie rolí medzi klientom a serverom bez potreby inicializovať nové spojenie |
| RSET | oznámenie o predčasnom ukončení prenosu správy |
| HELP | žiadosť o list príkazov, ktoré server podporuje |
| NOOP | overenie, že server odpovedá |
| STARTTLS | inicializácia TLS handshaku pre zabezpečenie spojenia so serverom, v prípade úspechu resetuje spojenie |
| QUIT | ukončenie konverzácie so serverom |

### 1.2.4 Simple Network Management Protocol

SNMP je protokol používaný na správu a monitorovanie zariadení v TCP/IP sieťach. Protokol SNMP je bežne zabudovaný v sieťových zariadeniach, ako sú smerovače, prepínače, servery, firewally a bezdrôtové prístupové body. SNMP zvyčajne využíva protokol UDP s portom 161. SNMP používa dátové štruktúry zvané MIB (Management Information Base) predstavujúce súbor hierarchicky usporiadaných informácií,

na ktoré sa dá dotazovať či ich konfigurovať. Existujú ako štandardné, tak aj proprietárne MIB platné len pre zariadenia či softvér konkrétneho výrobcu. Každý objekt v MIB má svoj OID (Object Identifier), ktorý jednoznačne identifikuje objekt v MIB hierarchii.

Na zariadeniach podporujúcich protokol SNMP beží softvér, ktorý je podľa funkcie buď manažér alebo agent. SNMP manažér beží na zariadení, ktoré spravidla plošne zbiera dáta z monitorovaných sieťových zariadení. To vykonáva za pomoci SNMP dotazov s konkrétnymi OID. SNMP agent beží na monitorovanom zariadení a zhromažďuje údaje o rôznych metrikách, ako je využitie CPU, využitie šírky pásma alebo miesta na disku. Na základe dotazov SNMP manažéra dotazované informácie zisťuje a zasiela späť. Pre viac informácií o protokole SNMP viď [16].
Základné príkazy, ktorými disponuje manažér, sú uvedené v tabuľke 1.5. Agent spravidla používa príkazy v tabuľke 1.6

Tab. 1.5: SNMP príkazy manažéra.

| **Príkaz** | **Popis** |
|---|---|
| Get Request | získanie hodnoty premennej alebo zoznamu premenných |
| Set Request | vykonanie konfigurácie alebo príkazov |
| GetNext Request | zistenie hodnôt ďalšieho záznamu v hierarchii MIB |
| GetBulk Request | získanie veľkých tabuliek údajov vykonaním viacerých príkazov GetNext Request |
| SNMP Inform | potvrdzuje prijatie správy trap |

Tab. 1.6: SNMP príkazy agenta.

| **Príkaz** | **Popis** |
|---|---|
| SNMP Response | odpoveď na príkaz |
| SNMP Trap | nevyžiadané správy trap od agentov SNMP spravidla upozorňujú SNMP manažéra, že nastala nejaká významná udalosť (napr. chyba) |

# 2 Strojové učenie

Jednou z najefektívnejších metód na rozpoznávanie protokolov sieťového prenosu je využitie modelov vytvorených za pomoci algoritmov strojového učenia. Strojové učenie ako podoblasť umelej inteligencie skúma využitie počítačových algoritmov, ktoré sa len na základe vstupných dát a skúseností môžu postupne zlepšovať, a tým prakticky napodobňovať učenie ľudí. Tieto algoritmy vytvárajú model, ktorý má za účel rozhodovať či predpovedať. Strojové učenie sa najčastejšie využíva napr. na rozpoznávanie obrazu a reči, predpovedanie v doprave, detekciu spamu, malvéru a podvodov, obchodovanie na burze, lekársku diagnostiku, automatický preklad jazyka či v autonómnych vozidlách. Strojové učenie sa primárne delí na učenie s učiteľom a bez učiteľa, podľa toho, či sú využívané označené dáta alebo nie. Nie je ojedinelé aj zmiešané využitie týchto dvoch prístupov, kedy sa využívajú označené aj neoznačené dáta, spravidla s veľkou prevahou tých neoznačených.

### Učenie s učiteľom

Pri učení s učiteľom sú používané označené dáta, čiže dáta, ktoré obsahujú ich správnu klasifikáciu či predpoveď. Počas učenia sú upravované váhy jednotlivých dát či iných od nich odvodených hodnôt až do momentu, kedy daný model dosiahne určitú požadovanú presnosť. Patria sem napríklad neurónové siete, naivná Bayesova metóda, lineárna regresia, logistická regresia, náhodný les či metóda podporných vektorov.

### Učenie bez učiteľa

Pri učení bez učiteľa sú využívané neoznačené dáta, pričom je snaha za pomoci príslušných učiacich algoritmov v dátach objaviť skryté vzory či vzťahy medzi jednotlivými dátami. Vďaka schopnosti objavovať podobnosti a rozdiely v informáciách sú ideálnym riešením na analýzu dát, strategické delenie zákazníkov alebo rozpoznávanie obrazu a reči. Medzi tieto algoritmy patria napríklad metódy zhlukovej analýzy či opäť niektoré typy neurónových sietí.

### Učenie posilňovaním

Ďalším prístupom podobným učeniu s učiteľom je tzv. učenie posilňovaním, ktoré však nevyužíva žiadne vzorové dáta. Miesto dát algoritmy odmeňujú pozitívne správanie, čiže prideľujú pozitívne hodnoty žiadúcim akciám/javom, a trestajú správania negatívne prideľovaním negatívnych hodnôt nežiadúcim akciám/javom. Tzv. agent sa teda na základe ohodnotenia závislého od jeho akcií snaží iteratívne upraviť svoje správanie, aby z dlhodobého hľadiska dosiahol čo najväčšiu odmenu. Tento prístup

je často využívaný v hernom priemysle, na správu zdrojov vo firmách, na personalizované odporúčania a v robotike. Najbežnejšími algoritmami sú napríklad SARSA (State–action–reward–state–action), Q-Learning či Deep Q-Networks.

#### Výber techník

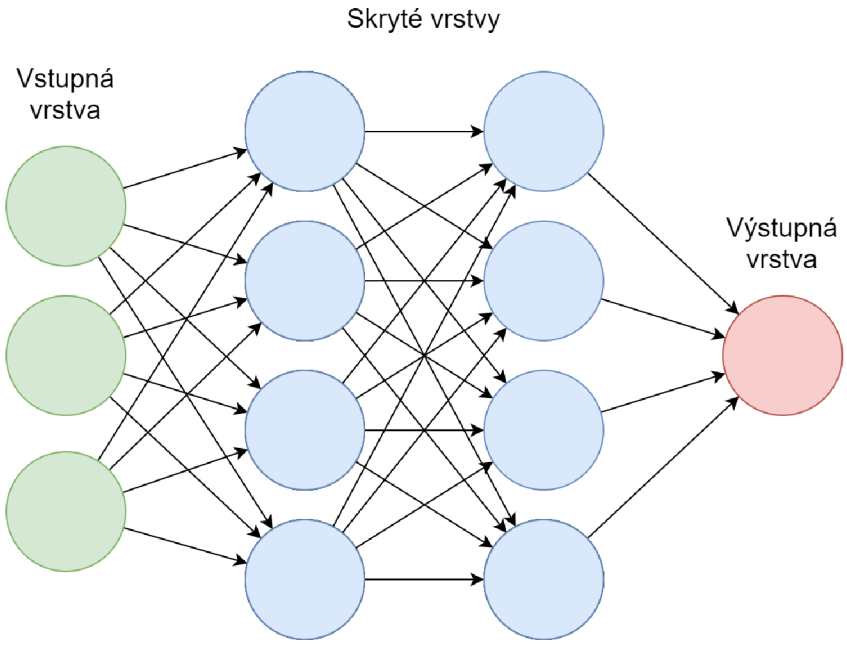
Na riešenie problému rozpoznávania protokolov v sieti boli vybrané tri techniky strojového učenia. Jednak boli vybrané neurónové siete pre ich všeobecne známy význam a úspešnosť pri detegovaní anomálií v sieťovom prenose. Algoritmus C4.5 a metóda podporných vektorov boli vybraté ako dva najpresnejšie techniky na klasifikáciu sieťového prenosu podľa publikácie [17].

## 2.1 Neurónové siete

Neurónové siete sú viacvrstvové siete prepojených neurónov a ako algoritmus strojového učenia sú používané najmä na klasifikovanie, predpovedanie či hľadanie skrytých vzorov v dátach. Svojou štruktúrou napodobňujú ľudský mozog a z určitého hľadiska aj učenie sa živého organizmu. Vrstvy neurónovej siete sa delia na vstupnú vrstvu, ktorej neuróny na vstup prijímajú vstupné hodnoty modelu, výstupnú vrstvu, ktorej jeden alebo viac výstupov predstavuje výstup z modelu, a skryté vrstvy, ktorými sú všetky vrstvy medzi vstupnou a výstupnou vrstvou. Najvyužívanejšie neurónové siete sú dopredné, čiže tie, v ktorých neuróny a ich prepojenia netvoria cykly. Neuróny v jednotlivých vrstvách prijímajú vstupy predchádzajúcich neurónov (až na neuróny vstupnej vrstvy) a svoj výstup poskytujú neurónom nasledujúcej vrstvy (okrem neurónov výstupnej vrstvy). Príklad doprednej neurónovej siete je možné vidieť na obrázku 2.1. Neurón, ako základná stavebná jednotka neurónových sietí, môže byť interpretovateľný ako jednoduchá funkcia, ktorá sa skladá z váh, hodnoty zaujatosti a tzv. aktivačnej funkcie.

### 2.1.1 Váhy a zaujatosť

Každý neurón prideľuje každému svojmu vstupu určitú váhu, ktorou je vstup násobený. Tým je možné určiť prospešnosť vstupu pre daný neurón. Ďalej neurón obsahuje hodnotu zaujatosti, ktorá je pripočítaná k sume vážených vstupov a umožňuje tak ovplyvniť vstup a tým pádom aj výstup aktivačnej funkcie. Váhy a hodnoty zaujatosti neurónov sú v procese učenia v rámci spätnej propagácie upravované tak, aby bola dosiahnutá čo najnižšia hodnota stratovej funkcie (viď podkapitola 2.1.3).




Obr. 2.1: Príklad neurónovej siete s dvoma skrytými vrstvami.

### 2.1.2 Aktivačná funkcia

Aktivačná funkcia, ako už jej názov napovedá, rozhoduje o tom, či má byť neurón aktivovaný, a teda či je jeho vstup do neurónovej siete prospešný a v akej miere. Úlohou aktivačnej funkcie je transformovať súčet vážených vstupov neurónu na výstupnú hodnotu a tým zaviesť do neurónovej siete nelinearitu. Práve vďaka aktivačným funkciám je možné riešiť zložitejšie problémy, nakoľko bez nich by model neurónovej siete bez ohľadu na počet vrstiev prakticky degradoval na lineárny regresný model. Aktivačnou funkciou môže byť napr. binárna funkcia, lineárna funkcia alebo nelineárne funkcie ako Sigmoid, Tanh či ReLU (Rectified Linear Unit) funkcia.

#### Binárna funkcia

Binárna funkcia transformuje neurón na tzv. binárny klasifikátor. Takáto funkcia vážený vstup porovná s určitou prahovou hodnotou, ak je táto hodnota väčšia ako prah, výstupom bude hodnota 1, ak je menšia ako prah, na výstupe bude 0. Binárna funkcia je pri komplexnejších problémoch nevyužiteľná.

#### Lineárna funkcia

Použitím lineárnej funkcie získame výstup priamoúmerný vstupu, pochopiteľne teda do modelu nezavádza nelinearitu, výstup bude bez ohľadu na počet vrstiev vždy lineárnou funkciou vstupu, čím celá vrstvová štruktúra prestáva dávať zmysel.

#### Sigmoida

Je bežne používaná v modeloch, ktorými je predpovedaná pravdepodobnosť, keďže má výstupný rozsah práve 0 až 1. Funkcia poskytuje hladký gradient a teda zabraňuje „skokom" vo výstupných hodnotách. Hodnoty gradientu (viď podkapitola 2.1.3) sú významné však len pre rozsah vstupných hodnôt funkcie -3 až 3, pri vyšších hodnotách sa hodnota gradientu príliš blíži k nule, sieť sa prestáva učiť a trpí tzv. problémom miznúceho gradientu.

#### Tanh

Funkcia tanh je podobná sigmoide. Prvý rozdiel je vo výstupnom rozsahu od -1 do 1, čiže platí, že čím je vstup väčší/pozitívnejší, tým bližšie bude výstupná hodnota k 1, a čím je vstup menší/negatívnejší, tým bližšie bude výstupná hodnota k -1. Jej výstup je orientovaný okolo nuly, preto je možné výstupné hodnoty ľahko mapovať ako silne negatívne, neutrálne alebo silne pozitívne. Tanh poskytuje omnoho strmejší gradient a taktiež trpí miznúcim gradientom pre vstupné hodnoty mimo približného rozsahu -4 až 4.

#### ReLU

ReLU neuróny deaktivuje v prípade, že je výstup lineárnej transformácie menší ako 0, pre pozitívne hodnoty sa správa ako lineárna funkcia. Má teda derivovanú funkciu a teda umožňuje spätnú propagáciu. Zároveň je výpočtovo efektívna a urýchľuje konvergenciu gradientného zostupu ku globálnemu minimu stratovej funkcie. Napriek tomu, že je ReLU v dnešnej dobe masívne využívaná, má jedno potenciálne významne negatívum, a to tzv. problém umierajúcej ReLU. Ten spočíva v tom, že pri záporných vstupných hodnotách je hodnota gradientu nulová. Z tohto dôvodu sa počas procesu spätnej propagácie niektoré neuróny neaktualizujú a teda môže nastať prípad, že neuróny nebudú nikdy aktivované a stanú sa z nich tzv. mŕtve neuróny. Naviac sa u ReLu všetky záporné vstupné hodnoty okamžite stanú nulovými, čo znižuje schopnosť modelu správne sa rozhodovať či trénovať.

### 2.1.3 Spätná propagácia, stratová funkcia a gradientný zostup

Spätnou propagáciu sa v neurónových sieťach nazýva proces postupného upravovania váh a hodnôt zaujatosti neurónov, aby bola dosiahnutá čo najnižšia hodnota stratovej funkcie. Hodnoty sú upravované pomocou algoritmu gradientného zostupu.

Gradientný zostup je optimalizačný algoritmus na hľadanie lokálneho minima diferencovateľnej funkcie. Podstatou je opakovanie krokov v opačnom smere, ako je

gradient (smer najväčšej zmeny) funkcie v aktuálnom bode, nakoľko ide o smer najstrmšieho zostupu. Na určenie gradientu funkcie sa využíva jej derivovaná funkcia. Pre viac informácií o gradientnom zostupe viď [18].

Stratová funkcia v jednoduchosti hovorí, aký dobrý je daný model vzhľadom k poskytnutým dátam, pričom model je najlepšie naučený, ak je strata nulová. Stratová funkcia na vyhodnocovanie straty využíva hodnoty z výstupu modelu a požadované hodnoty z označených dát. Typy stratovej funkcie sú:

- regresná – využívaná, ak je výstupom predpoveď určitej spojitej hodnoty,
- binárnej klasifikácie – pre rozhodovanie medzi dvoma triedami,
- viac-triednej klasifikácie – pre rozhodovanie medzi viacerými triedami.

Pre viac informácií o umelých neurónových sieťach viď [19].

## 2.2 C4.5 algoritmus

C4.5 je algoritmus používaný na generovanie rozhodovacieho stromu. Umožňuje využitie nie len diskrétnych, ale aj spojitých atribútov na rozhodovanie, a to tak, že rozdeľuje možné hodnoty atribútu na diskrétnu množinu intervalov. Dokáže pracovať s dátami s chýbajúcimi hodnotami. Každému atribútom prideľuje vlastnú váhu, teda hodnotu jeho významu pri rozhodovaní. C4.5 si v každom uzle stromu vyberie jeden z ostávajúcich atribútov (na základe ktorých sa v danej vetve zatiaľ nerozhodoval), ktorý má najnižšiu entropiu, čiže najvyšší normalizovaný informačný zisk, a teda najefektívnejšie rozdelí množinu vzoriek datasetu na nevyvážené podmnožiny.

Pri učení v každom uzle existujú tri špeciálne prípady:

- všetky zvyšné vzorky datasetu patria do rovnakej triedy – vytvorí sa listový uzol rozhodovacieho stromu, ktorý hovorí, že daná trieda je výstupná,
- ani jeden zo zvyšných atribútov neposkytuje žiadny informačný zisk – vytvorí sa rozhodovací uzol o úroveň vyššie podľa očakávanej hodnoty triedy,
- vyskytla sa inštancia predtým nevidenej triedy – opäť sa vytvorí rozhodovací uzol o úroveň vyššie na základe očakávanej hodnoty.

Po vytvorení stromu ho znova celý prejde a pokúsi sa odstrániť vetvy, ktoré nepomáhajú, tým, že ich nahradí listovými uzlami. Pre viac informácií viď [20].

## 2.3 Metódy podporných vektorov

Cieľom metód podporných vektorov je nájsť hyperrovinu v n-rozmernom priestore, ktorá jednoznačne rozdeľuje, a teda klasifikuje dátové body. Hľadaná rovina by mala mať maximálnu rezervu, t.j. maximálnu vzdialenosť medzi dátovými bodmi oboch tried. Rozmer (dimenzia) priestoru závisí od počtu atribútov datasetu. Podporné

vektory sú dátové body, ktoré sú blízko k hyperrovine a ovplyvňujú jej polohu a orientáciu. Čiže sú to body, na základe ktorých je zostavený výsledný model. Vymazaním podporných vektorov sa zmení poloha hyperroviny.

Výhodami metódy podporných vektorov sú:

- efektivita vo vysoko-dimenzionálnych priestoroch a v prípade, keď je počet dimenzií väčší ako počet vzoriek,
- pamäťová efektivita – používa totiž podmnožinu trénovacích bodov (podporných vektorov) na výsledné rozhodovanie,
- univerzálnosť.

Pre viac informácií o metódach podporných vektorov viď [21].

# 3 Databázy

Dáta, ktoré bude výsledný systém používať a taktiež ktoré bude na základe svojej činnosti produkovať, je potrebné mať vhodným spôsobom uložené. V prvom rade tak, aby k nim bol jednoduchý a rýchly prístup, a zároveň aby boli vhodne organizované pre prípad, že v nich bude potrebné určité dáta vyhľadať. Na to systému poslúži databáza. Databáza je v informatike organizovaný súbor dát uložených elektronicky v určitom súborovom systéme (v prípade menších databáz) alebo na samostatnom hardvéri či cloudovom úložisku. Databáza je zvyčajne riadená systémom DBMS (Database Management System), ktorý slúži zároveň ako rozhranie medzi databázou a jej užívateľom či programom, ktorý ju obsluhuje či používa. DBMS spravidla poskytuje operácie umožňujúce čítanie, zápis, organizovanie, analýzu či logické prepojenie dát a celkovú správu databázy a uľahčuje dohľad nad databázou jej monitorovaním. Pre viac informácií o databázach viď [22].

Existuje mnoho rôznych druhov a implementácií databáz, avšak z dôvodu verzatility a spoľahlivosti výsledného riešenia bude využívaná relačná databáza s jazykom SQL (Structured Query Language).

### Relačná databáza a SQL

Relačná databáze je založená na tzv. relačnom modeli, ktorý predstavuje intuitívny a jednoduchý spôsob reprezentácie dát v tabuľkách. Každý riadok v tabuľke predstavuje jeden záznam s jedinečným identifikátorom, tzv. kľúčom. Stĺpce tabuľky predstavujú jednotlivé atribúty záznamov. Pre každý z atribútov záznam obsahuje spravidla jednu hodnotu, či v prípade, že sa nejedná o kľúčový či explicitne povinný atribút, nemusí obsahovať žiadnu (resp. obsahuje hodnotu NULL). Pre viac informácií o relačných databázach viď [23].

SQL je štandardizovaný programovací jazyk používaný na správu relačných databáz. Slúži na komunikáciu s relačným DBMS za pomoci dotazov. Dotaz vždy začína SQL príkazom reprezentujúcim operáciu, ktorá má byť vykonaná. Základnými SQL príkazmi sú:

- SELECT – na získanie dát s tabuľky (všetkých alebo podľa výberu),
- CREATE – vytvorí novú databázu alebo tabuľku,
- DELETE – zmaže záznam z tabuľky,
- INSERT INTO – pridá záznam záznam do tabuľky,
- UPDATE – vykoná zadané zmeny v záznamoch.

Pri vytváraní tabuľky je atribútu pridelený typ a prípadné obmedzenie (angl. constraint). Najpoužívanejšie dátové typy sú:

- BIT – jeden bit (1 alebo 0),
- INT – celé čísla,

- FLOAT – čísla s desatinnou čiarkou,
- VARCHAR – text premennej veľkosti (NVARCHAR pre Unicode),
- DATETIME – dátum a čas.

Základnými obmedzeniami, ktoré je možné priradiť atribútu, sú:

- NOT NULL – stĺpec nemôže mať hodnotu NULL,
- UNIQUE – všetky hodnoty v stĺpci sú rôzne,
- PRIMARY KEY – je kombináciou NOT NULL a UNIQUE,
- DEFAULT - predvolená hodnota stĺpca, nahradí hodnotu NULL.

Pre viac informácií o jazyku SQL viď [24].

### SQLite3

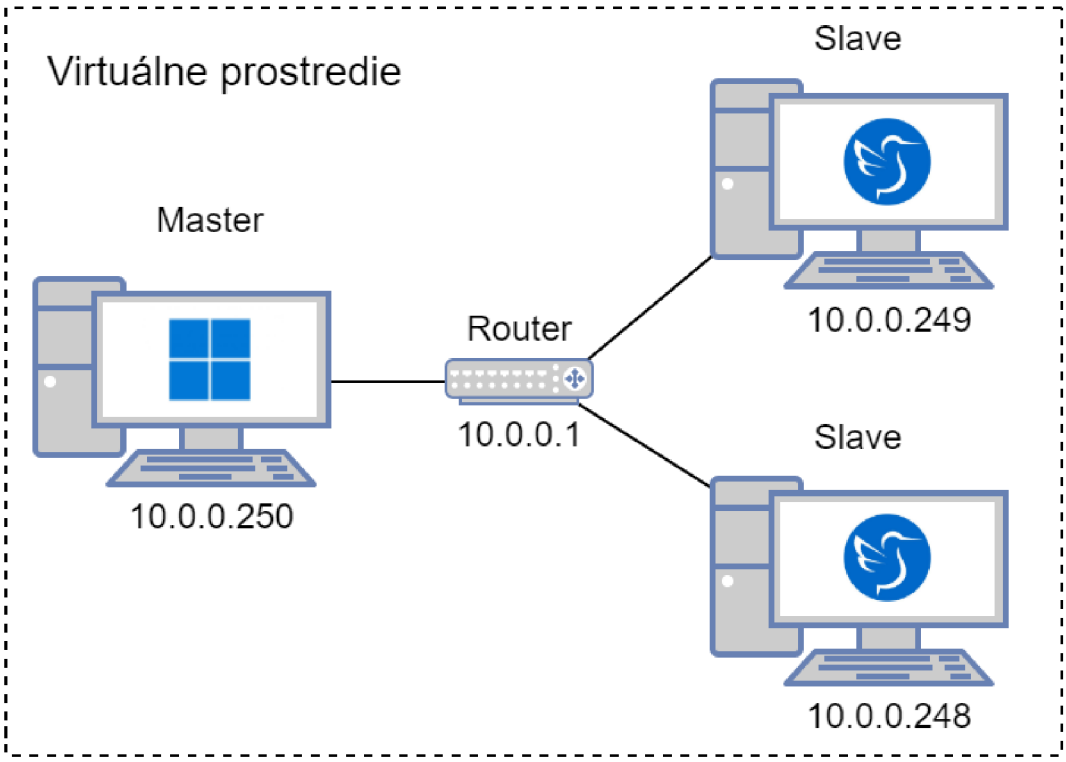
Ako už bolo spomenuté, existuje mnoho rôznych implementácií databáz, ako aj samotných SQL databáz. Pre výsledný systém však bola na základe prieskumu dostupných riešení vybratá open-source knižnica SQLite3 [25], nakoľko ňou inicializovaná databáza nevyžaduje samostatný hardvér a celá je obsiahnutá v jednom *.db* súbore. Riešenie je naviac malé a rýchle. Jedná sa o vysoko spoľahlivý a plnohodnotný SQL databázový engine, ktorý je zároveň svetovo nepoužívanejší.

Všetky dáta uložené v SQLite3 databáze patria do jednej z nasledujúcich tried:

- NULL – prázdna hodnota,
- INTEGER – v závislosti od veľkosti hodnoty až 8-bajtové celé číslo,
- REAL – 8-bajtové číslo s desatinnou čiarkou,
- TEXT – textový reťazec uložený pomocou kódovania danej databázy,
- BLOB – údaje sú uložené v podobe, v akej poli predané na vstup do databázy (spravidla binárne dáta).

# 4 Experimentálne pracovisko

Experimentálne pracovisko pozostáva z jedného slave zariadenia naslúchajúceho na TCP porte 502 a dvoch master zariadení, ktoré budú slave zariadeniu zasielať požiadavky. Spomedzi dostupných simulačných nástrojov boli vybrané implementácie EasyModbus, konkrétne .NET implementácia slave zariadenia a Python master implementácia. Na virtuálizáciu potrebných zariadení bol použitý Oracle VM VirtualBox, zariadenia boli prepojené virtuálnym routerom s DHCP (Dynamic Host Configuration Protocol) serverom a bez prístupu na internet. Model pracoviska je možné vidieť na obrázku 4.1.




Obr. 4.1: Experimentálne pracovisko.

Na virtuálnom slave zariadení s operačným systémom Windows 11 bola spustená aplikácia EasyModbusServerSimulator, ktorá je súčasťou balíka EasyModbus. Súčasťou balíka je aj implementácia klienta (master), tá síce umožňuje využitie na skúšobné účely, ale neposkytuje možnosť pokročilejšej simulácie. Preto bola využitá knižnica EasyModbusTCP.PY [26]. Pre virtuálne master zariadenia s odľahčeným operačným systémom Lubuntu bol vytvorený klientský skript využívajúci spomenutú knižnicu, ktorý si na úvod vyžiada IP adresu slave zariadenia a následne začne konštantne naväzovať TCP spojenia z náhodného portu a vykonávať náhodne 3 až 10 z nasledujúcich dostupných operácií:

- Read – Coils, Discrete Inputs, Holding Registers, Input Registers,
- Write – Single Coil, Single Register, Multiple Coils, Multiple Registers.

Vždy po vykonaní operácií je spojenie uzavreté. Náhodný je index pre čítanie-/zápis (v rozmedzí 0 až 65 533), bitové hodnoty k zápisu, celočíselné hodnoty k zápisu (v rozmedzí 0 až 65 535) a prípadný počet týchto hodnôt (v rozmedzí 1 až 123). Všetky uvedené rozmedzia sú minimálne/maximálne pre využitú implementáciu mastera.

#### Zachytávanie paketov prvotnej Modbus/TCP simulácie

Na spustenie skriptu na master zariadeniach je potrebné najprv nainštalovať vybraný balík easymodbus pomocou príkazu „pip3 install easymodbus". Najprv je na servery spustená aplikácia Modbus slave zariadenia a taktiež je spustené zachytávanie prenosu v nástroji Wireshark [27] s filtrom „ip.src == 10.0.0.0/24 && ip.dst == 10.0.0.0/24", ktorý zaistí odfiltrovanie všetkých paketov, ktoré sa netýkajú prenosu medzi zariadeniami simulácie (mimo ich privátnu sieť). Ďalej sú spustené klientské skripty a je do nich zadaná IP adresa servera. Po potvrdení začína nadmerný prenos dát viditeľný v nástroji Wireshark. Takto bolo odchytených a uložených vyše 1,3 milióna paketov, ktoré boli použité pri testovaní klasifikačných modelov (viď podkapitola 5.5).

## 4.1 Simulácia priemyselných a bežných aplikačných protokolov

Na základe zhodnotenia po testovaní rozpoznávacích modelov (viď podkapitola 5.7) bol následne zvolený odlišný prístup k simulácii protokolov. Simulované by malo byť konkrétne využitie protokolov, nie náhodný chaos požiadavkou a odpovedí. Simulované boli protokoly Modbus/TCP, DNP3, HTTPS a FTP. Dáta simulácie boli zachytávané a následne uložené do *.pcapng* súboru.

#### Modbus/TCP simulácia

Vzhľadom k možnosti prispôsobenia Modbus/TCP serveru bol za pomoci knižnice pymodbus vytvorený Python skript simulujúci Modbus/TCP server. Tento server udržuje 3 celočíselné hodnoty v holding registroch a každú sekundu ich rôznymi spôsobmi náhodne pozmení. Toto konkrétne správanie hodnôt by v reálnom svete mohlo byť napr. výstupom meraní rôznych fyzikálnych veličín v nestálom prostredí. Z dôvodu kompatibility bola na základe rovnakej knižnice vytvorená aj simulácia Modbus/TCP klienta. Tá opäť každú sekundu číta uvedené tri holding registre.

Spojenie je však tentokrát udržované bez prerušenia. Jedinou výnimkou je prerušenie a obnovenie spojenia po jednej hodine, aby bolo jednoduchšie zaviesť do učenia modelu aj vstupy s neznámym tcp.direct. Tými budú práve pakety z prvej hodiny simulácie bez paketov naviazania spojenia (bez TCP handshaku).

### DNP3 simulácia

Podobným spôsobom ako u Modbus/TCP bola za pomoci knižnice pydnp3 a exemplárnych oustation/master skriptov vytvorená simulácia protokolu DNP3. Až na to, že outstation (server) a master (klient) pracujú s hodnotami s desatinnou čiarkou a master sa na tieto hodnoty nedotazuje priamo pomocou adries ale pýta si všetky zmenené hodnoty. K tomu ešte pravidelne zasiela outstation zariadeniu dátum a čas. Z dôvodu nedostatku verejne dostupných datasetov k protokolu DNP3 bola vytvorená aj druhá simulácia, ktorá využívala nástroj CDOAN-DNP3 s licenciou vyžiadanou konkrétne pre účely tejto práce. Simulácia periodicky každé 2 sekundy volala funkciu „READ, Analog Output" na čítanie analógových output hodnôt, každú sekundu „READ, Class 123" na čítanie všetkých troch tried input dát a každých 5 sekúnd „Test Function For Link" na zistenie stavov spoja.

### HTTPS simulácia

Pre demonštráciu funkčnosti rozpoznávania protokolov aj na šifrovaných dátach bola vybraná miesto protokolu HTTP bezpečná obdoba HTTPS. Serverom je open-source riešenie nginx [28]. Na koreni servera je možné nájsť offline nástroj na testovanie rýchlosti spojenia spustiteľný pomocou jedného tlačítka a ďalej sú na 5 rôznych adresách dostupné rôzne obrázky. Klient je simulovaný pomocou knižnice selenium [29]. Opakovane sa náhodne dotazuje na webovú stránku s testovacím nástrojom a dostupné obrázky v polonáhodných intervaloch. Pri prvom krátkom spustení simulácie bola šanca, že simulovaný užívateľ klikol na tlačítko spustenia testu rýchlosti spojenia. To však generovalo príliš veľa jednotvárneho prenosu, preto bola simulácia po určitom čase spustená znova avšak bez možnosti testovania, aby sa predišlo prílišnej zaujatosti pri učení modelu na týchto dátach.

### FTP simulácia

FTP serverom je open-source riešenie FileZilla Server [30]. Klient simulovaný za pomoci vstavanej Python knižnice ftplib. Server a klient si na žiadosť klienta medzi sebou náhodne vymieňajú za pomoci príkazov RETR a STOR opäť 5 rôznych obrázkov v polonáhodných intervaloch.

### Analýza simulácií

Simulácie boli spustené naraz medzi jedným klientom a dvoma servermi a bežali rovnaký čas. Modbus/TCP simuláciu bolo možné v zachytených paketoch identifikovať pomocou portu 502 a bolo zozbieraných 36618 paketov. DNP3 simulácia bola identifikovateľná portom 20000 a bolo zozbieraných 32462 paketov. HTTPS simuláciu bolo možné rozpoznať pomocou portu 443 a bola spomedzi simulácií najobjemnejšia, ako veľkosťami paketov, tak aj ich počtom 259319. Simulácia FTP pozostávala z dvoch častí, z čoho jedno riadiace spojenie rozpoznateľné pomocou portu 2121, a druhé dátové spojenie, ktoré predstavovalo zvyšný TCP prenos medzi daným klientom a serverom.

# 5 Tvorba rozpoznávacích modelov

Táto časť práce sa bude venovať vytváraniu modelov na rozpoznávanie protokolu Modbus/TCP za pomoci strojového učenia v jazyku Python. Jednotlivými krokmi budú identifikácia problému a vstupných dát, zber dát, rozbor dostupných dát vzhľadom k využiteľnosti pri rozpoznávaní, spracovanie dát a nakoniec trénovanie a testovanie modelov.

## 5.1 Identifikácia problému a vstupných dát

Prvými fundamentálnymi krokmi v procese strojového učenia a tvorby modelu, bez ohľadu na zvolenú techniku, sú identifikácia problému a identifikácia vstupných dát. Výsledný model bude riešiť problém interpretovateľný jednou otázkou s odpoveďou áno/nie: „Patria zachytené pakety protokolu Modbus/TCP?". Vstupom do tohto modelu, na ktorý sa dá pozerať ako na funkciu, bude $n*c$ hodnôt, kde $n$ označuje počet atribútov, a $c$ počet paketov, z ktorých hodnoty pochádzajú. Tieto hodnoty môžu byť buď priamo extrahované z paketu, alebo odvodené z iných hodnôt. Výstupom modelu bude jediná hodnota odpovedajúca na uvedenú otázku. Mala by teda byť buď binárneho charakteru, alebo by sa mala pohybovať medzi dvoma rozličnými hodnotami predstavujúcimi odpovede „áno" a „nie".

## 5.2 Zber dát

V rámci zberu dát je potrebné zozbierať čo najväčšie množstvo paketov zachytených z prenosu využívajúceho protokol Modbus/TCP, ako aj dáta, využívajúce čo najviac ďalších protokolov, aby výsledný model čo najlepšie dokázal odlíšiť prenos protokolu Modbus/TCP od prenosu ľubovoľného iného protokolu. Tieto dáta by mali byť naviac vyvážené, čiže počet Modbus/TCP paketov pri učení by nemal byť príliš odlišný od počtu paketov ostatných protokolov, aby sa vo výsledku model neprikláňal viac k jednej alebo k druhej výstupnej hodnote. Prenos protokolu Modbus/TCP však nie je jednotvárny. To znamená, že parametre prenosu závisia na tom, aké zariadenia ním komunikujú, aké dáta zasielajú a ako často. Preto boli vyhľadané a zozbierané dáta z viacerých zdrojov.

Zachytené dáta s paketmi protokolu Modbus/TCP pochádzajú z GitHub repozitárov tjcruz-dei/ICS_PCAPS [31], ITI/ICS-Security-Tools [32] (bro, openics) a antoine-lemay/Modbus_dataset [33]. Bolo vybraných 16 rôznych PCAP (Packet Capture) súborov s bežným Modbus/TCP prenosom zo 4 na sebe nezávislých zdrojov. Tieto dáta je možné rozšíriť o pakety zachytené vrámci experimentálneho pracoviska, budú však neskôr využité na testovanie modelu a preto nie je vhodné model na

týchto dátach učiť či validovať. Pakety neobsahujúce protokol Modbus/TCP pochádzajú z domény Netresec [34]. Hlavne ide o dáta od Chappell University a o trénovací dataset z dielne „Hands-on Network Forensics - FIRST 2015" s nabitým prenosom rôznych aplikačných protokolov. Pri výbere dát z oboch skupín bolo prihliadané aj na to, aby počet paketov v jednom datasete nebol výrazne vyšší ako v tom druhom. Mohlo by to mať za výsledok to, že sa model skôr bude prikláňať k hodnote, ktorá sa vyskytuje častejšie.

## 5.3 Predspracovanie dát

Zozbierané dáta je potrebné analyzovať a predspracovať, teda uviesť do podoby vhodnej na vstup do modelu. V rámci analýzy dát bude najprv určené, ktoré atribúty budú vrámci zozbieraných dát dostupné, a z nich vybrané tie, ktoré potenciálne nesú informáciu použiteľnú na riešenie problému. Ďalej budú hodnoty extrahované a prípadne bude vykonaná vhodná agregácia, normalizácia či ďalšie úkony, napr. prevedenie dát do jednoduchšej podoby, a to bez straty informácie, ktorú nesú.

### 5.3.1 Analýza dát a výber atribútov

Na analýzu paketov bude použitý open-source nástroj Wireshark. Dáta využiteľné na učenie sa môžu nachádzať na linkovej, sieťovej či transportnej vrstve a taktiež to môžu byť hodnoty, ktoré nie sú priamo súčasťou paketu, ale sú merateľné alebo zaznamenané a uložené v PCAP súbore. Učenie bude zamerané na protokoly Ethernet a IPv4, keďže dáta s inými protokolmi nie sú k dispozícii. Po analýze paketu bolo zistené, že sú dostupné významné dáta uvedené v tabuľke 5.1, kde sú taktiež zanalyzované súvislosti s vyšším protokolom či rozoznávaním staníc. Z dát boli vybrané tieto atribúty: frame.time_delta, frame.len, ip.hdr_len, ip.len, ip.proto, tcp.srcport, tcp.dstport, tcp.len, tcp.hdr_len a tcp.flags.[ack, fin, push, reset, syn, urg].

### 5.3.2 Extrahovanie hodnôt

Pre zefektívnenie extrahovania hodnôt budú najprv zlúčené všetky dáta dokopy, zvlášť pre dáta protokolu Modbus/TCP a zvlášť pre dáta ostatných protokolov. Najprv budú ale prevedené na spoločný formát *.pcapng* pomocou nástroja tshark [35] príkazmi vo výpise 5.1. Zlúčenie sformátovaných dát bude vykonané pomocou nástroja mergecap [36] príkazom vo výpise 5.2. Pred extrahovaním hodnôt budú pomocou Wiresharku filtrom „tcp.port == 502" odfiltrované z Modbus/TCP dát prípadné pakety iných protokolov a zobrazené dáta budú uložené do nového *.pcapng* súboru. Hodnoty budú extrahované do *.csv* súborov pomocou príkazu vo výpise 5.3.

Tab. 5.1: Dostupné hodnoty v pakete.

| Názov | Popis | Využitie |
|---|---|---|
| **frame** | **dáta zmerané či zaznamenané** | |
| *.time* | *čas, ktorý ubehol od zachytenia prvého paketu* | *vždy inkrementovaná, nemá význam pre učenie* |
| .time_delta | rozdiel času daného paketu a paketu predchádzajúceho | obsahuje informáciu o tom, ako na seba jednotlivé pakety časovo nadväzujú |
| .len | dĺžka celého rámca | súvisí s veľkosťou enkapsulovaných dát |
| **eth** | **dáta protokolu Ethernet (linková vrstva)** | |
| *.src*<br>*.dst* | *MAC adresa zdroja a cieľa* | *nenesú žiadnu využiteľnú informáciu* |
| *.type* | *protokol vyššej (spravidla sieťovej) vrstvy* | *informáciu už obsahujú prítomné ip dáta* |
| **ip** | **dáta protokolu IPv4 (sieťová vrstva)** | |
| *.version* | *verzia IP protokolu* | *vždy rovnaká* |
| .hdr_len<br>.len | veľkosť IP hlavičky a celková veľkosť IP datagramu | na výpočet veľkosti dátovej časti datagramu |
| *.id*<br>*.flags.x*<br>*.frag_offset* | *hodnoty súvisiace s fragmentáciou* | *fragmentácia závisí od sieť. nastavení, bez relevancie* |
| *.ttl* | *time-to-live paketu* | *opäť závisí od siete* |
| .proto | protokol vyššej (spravidla transportnej) vrstvy | využitý transportný protokol súvisí s vyššími protokolmi |
| .src<br>.dst | IP adresa zdroja a cieľa | rozlíšenie komunikujúcich strán |
| **tcp** | **dáta protokolu TCP (transportná vrstva)** | |
| .srcport<br>.dstport | TCP port zdroja a cieľa | často obsahuje port vyhradený pre protokol vyššej vrstvy, rozlíšenie komunikujúcich strán |
| .len | veľkosť dátovej časti segmentu | súvisí s veľkosťou enkapsulovaných dát |
| .hdr_len | veľkosť hlavičky segmentu | úzko súvisí s použitým protokolom vyššej vrstvy |
| .flags.x | TCP príznaky (ack, fin, reset, syn) | súvisia s riadením spojenia, rozlíšenie komunikujúcich strán |

Výsledný súbor bude obsahovať hodnoty oddelené čiarkou, kde jeden riadok predstavuje dáta z jedného paketu. Do prvého riadku súboru bude vypísaná hlavička s názvami atribútov. Premenná s názvom *occurrence* bude nastavená na *f* (z ang. slova first), aby sa v prípade násobných hodnôt vypísala len prvá. V prípade, že by sa táto premenná nenastavila, násobné hodnoty by boli vypísané za seba a oddelené čiarkou a súbor by bol tým nepoužiteľný, nakoľko počet hodnôt v riadku by v takom prípade nekorešpondoval počtu atribútov.

Výpis 5.1: Príkazy na zmenu formátu na *.pcapng*.

```
tshark.exe -F pcapng -r <CESTA_K_DÁTAM>\*.pcap -w                    1
   ↪ <CESTA_K_VÝSTUPNÝM_SÚBOROM>
tshark.exe -F pcapng -r <CESTA_K_DÁTAM>\*.cap -w                     2
   ↪ <CESTA_K_VÝSTUPNÝM_SÚBOROM>
```

Výpis 5.2: Príkaz na zlúčenie dát .

```
mergecap.exe -aF pcapng                                              1
   ↪ <CESTA_K_VSTUPNÝM_SÚBOROM>\*.pcapng -w
   ↪ <CESTA_K_VÝSTUPNÉMU_SÚBORU>
```

Výpis 5.3: Príkaz na extrahovanie dát do súboru *.csv* .

```
tshark.exe -r <CESTA_K_PCAPNG_SÚBORU> -T fields -e                   1
   ↪ frame.time_delta -e frame.len -e ip.proto -e ip.len
   ↪ -e ip.hdr_len -e tcp.srcport -e tcp.dstport -e
   ↪ tcp.hdr_len -e tcp.len -e tcp.flags.ack -e
   ↪ tcp.flags.reset -e tcp.flags.syn -e tcp.flags.fin
   ↪ -E header=y -E separator=, -E occurrence=f >
   ↪ <CESTA_K_VÝSTUPNÉMU_CSV_SÚBORU>
```

### 5.3.3 Spracovanie hodnôt

Extrahované hodnoty budú ďalej vhodné upravené na vstup do modelu. Na prácu s dátami budú použité Python knižnice pandas [37], numpy [38] a sklearn [39]. Knižnica pandas definuje objekt typu dataframe, do ktorého je možné načítať celý dataset z *.csv* súboru a vykonávať nad ním rôzne operácie. Knižnica numpy bude použitá len na operácie pre zmenu tvaru datasetu a knižnica sklearn len na oddelenie testovacích dát od dát na trénovanie. Ako prvá bude odčítaná veľkosť hlavičky IP paketu od celkovej veľkosti paketu. Výsledok bude zapísaný do nového atribútu s názvom ip.data_len a atribúty použité na výpočet budú odstránené. Ďalej budú vynulované negatívne a príliš veľké hodnoty relatívneho času. Abnormalita týchto hodnôt je spôsobená spájaním súborov so zachyteným prenosom. Taktiež bude zavedený nový

atribút tcp.direct, ktorý bude vypovedať o smere paketu. Na určenie smeru budú využité TCP porty a flagy. Pridaním daného atribútu sa táto informácia dostane aj k paketom, ktoré nemajú potrebné SYN, FIN a ACK flagy na určenie smeru paketu. Pakety s hodnotou:

- -1 – pochádzajú od stanice, ktorá inicializovala spojenie,
- 1 – pochádzajú od stanice, s ktorou bolo spojenie inicializované,
- 0 – nie je možné určiť smer paketu.

Keďže sú TCP porty zväčša len náhodnými hodnotami, bude vhodné tieto hodnoty odstrániť, keďže nie sú prospešné pre učenie a rozpoznávanie. Avšak stále môžu obsahovať port 502, ktorý je využívaný protokolom Modbus/TCP na strane servera. Preto bude vytvorený nový atribút s názvom tcp.modbus_port, ktorý bude nadobúdať hodnoty:

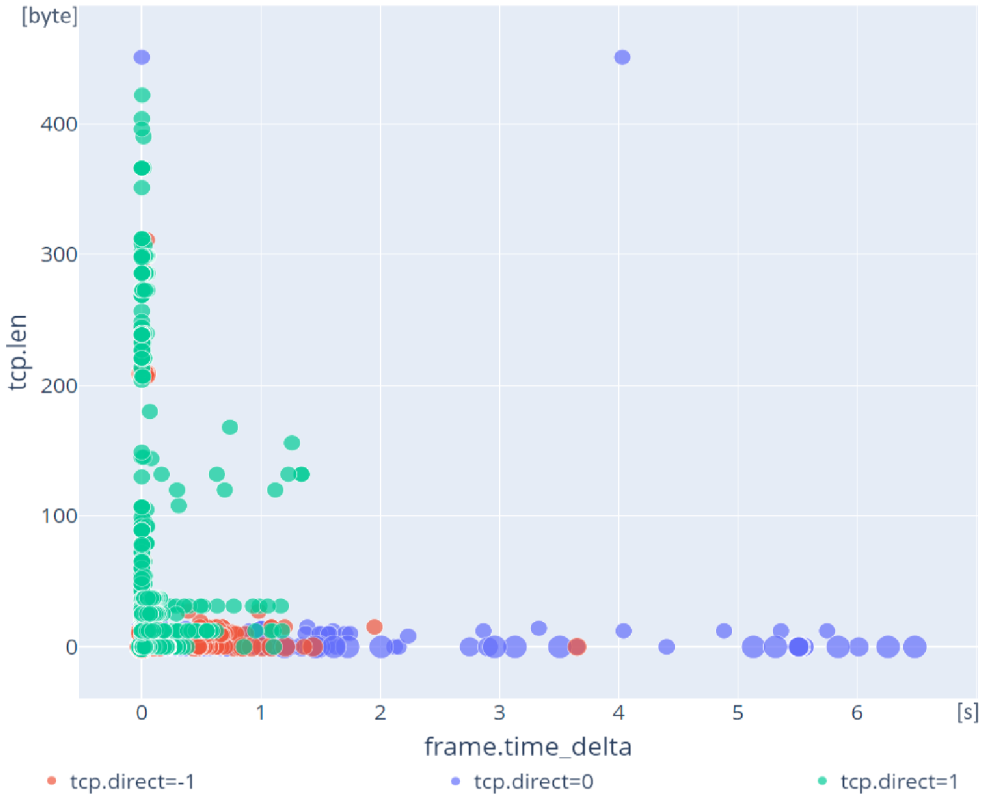
- -1 – ak je 502 port zdroja,
- 1 – ak je 502 port cieľa,
- 0 – ak port 502 nie je použitý.

Je však nutné brať do úvahy, že ak bude atribút tcp.modbus_port danému modelu poskytnutý, mohol by sa na neho pri učení až príliš zamerať. V prípade použitia iného portu než 502 by protokol Modbus/TCP nemusel byť rozpoznaný. Mimo iné, takmer pri všetkých atribútoch je potrebné brať do úvahy, že nemusia byť dostupné v prípade použitia iných protokolov. Absenciu týchto hodnôt bude teda vždy predstavovať hodnota 0. Z tohto dôvodu budú binárne hodnoty TCP príznakov upravené, a to tak, že hodnota 1 zostane rovnaká a hodnota 0 bude po novom -1. Všetky spomenuté operácie je samozrejme potrebné vykonať nad oboma datasetmi.

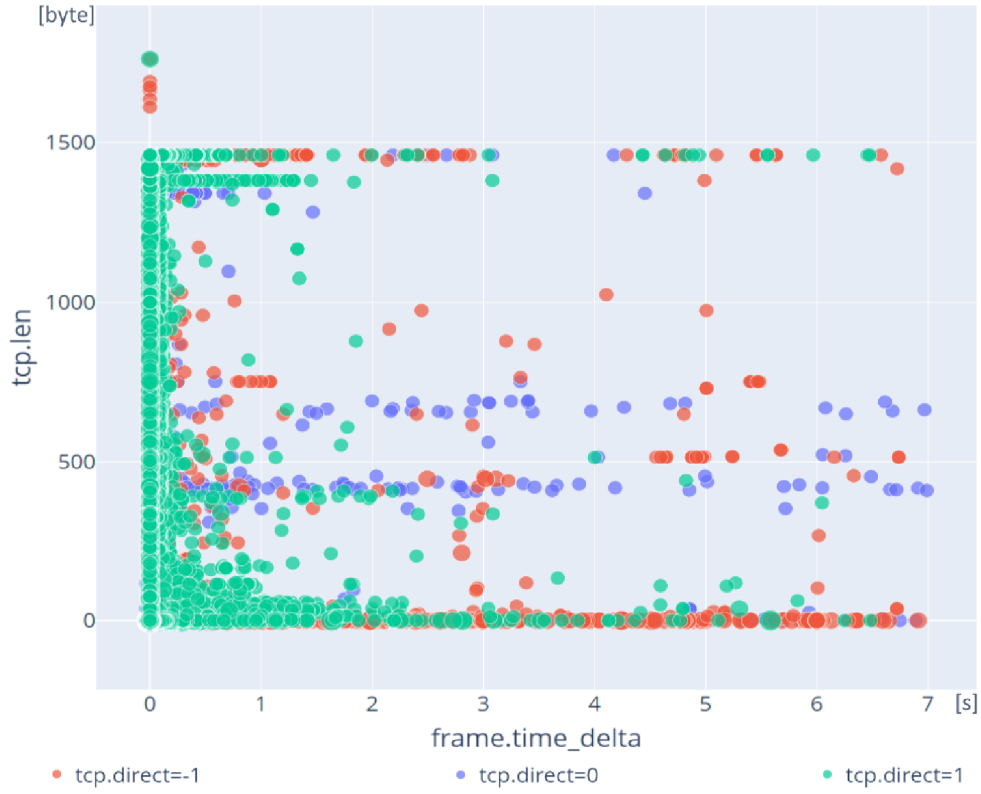
Na obrázkoch 5.1 a 5.2 je možné vidieť rozptýlenia hodnôt oboch datasetov, pričom sú brané do úvahy 4 dôležité atribúty (frame.time_delta, tcp.len, tcp.direct a tcp.hdr_len – reprezentovaný veľkosťou značky). Na obrázkoch je možné sledovať ako sa medzi datasetmi líšia vzťahy jednotlivých hodnôt. Na prvý pohľad je možné vidieť, že dáta Modbus/TCP datasetu tvoria jednoduchší vzor, čo môže naznačovať jednoduchšie rozpoznávania tohto protokolu medzi ostatnými. U Modbus/TCP datasetu je hlavne možné pozorovať pomerne malú diverzitu v hodnotách paketov smerujúcich od zariadenia, ktoré inicializovalo spojenie. Grafy boli vykreslené za pomoci knižnice plotly [40].

### 5.3.4 Finálne spracovanie dát

Kategorické atribúty v datasetoch obsahujú zväčša kategórie -1, 0 a 1. Na to, aby boli použiteľné na vstupe do modelu, bude použitý tzv. One-Hot Encoding. Každý kategorický atribút (okrem ip.proto) bude nahradený toľkými atribútami, koľko rôznych




Obr. 5.1: Rozptýlenie hodnôt Modbus/TCP datasetu.




Obr. 5.2: Rozptýlenie hodnôt datasetu bez Modbus/TCP.

hodnôt môže nadobúdať. Napr. tcp.direct bude nahradený hodnotami tcp.direct_-1, tcp.direct_0 a tcp.direct_1 a vždy, keď bude nadobúdať hodnotu 0, bude mať tcp.direct_0 hodnotu 1 a ostatné hodnoty budú nulové, čo analogicky platí aj pre ostatné kategórie. Keďže je možných hodnôt v ip.proto až príliš mnoho, bude nutné tento atribút z učenia vynechať. S dôvodu veľkosti datasetov a obmedzenia operačnej pamäte pri učení bude potrebné vynechať aj vstupné atribúty TCP príznakov, z ktorých už beztak bola extrahovaná najdôležitejšia informácia (tcp.direct). V datasete sa po úpravách tým pádom budú nachádzať atribúty frame.time_delta, frame.len, ip.data_len, tcp.hdr_len, tcp.len, tcp.direct_[-1, 0, 1], tcp.modbus_port_[-1, 0, 1].

Aby mohol model rozhodovať aj na základe vzťahov medzi paketmi, nie je vhodné ho nechať rozhodovať na základe jedného paketu, ale na určite časti prenosu. Na vstup do modelu preto budú dávané hodnoty 10 paketov. To je dostačujúce množstvo na to, aby medzi nimi bolo možné odpozorovať určité rozdielnosť či naopak súvislosti. Na druhú stranu je to dosť malá hodnota aby model nebol príliš komplexný a aby toľko paketov vôbec bolo zachytených pri kratších prenosoch. Zvlášť z dát protokolu Modbus/TCP a dát iného TCP prenosu je najprv z konca odstránených toľko paketov, aby bol ich počet deliteľný 10. Ďalej sú hodnoty paketov pospájané do skupín 10 paketov, čo má za výsledok 10-krát menej riadkov a zároveň 10-krát viac stĺpcov (vstupných hodnôt). Konečné datasety takto obsahujú 225 476 pozitívnych a 310 108 negatívnych vzoriek (každá po 10 paketov). K dátam je ďalej ako správny výsledok predikcie pridaná hodnota 1 v prípade, že ide o pakety protokolu Modbus, a 0, ak nie. Tým sú dáta pripravené na učenie. Je však nutné poznamenať, že atribúty frame.len, ip.data_len a tcp.len nesú jednu rovnakú podstatnú informáciu a pred učením bude vždy ponechaný len jeden z nich v závislosti od toho, z pohľadu ktorej vrstvy bude model predikovať.

## 5.4 Trénovanie modelov

Pomocou algoritmov vybraných v kapitole 2 boli ďalej modely trénované. Trénovanie prebiehalo najmä na GPU s rýchlosťou 875 MHz a s 2496 jadrami (Google Colaboratory GPU [41]). Ak nebolo možné využiť GPU, bolo použité jedno 4600 MHz CPU jadro. To bolo využité aj na vyhodnotenie doby predikcie všetkých modelov na dosiahnutie rovnakých podmienok počas testu. Dataset bol pred trénovaním každého modelu rozdelený na trénovacie a testovacie dáta v pomere 4 ku 1. Keďže rozhodovanie výsledných modelov bude prebiehať z pohľadu vrstvy s čo najviac informáciami, boli odstránené nadbytočné atribúty frame.len a ip.data_len. Vzorky datasetu boli náhodne poprehadzované.

### 5.4.1 Model neurónovej siete

Na tvorbu modelu ANN (Artificial Neural Network) bola využitá známa knižnica tensorflow [42], konkrétne jej súčasť keras. Ako prvá bola zvolená aktivačná funkcia, ktorá je využívaná na skrytých vrstvách. Keďže dataset obsahuje mnoho významných atribútov, ktoré nie je možné jednoznačne zhora ohraničiť a ktoré neobsahujú záporné hodnoty, bolo jednoznačne vhodné využiť aktivačnú funkciu ReLU.

#### Konfigurácia modelu

Ďalej bol na už jedinom datasete natrénovaný viac-menej náhodný model za účelom zistiť, či majú dáta vhodnú podobu. Prvá vrstva sekvenčného modelu má počet neurónov rovný počtu vstupných hodnôt, a teda 110. Bola pridaná jedna skrytá vrstva s aktivačnou funkciou ReLU s neurónmi väčšieho počtu ako je počet vstupných hodnôt. A ako posledná bola pridaná výstupná vrstva so sigmoidnou aktivačnou funkciou, ktorá je vhodná pri rozhodovaní v hodnotách 0 až 1. Bola zvolaná najnovšia aktivačná funkcia „Adam", ktorá je moderným rozšírením optimalizačnej funkcie gradientného zostupu. Ako stratová funkcia bola použitá binárna funkcia „BinaryCrossentropy". Vo výpise 5.4 je možné vidieť kód modelu. Vstupná vrstva je definovaná pomocou parametru „input_dim" vrámci prvej skrytej vrstvy.

Výpis 5.4: Konfigurácia prvého modelu neurónovej siete.

```
model = keras.Sequential()
model.add(Dense(512, input_dim=df_train.shape[1],
    ↪ activation = "relu"))
model.add(Dense(1, activation = "sigmoid"))

model.compile(optimizer = keras.optimizers.Adam(),
    ↪ loss = keras.losses.BinaryCrossentropy(),
    ↪ metrics = ["accuracy"])
```

Hodnoty označujúce správnu predikciu vyňaté do samotného dataframu. Pre prvé spustenie bol zvolený nízky počet epoch a iteratívne bola zvolená optimálna veľkosť dávky (počet vstupov, po ktorých prebieha spätná propagácia), aby trénovanie nebolo príliš pomalé (a zároveň aby pri učení s viacerými epochami nedochádzalo k degradácii modelu). Trénovanie bolo spustené príkazom vo výpise 5.5 kde bola nastavený podiel validačných dát 0,2. Už po prvej epoche bolo možné vidieť (viď výpis 5.6), že strata okamžite klesla do zanedbateľných hodnôt.

Výpis 5.5: Príkaz na spustenie trénovania neurónovej siete.

```
model.fit(df_train, target_train, epochs = 3,
    ↪ batch_size = 256, validation_split=0.2)
```

Výpis 5.6: Priebeh trénovania prvej neurónovej siete.

```
Epoch 1/3                                                    1
- loss: 0.4964 - accuracy: 0.9902                            2
- val_loss: 8.4325e-04 - val_accuracy: 0.9997                3
```

#### Analýza a ladenie modelu

Po sčítaní váh medzi vstupnou a skrytou vrstvou pre každý z 11 vstupov bolo možné vidieť, že súčet bol pre tcp.modbus_port_0 približne 330, zatiaľ čo súčty zvyšných váh boli všetky záporné. To znamená, že model sa rozhoduje prevažne na základe toho, či je port 502 použitý, alebo nie. Po odobratí atribútu tcp.modbus_port z procesu učenia bol po 10 epochách súčet váh vždy záporný a presnosť sa iteratívne zlepšovala každou etapou. Týmto bolo overené, že algoritmus pracuje tak ako má. Jednak, že sa podľa očakávaní zameral na „istý" atribút, ale že aj bez neho je schopný iteratívne zlepšovať presnosť modelu. Ďalej boli iteratívne zvolené parametre neurónovej siete s cieľom dosiahnuť optimálnu presnosť predikcie, a minimalizovať komplexitu a dobu trénovania. Nebola potrebná žiadna ďalšia skrytá vrstva, počet neurónov v tej existujúcej bol zredukovaný na 128, trénovanie bolo ukončené po 20 epochách s veľkosťou dávky 2048. Trvalo 18 sekúnd a evaluácia na testovacích dátach, ktoré neboli využité pri trénovaní, dosahovala presnosti 99,83 %. S daným modelom však bolo možné dosiahnuť až presnosti 99,92 %, na čo postačilo ďalších 123 sekúnd trénovania s iteratívnym zväčšovaním dávok a počtu epoch. Doba jednej predikcie modelu je 9,66e-2 sekundy.

### 5.4.2 C4.5 model

Na tvorbu C4.5 rozhodovacieho stromu bol použitý jeden z mála dostupných balíkov chefboost [43]. Na trénovanie boli použité rovnaké dáta ako u ANN modelu s jediný rozdielom, že hodnoty označujúce správnu predikciu boli ponechané v dataframe s dátami pre trénovanie a názov tohto atribútu bol predaný učiacemu algoritmu. Trénovanie bolo spustené príkazmi vo výpise 5.7. Implementácia nepodporuje využitie GPU, preto bolo použité iba jedno jadro CPU. Možnosť paralelného trénovania na viacerých jadrách bola nefunkčná.

Výpis 5.7: Príkaz na spustenie trénovania C4.5 modelu.

```
config = {'algorithm': 'C4.5'}                               1
model = chef.fit(df_train, config=config,                    2
    ↪ target_label='Decision')
```

Trénovanie trvalo 2167 sekúnd, presnosť modelu je 98,42 % a doba jednej predikcie je 1,56e-5 sekundy, čo je výrazne nižšia hodnota oproti ostatným modelom.

### 5.4.3 Support Vector Machine model

Na trénovanie modelu SVM (Support Vector Machine) bol použitý Python balík thundersvm [44]. Tento balík bol vybratý nakoľko podporuje trénovanie SVM modelov na GPU. Algoritmu boli poskytnuté rovnaké dáta ako u neurónovej siete a trénovanie bolo spustené príkazmi vo výpise 5.8.

Výpis 5.8: Príkaz na spustenie trénovania SVM modelu.

```
model = SVC()
model.fit(df_train, target_train)
```

Trénovanie trvalo 356 sekúnd a presnosť modelu je 99,74 %. Doba jednej predikcie je 7,38 sekundy, čo aj napriek vysokej presnosti robí z tejto metódy nevhodného kandidáta na klasifikáciu prenosu v reálnom čase.

## 5.5 Komplementárne testovanie modelov

Natrénované modely boli ďalej testované na rozličných datasetoch. Prvý z datasetov opäť pochádza z domény Netresec, konkrétne ide o dataset z Megalodon Challenge [45]. Pakety datasetu boli zachytené na webovom a dátovom servery. Druhý dataset pozostáva zo vzorových zachytených paketov z Wireshark domény [46], ktoré obsahujú množstvo rôznych protokolov. Dáta boli stiahnuté pomocou príkazu vo výpise 5.9 a spracované rovnako ako ostatné datasety. Avšak, behom konverzie boli niektoré problémové *.cap* súbory z testovacích dát vynechané. Tretím datasetom sú pakety zachytené behom Modbus/TCP simulácie v kapitole 4. Prvý dataset po spracovaní obsahuje 535 554 negatívnych vzoriek, druhý 61 821 negatívnych vzoriek a tretí 131 228 pozitívnych vzoriek (každá po 10 paketov).

Výpis 5.9: Príkaz na stiahnutie všetkých vzorových zachytených dát.

```
wget -e robots=off -nc -r -l 1 --accept-regex='.*do=get.*(
    ↪ p?cap|pcapng)(\.gz)?$' --ignore-case wiki.wireshark.
    ↪ org/SampleCaptures?action=AttachFile
```

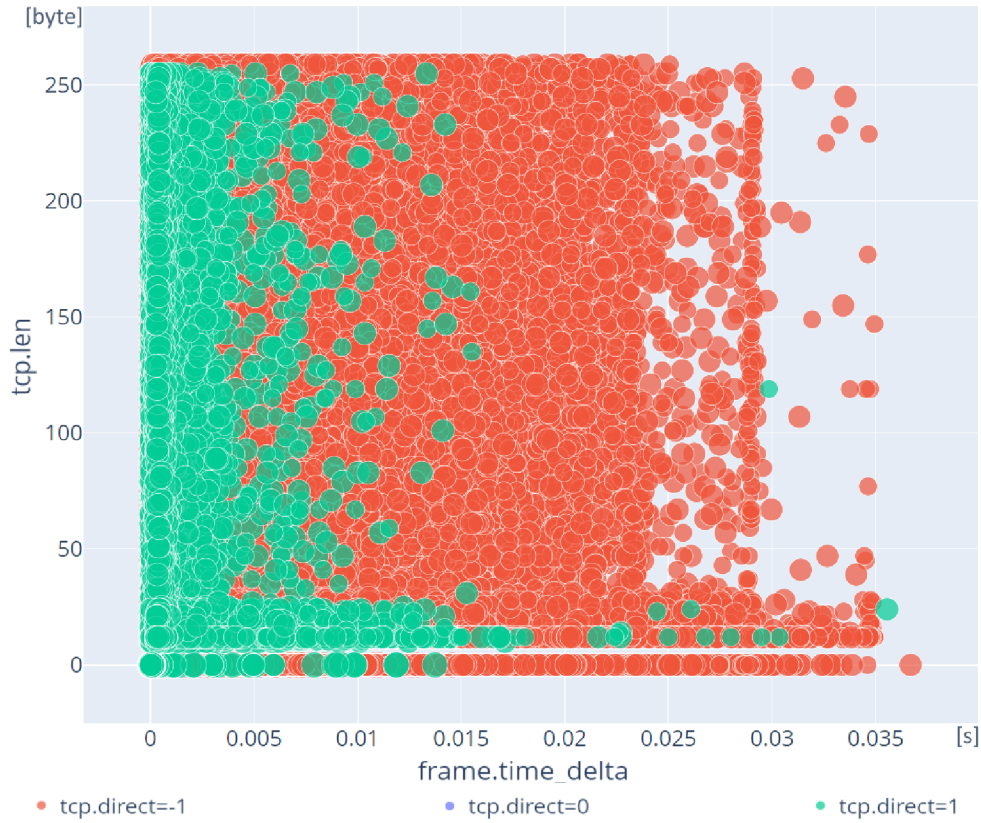
Testovaný nebol model SVM, nakoľko by jeho testovanie trvalo v rádoch mesiacov. Zvyšné dva modely podstúpili rovnakú evaluáciu ako pri ich testovaní na testovacích dátach. Pre Wireshark dataset však boli natrénované nové ANN a C4.5 modely, ktoré využívajú len dáta bez hodnôt sieťovej a transportnej vrstvy, a to konkrétne atribúty frame.time_delta a fram_len. Tento dataset totiž obsahuje množstvo paketov bez sieťového protokolu a preto bolo vhodné na ňom odskúšať, ako si techniky poradia s menším počtom vstupných atribútov. Pre vyššiu presnosť modelu ANN bol zvýšený počet neurónov v skrytej vrstve na 512 a boli pridané ďalšie dve zhodné skryté vrstvy.

## 5.6 Porovnanie modelov

V tabuľke 5.2 je možné vidieť všetky zmerané a vyhodnotené štatistiky modelov vrátane výsledkov ich testovania. Štatistiky nových modelov pre rozhodovanie z pohľadu spojovej vrstvy sú označené „L2". Najlepšie výsledky v presnosti klasifikácie má model ANN, model C4.5 je na druhú stranu výrazne rýchlejší pri predikovaní. Práve v tomto kritériu neobstál model SVM. Model ANN významne predčil model C4.5 v presnosti predikcie na datasete Wireshark vzorkách. Zatiaľ čo neurónovú sieť bolo pre vyššiu presnosť potrebné rozšíriť, algoritmus C4.5 naopak trénovanie skončil pri značne nižšom počte uzlov ako pri jeho predchodcovi s viacerými vstupmi. Ani jeden z modelov však neobstál pri rozpoznávaní protokolu Modbus/TCP na simulovaných dátach, čo značne zdeformovalo celkové hodnotenie modelov. Dôvod neúspechu napovedá aj obrázok 5.3 s rozptýlením hodnôt tohto datasetu v porovnaní s rozptýlením hodnôt Modbus/TCP datasetu využitého na trénovanie na obrázku 5.1, z čoho je pochopiteľné, prečo model ANN označil takmer dve tretiny Modbus/TCP vzoriek falošne negatívne.

Tab. 5.2: Porovnanie techník strojového učenia.

| **Model** | ANN | C4.5 | SVM |
|---|---|---|---|
| **Čas trénovania** | 141 s | 2167 s | 356 s |
| **Čas predikcie** | 9,66e-2 s | 1,56e-5 s | 7,38 s |
| **Presnosť (Testovacie dáta)** | 99,92 % | 98,42 % | 99,74 % |
| **Presnosť (Dataset - webový server)** | 99,86 % | 96,68 % | - |
| **Presnosť (Dataset - Modbus simulácia)** | 35,93 % | 58,42 % | - |
| **L2: Presnosť (Testovacie dáta)** | 99,01 % | 94,70 % | - |
| **L2: Presnosť (Dataset - wireshark.org)** | 95,45 % | 65,78 % | - |
| **Správne pozitívne/pozitívne vzorky** | 76,41 % | 72,09 % | - |
| **Správne negatívne/negatívne vzorky** | 99,90 % | 89,83 % | - |
| **L2: Správne pozitívne/pozitívne vzorky** | 99,55 % | 80,05 % | - |
| **L2: Správne negatívne/negatívne vzorky** | 98,31 % | 71,76 % | - |
| **F1-score** | 0,8659 | 0,7999 | - |
| **L2: F1-score** | 0,9893 | 0,7568 | - |




Obr. 5.3: Rozptýlenie hodnôt datasetu Modbus/TCP simulácie.

## 5.7 Zhodnotenie modelov pre využitie v systéme

Na základe výsledkov testovania je možné vyvodiť, že je značne odlišnou úlohou naučiť model rozpoznávať konkrétny spôsob využitia protokolu od úlohy naučiť ho rozpoznávať protokol ako taký. Rovnako boli aj modely v rámci tejto práce naučené rozpoznávať určitý spôsob využitia protokolu Modbus/TCP. Model ANN mal dokonca tendenciu označiť simulovaný prenos vo väčšine prípadov negatívne, nakoľko bol učený na značne odlišných typoch prenosu.

Vzhľadom k presnosti predikcie a prispôsobiteľnosti bol vybraný a bude v rámci systému používaný model ANN. A to samozrejme v prípade, že sa doba predikcie tohto modelu behom práce nestane kritickou. Použitá implementácia ANN má naviac výhodu v možnostiach trénovania na GPU a dotrénovania existujúceho modelu. Ďalej bude vhodné upraviť počet paketov, na ktorých je predikcia vykonávaná, na 8 paketov. Je to totiž najnižší počet paketov, ktorý má jedno TCP spojenie pri zaslaní len jednej správy. Pakety týchto spojení by ináč s nutnosťou 10 paketov nikdy neboli predikované.

## 5.8 Tvorba modelov využitých v systéme

Pre použitie vo finálnom systéme bolo vybraným protokolom vytvorených 5 modelov. Priemyselnými protokolmi sú Modbus/TCP a DNP3 vybrané na základe dostupnosti datasetov vyobrazenej v článku [47]. Bežnými aplikačnými protokolmi sú HTTPS a FTP (zvlášť modely pre riadiace a dátové spojenie) vzhľadom k ich vysokej používanosti a ich spoločnej možnosti využitia na prenos súborov, čo môže zaviesť vyššiu obtiažnosť pri odlišovaní týchto dvoch protokolov. Modely boli trénované na dátach zozbieraných počas simulácií popísaných v podkapitole 4.1 a na prípadných dátach z verejných zdrojov. Na vstupoch do modelov boli použité dáta z 8 paketov s rovnakými atribútmi ako v podkapitole 5.4: frame.time_delta, frame.len, ip.data_len, tcp.hdr_len, tcp.len, tcp.direct_[-1, 0, 1]. Negatívnymi dátami boli dáta z domény Wireshark použité pri trénovaní prvého Modbus/TCP modelu. Z nich boli samozrejme odfiltrované dáta spomenutých protokolov. Pre každý model boli naviac ako negatívne dáta použité pozitívne dáta ostatných modelov.

Na trénovanie Modbus/TCP modelu boli použité dáta z podkapitoly 5.2 a dáta zachytené počas simulácií. DNP3 model bol trénovaný ako na dátach simulácií, tak aj na dátach z pomerne malého datasetu použitého v [48]. FTP modely boli z dôvodu absencie verejne dostupných datasetov natrénované čisto na simulovaných dátach. Model HTTPS bol trénovaný ako na dátach simulácie, tak na dátach webového serveru z Megalodon Challenge použitých aj v kapitole 5.5. Skryté vrstvy modelov obsahovali po 128 neurónov. Modely boli trénované z dôvodu objemnejšieho datasetu s veľkosťou dávky 2048. V rámci tabuľky 5.3 je možné vidieť nastavenia špecifické pre modely jednotlivých protokolov, ako aj presnosť evaluácie modelov na testovacích dátach. Ostatné nastavenia boli zhodné s nastaveniami vo výpisoch 5.5 a 5.6.

Tab. 5.3: Nastavenie trénovania finálnych modelov a presnosť evaluácie.

| **Model** | **Počet skrytých vrstiev** | **Počet epoch** | **Presnosť evaluácie (testovacích dáta)** |
|---|---|---|---|
| Modbus/TCP | 2 | 100 | 99,96 % |
| DNP3 | 1 | 100 | 99,94 % |
| HTTPS | 2 | 200 | 99,82 % |
| FTP (control) | 1 | 50 | 99,99 % |
| FTP (data) | 1 | 100 | 99,93 % |

# 6 Systém na rozpoznávanie zariadení a protokolov v sieti

Systém bude odchytávať prenos na sieti, identifikovať dvojice zariadení na základe ich IP adries, portov prípadne iných identifikátorov a rozpoznávať protokol, ktorým komunikujú. Ak, samozrejme, ide o protokol, ktorého modelom systém disponuje.

## 6.1 Prístupy k riešeniu

K riešeniu je možno voliť z viacero prístupov. Prvými prístupmi, ktorými je potrebné sa zaoberať, sú prístupy na základe počtu tried klasifikácie jedného modelu. Jeden model môže určovať:

- s ako pravdepodobnosťou ide o jeden daný protokol – systém tak bude jednoducho modulovateľný, model každého nového protokolu by bol pridaný zvlášť a nebolo by potrebné nijak zasahovať do už prítomných modelov,
- o ktorý s podporovaných protokolov najpravdepodobnejšie ide – zložitejšia modulácia, po pridaní protokolu by bolo potrebné pretrénovať model a pridať výstup pre daný protokol, réžia modelov vrámci systému by však bola jednoduchšia a rozhodovanie by bolo vo väčšine prípadov rýchlejšie, keďže by na rozhodovanie stačilo použiť vždy len jeden model.

Systém by mal byť vo výsledku čo najjednoduchšie modulovateľný, preto bude zvolený prvý prístup – jeden model na jeden protokol.

Ďalšími prístupmi sú prístupy k vstupným dátam, u ktorých existujú nasledujúce možnosti:

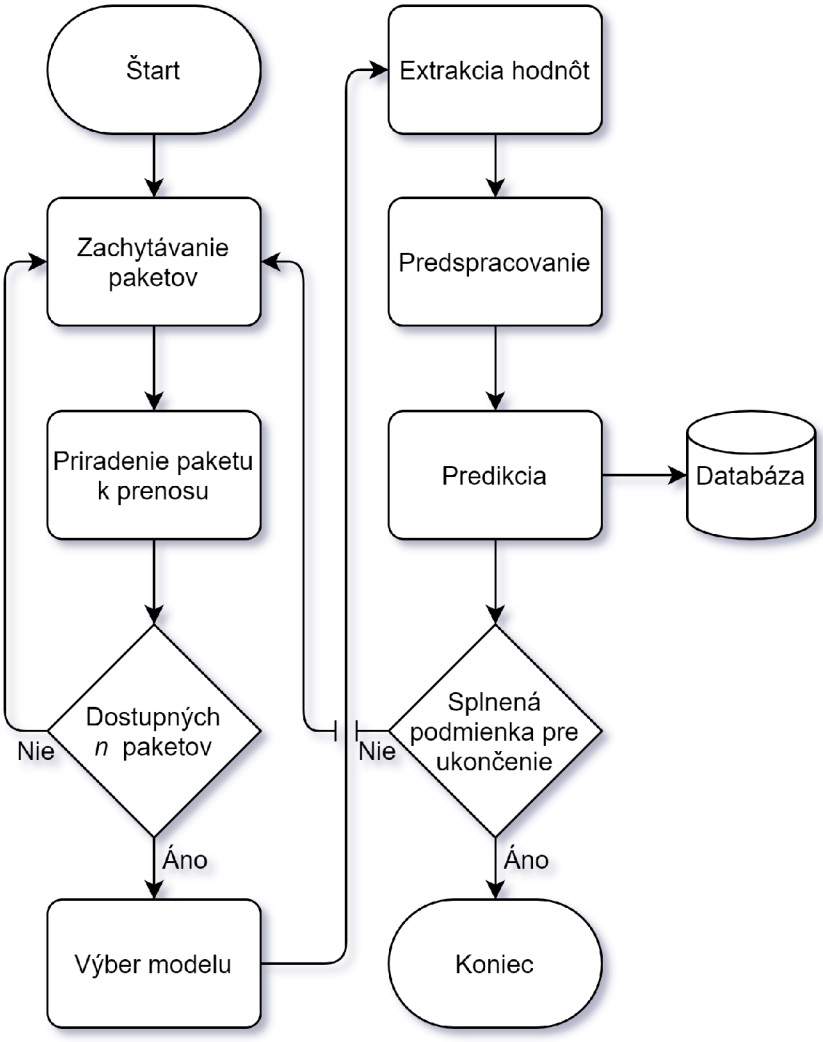
- všetky modely budú mať rovnaké vstupné dáta – rýchlejšie spracovanie dát, jednoduchšie nastavenia modelov v rámci systému,
- každý model bude mať svoju vlastnú množinu vstupných dát – rýchlejšie predikcie z dôvodu menšieho priemerného počtu vstupných dát do modelov, avšak pre užívateľa jednoduchšie modulovateľný systém v prípade, že bude chcieť pridať model s novým, do tej doby nepoužívaným atribútom.

Keďže je vhodné minimalizovať nároky na výkon a čo najviac urýchliť pomerne pomalé predikcie modelov, aby systém zvládal spracovávať čo najviac frekventované prenosy, bol zvolený osobitný prístup k modelom vzhľadom ku vstupným dátam. To naviac umožní efektívnejšiu modularitu systému.

## 6.2 Návrh implementácie systému

Úlohou systému je identifikovať komunikujúce zariadenia a rozpoznávať protokol zachyteného prenosu. Pre túto úlohu bude nutné prenos zachytiť. Vrámci implementácie teda bude nutné zvoliť knižnicu, za pomoci ktorej bude možné pakety zachytávať a ktorá bude poskytovať prístup ku všetkým dátam paketu.

Každý paket bude po zachytení priradený k prenosu dvojice staníc na základe ich identifikátorov. Ak je daný paket už ich $n$-tým priradeným paketom (kde $n$ je počet paketov na základe ktorých budú protokoly rozpoznávané), bude vybraný model na predikciu a z paketov budú extrahované potrebné dáta, prípadne odvodené ďalšie atribúty pre model. Všetky dáta týchto $n$ paketov budú následne predspracované pre daný model, modelom predikované a výsledok predikcie bude spolu s užitočnými parametrami prenosu uložené do databázy. Diagram činnosti takéhoto systému je možné vidieť na obrázku 6.1.




Obr. 6.1: Diagram návrhu hlavnej činnosti systému.

Systém by mal bežať na určitom zariadení, sonde, zapojenej do sieťového zariadenia podporujúceho port mirroring, cez ktoré prechádza celý prenos siete, alebo ktoré vie kópiu všetkého prenosu na sieti k sonde dopraviť iným spôsobom (napríklad vo forme streamu zachytených paketov).

## 6.3 Implementácia systému

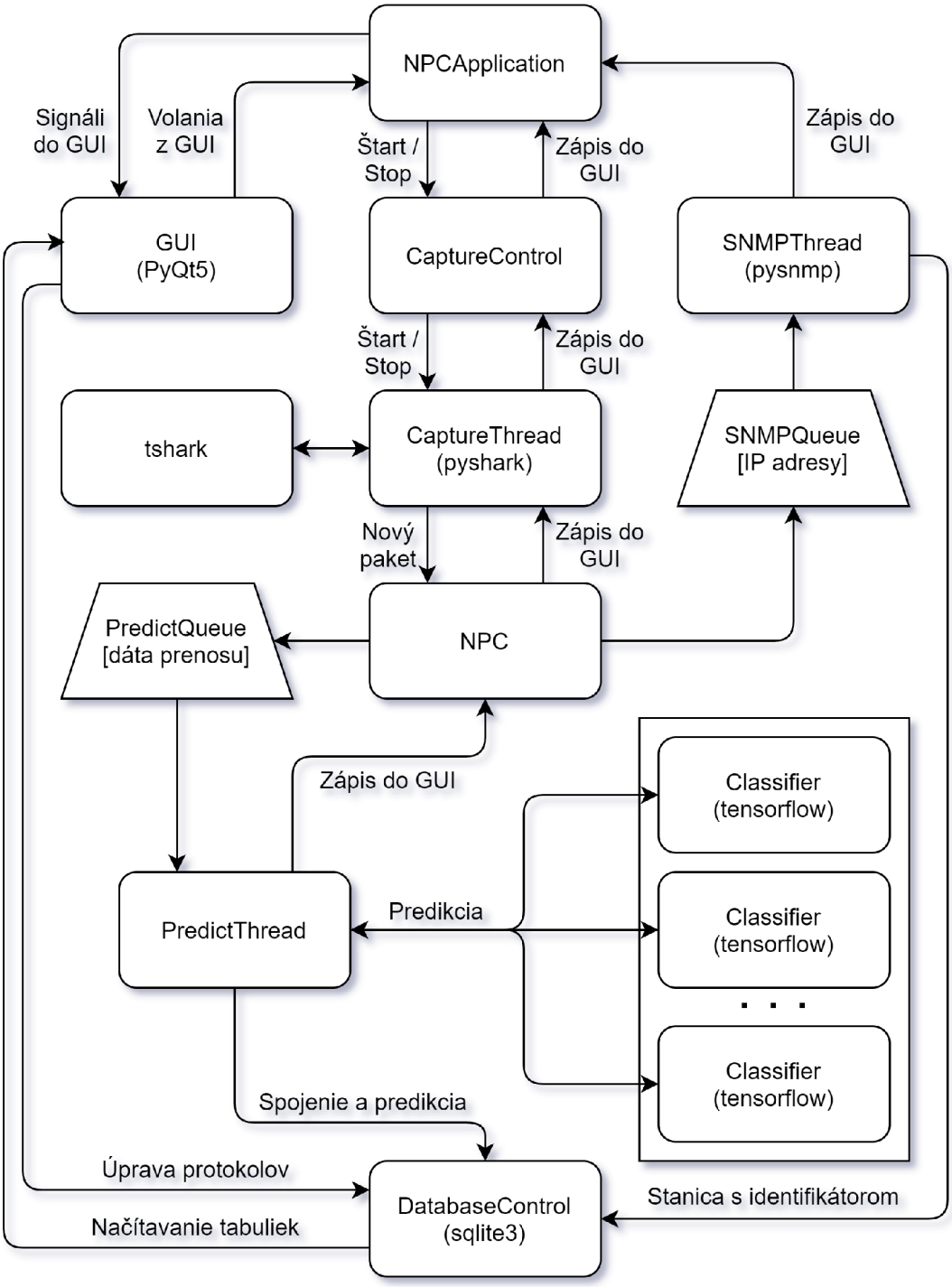
Implementácia systému pozostáva zo štyroch hlavných tried a beží na štyroch vláknach (vrátane hlavného vlákna), avšak nepočítajúc s vláknami, ktoré si použité balíčky prípadne vytvárajú podľa potreby samé. Objekty týchto triedy spolu zabezpečujú zachytávanie paketov, ich spracovanie, predikciu na základe poskytnutých modelov, komunikáciu s databázou pre čítanie a zápis potrebných dát a dotazovanie sa na identifikátory pomocou SNMP protokolu. Výmenu informácií medzi jednotlivými súčasťami systému je možné vidieť na diagrame na obrázku 6.2. Diagram obsahuje hlavný objekt NPCApplication, ktorý za pomoci signálov spravuje GUI (Graphical User Interface), spracováva volania z GUI a poskytuje možnosť výpisu textu do GUI ostatným objektom a vláknam. Samotné GUI na základe signálov od objektu NPCApplication načítava dáta z databázy, prípadne do nej zapisuje zmeny vykonané užívateľom. Diagram ďalej znázorňuje smer dát či príkazov predávaných medzi jednotlivými objektami systému, ako aj zápis a čítanie z/do front či zapisovanie do databázy.

### Trieda CaptureControl

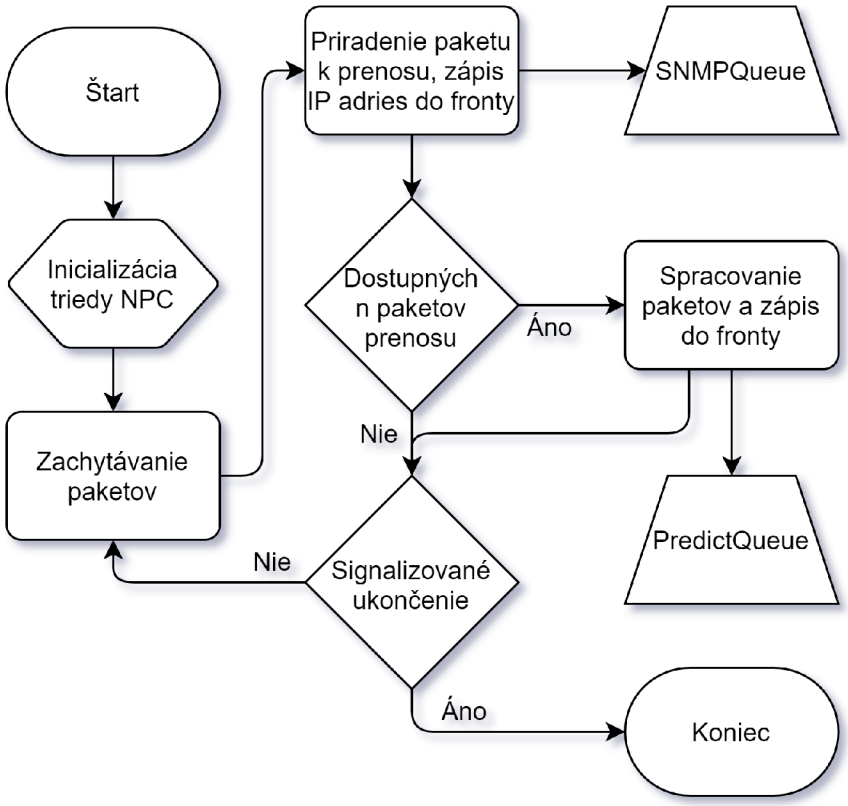
Trieda CaptureControl zastáva rolu prostredníka medzi hlavným objektom aplikácie a samostatných vláknom zachytávania paketov a ponúka funkcie pre riadenie tohoto vlákna. To zahŕňa spúšťanie a ukončenie vlákna zachytávania na danom rozhraní či informovanie užívateľa o stave zachytávania pomocou užívateľského rozhrania.

### Trieda CaptureThread

Vlákno zachytávania, ktoré sa inicializuje v rámci CaptureControl objektu, inicializuje objekt triedy NPC (Network Packet Classifier) a každý zachytený paket mu pomocou funkcie predáva na spracovanie. Na zachytávanie paketov bol využitý balíček pyshark [49], ktorý je Python wrapperom pre nástroj tshark. Nástroj umožňuje kontinuálne zachytávanie paketov na zadanom sieťovom rozhraní a umožňuje jednotlivo spracovávať každý zachytený paket. Pre diagram činnosti vlákna viď 6.3.




Obr. 6.2: Diagram výmeny informácii medzi jednotlivými súčasťami systému.



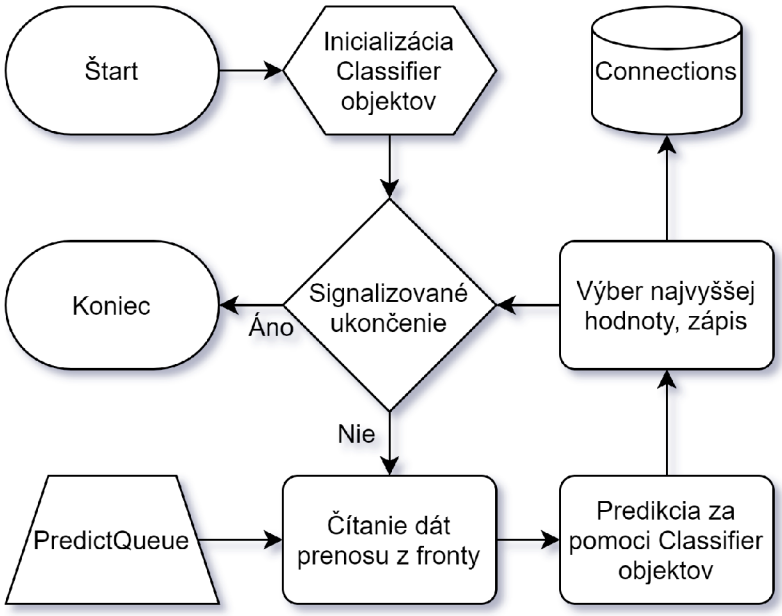

Obr. 6.3: Diagram činnosti vlákna CaptureThread.

### Trieda NPC

Ako už bolo spomenuté, objekt triedy NPC je inicializovaný vláknom CaptureThread a zodpovedá za spracovanie každého zachyteného paketu. Taktiež spravuje predikčné vlákno. NPC objekt pakety triedi podľa jednotlivých spojení identifikovaných IP adresami a portami. V prípade dosiahnutia potrebného počtu paketov pre predikciu z paketov extrahuje/odvodí potrebné atribúty a takto spracované dáta vloží do fronty pre predikčné vlákno. Mimo to má objekt NPC za úlohu predávať každú zachytenú IP adresu do fronty pre SNMP vlákno.

### Trieda PredictThread

Vlákno triedy PredictThread sa stará o predikciu, zápis do databázy či výpis informácií do užívateľského rozhrania. Každému modelu protokolu vytvorí vlastný objekt triedy Classifier. V rámci predikcie je za pomoci všetkých Classifier objektov predikovaný protokol daného prenosu. Protokol s najvyššou hodnotou predikcie je spolu s informáciami o spojení zapísaný do databázy. Pre diagram činnosti vlákna viď 6.4.




Obr. 6.4: Diagram činnosti vlákna PredictThread.

### Trieda Classifier

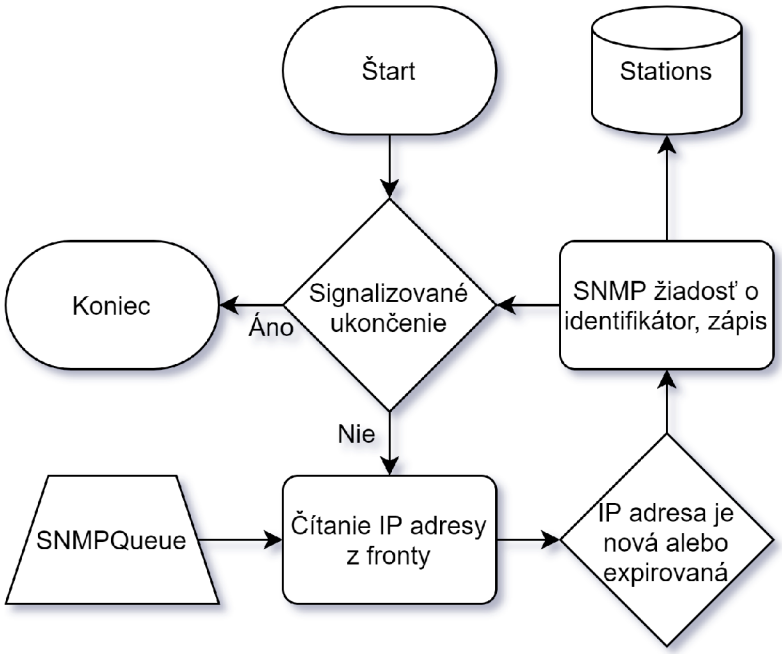
Objekty triedy Classifier sa starajú o prípadný One-Hot Encoding, dáta upravujú do vhodnej podoby na vstup do modelu a pomocou tensorflow modelu vykonávajú predikciu. Pri inicializovaní predikčného vlákna sú z databázy do týchto objektov načítané informácie o rozpoznávaných protokoloch potrebné pre spracovanie paketov či predikciu.

### Trieda SNMPThread

Pri spustení systému je inicializované a spustené vlákno triedy SNMPThread, ktoré má na starosti pomocou protokolu SNMP a balíčku pysnmp [50] dotazovať sa na identifikátory staníc so zachytenými IP adresami, ktoré, ako bolo spomenuté, dostáva pomocou fronty od NPC objektu. Ďalej žiada základné informácie o systéme stanice a následne IP adresu zapíše do databázy s prípadnými získanými informáciami. Pre diagram činnosti vlákna viď 6.5.

### Trieda DatabaseControl

Každé vlákno, ktoré nejakým spôsobom používa databázu, v určitej fáze behu inicializuje objekt triedy DatabaseControl. Tento objekt má pod správou jedno spojenie s databázou a poskytuje vláknam funkcie pre vytváranie definovaných tabuliek, ich




Obr. 6.5: Diagram činnosti vlákna SNMPThread.

úpravu či čítanie z nich. Pre jednoduchosť využitia zaobaľuje dlhé SQL dotazy v podobe textu a poskytuje spoľahlivé (v prípade chyby opakované) vykonanie týchto dotazov s následným odomknutím databázy pre ostatné vlákna.

Objekt DatabaseControl spravuje tri tabuľky:

- app_protocols – obsahuje informácie o protokoloch, ktoré má byť systém schopný rozpoznávať:
  - name – názov protokolu,
  - model_path – relatívna/úplna cestu k modelu,
  - attributes – list názvov atribútov, ktoré má objekt NPC extrahovať,
  - model_input_attributes – list názvov atribútov v poradí, akom vstupujú do modelu (po prípadnom One-Hot Encodingu),
  - one_hot_values – slovník s názvami a atribútov, pre ktoré má byť vykonaný One-Hot Encoding, a všetkými hodnotami, ktoré môžu nadobúdať,
- connections – obsahuje informácie o zachytených spojeniach vrátane predikcií:
  - prediction_time – čas predikcie,
  - ip_address_a – zdrojová IP adresa prvého zachyteného paketu spojenia,
  - ip_address_b – cieľová IP adresa prvého zachyteného paketu spojenia,
  - port_a – zdrojový port prvého zachyteného paketu spojenia,
  - port_b – cieľový port prvého zachyteného paketu spojenia,
  - transport_protocol – protokol transportnej vrstvy (TCP/UDP),
  - prediction – výsledok predikcie,

- stations – obsahuje informácie o jednotlivých staniciach získané pomocou protokolu SNMP:
  - identification_time – čas zaslania dotazu o identifikátor,
  - ip_address – IP adresa, na ktorú bol dotaz zaslaný,
  - identifier – identifikátor zariadenia (MAC adresa),
  - os_info – informácie o systéme stanice.

### SNMG agent

Pre funkčnosť systému je nutné mať na známych staniciach zapnutého tzv. agenta, ktorý odpovedá na SNMP dotazy. Pre testovacie účely bol na virtuálnych zariadeniach simulujúcich komunikáciu pomocou vybraných protokolov použitý jednoduchý agent, ktorý bol upravený tak, aby odpovedal na dotaz s OID 1.3.6.1.2.1.2.2.1.6 (IfPhysAddress) MAC adresou a na dotaz s OID 1.3.6.1.2.1.1.1 (SysDescr) informáciou o operačnom systéme. Komunikácia agenta so systémom však nie je nijak zvlášť zabezpečená či šifrovaná, čo ju robí náchylnú na MITM (Man In The Middle) útoky. Riešením by bolo napríklad využitie SNMPv3 s AES-256 a vopred zvoleným kľúčom. V neposlednom rade analýzu ochrany proti replay útokom v SNMPv3 je možné nájsť v článku [51].

### Doplňujúce informácie o systéme

Systém klasifikuje prenos daného spojenia vždy po uplynutí určitého nastaveného času od poslednej klasifikácie. Využíva na to časy zapisované v momente priradenia paketu k spojeniu. Výhodou tohto prístupu je jednak to, že systém nie je pri príliš frekventovaných prenosoch zaťažený neustálou klasifikáciou, a naviac môže pakety akumulovať a následne presnejšie klasifikovať prenos na viacerých paketoch spriemerovaním hodnôt jednotlivých predikcií. Systém taktiež pravidelne premazáva pakety starých spojení, pričom ich vyhodnocuje ak sú dostatočného počtu. Doplnkovou možnosťou, ktorou je možné predikovať zachytený prenos, je importovanie PCAP súboru dostupné v menu GUI. Systém po zvolení súboru spracuje postupne všetky jeho pakety a následne ich hromadne predikuje pre každé jedno spojenie. Však v prípade, že je obmedzená veľkosť hromadných predikcií, môže systém začať predikovať už pri načítavaní paketov. Čo znamená, že inicializuje predikciu už načítaných paketov, ak by pridaním ďalšieho paketu mal ich počet presiahnuť toto obmedzenie.

Nevýhodou systému a jeho práce s modelmi je, že aj jeden nepresný model môže výrazne degradovať presnosť celého systému. A to najmä v prípade, že má tendenciu falošne predikovať s vysokými hodnotami. A teda čím bližšie sú tieto predikcie hodnote 1, tým má horší dopad na systém. Takýto model potom dokáže „prebiť"

iný model, ktorý správne predikuje s hodnotou nad 0,5. Možným spôsobom riešenia tohoto problému by mohlo byť zavedenie systému priorizácie modelov, kedy by modelom užívateľ prideľoval napr. číselné priority na základe ich „dôveryhodnosti". Modely by sa vyhodnocovali v prioritami zavedenom poradí (samozrejme od najvyššej priority) a pri prvej pozitívnej predikcii by bolo predikovanie zastavené, čím by sa preskočili menej dôveryhodné modely. To by okrem hlavnej funkcie mohlo výrazne urýchliť celý proces predikovania. Ďalším možným spôsobom je ukladanie všetkých predikcií, prípadne aj s ich hodnotami. Vhodnosť ktoréhokoľvek riešenia však záleží na spôsobe využitia systému.

### Príprava a spustenie systému

Pred spustením aplikácie je najprv nutné nainštalovať nástroj tshark a následne aj potrebné Python balíčky zo súboru *requirements.txt*. Balíčky PyQt5 a PyQtChart musia byť pri tom na rovnakej verzii. Systém je možné spustiť pomocou súboru *main.py*. Podľa systému, na ktorom má systém bežať, bude možno potrebné zvoliť inú zostavu balíčka tensorflow. Pred spustením systému je vhodné nahliadnuť do konfiguračného súboru z názvom *config.json*. Ten obsahuje nasledujúce možnosti pre nastavenie systému:

- packets_count – počet paketov, ktorých dáta aktivované modely používajú na predikovanie,
- predict_timeout – určuje minimálni čas, ktorý musí prejsť medzi jednotlivými predikciami jedného spojenia,
- recycle_period – perióda s ktorou je spúšťaný proces mazania starých paketov; určuje však aj minimálny čas, ktorý musí prejsť od posledného priradenia paketu k spojeniu, aby mohli byť pakety spojenia vymazané,
- path_to_database – relatívna/absolútna cesta k databáze,
- entities_table_name – názov tabuľky s modelmi v databáze (v rámci prace je ním „app_protocols"),
- import_pcap_max_bulk_size – maximálna veľkosť hromadných predikcií pri importovaní PCAP súboru (0 – neobmedzený).

### Grafické užívateľské rozhranie

Na ovládanie systému, demonštráciu jeho výstupov a pridávanie či úpravu protokolov bolo vytvorené za pomoci balíčka PyQt5 [52] grafické užívateľské rozhranie. To pozostáva z menu tlačítka a piatich kariet obsahujúcich rôzne prvky. Prvá karta nesie názov „Capture" (viď obrázok 6.6) a obsahuje jednak ovládacie prvky zachytávania paketov vrátane poľa na zvolenie jedného z dostupných sieťových rozhraní

zo zoznamu a štart/stop tlačítka, ako aj zobrazenie tabuľky „connections” z databázy a jej príslušné obnovovacie tlačítko. Druhá karta s názvom „Stations” (viď obrázok 6.7) slúži len na zobrazenie tabuľky „stations” z databázy. Tretia karta s názvom „Protocols” (viď obrázok 6.8) slúži na modifikáciu nastavení modelov v tabuľke s nastaveným názvom (v rámci práce „app_protocols”). V tejto karte je možné prepisovať jednotlivé polia priamo v tabuľke, pomocou tlačidiel pridávať a odoberať záznamy a vykonané zmeny buď zrušiť alebo zapísať do databázy. Štvrtá karta s názvom „Raw Output” (viď obrázok 6.9) obsahuje textové pole slúžiace na výpis textu do užívateľského rozhrania z ľubovoľnej súčasti systému. Vďaka tejto karte je možné v reálnom čase sledovať, čo sa v systéme deje. Do poľa sú mimo iné vypisované aj chybové hlášky, ako napr. že je databáza uzamknutá iným procesom mimo systému. Posledná piata karta s názvom „Chart” (viď obrázok 6.10) obsahuje koláčový graf so štatistikou momentálneho stavu predikcií všetkých spojení. Vyobrazené sú všetky výsledné predikcie, ktoré boli priradené aspoň jednému spojeniu, čiže protokol bez spojenia v grafe zobrazený nebude. Vedľa grafu sa nachádza aj tabuľka zreteľnejšie prezentujúca vyobrazenú štatistiku.

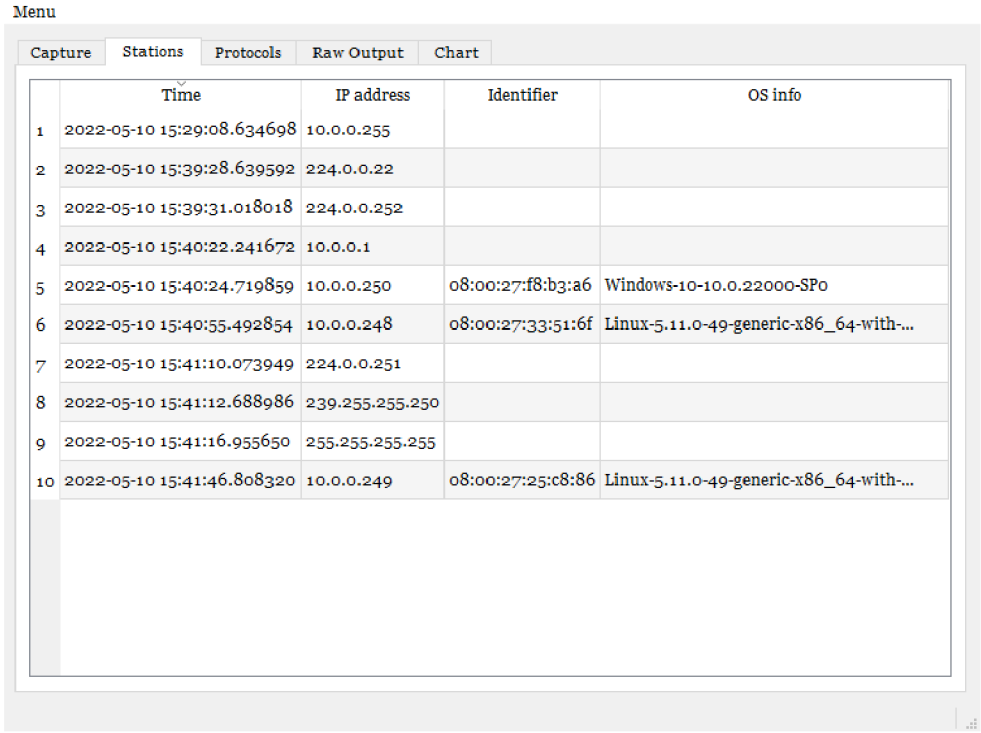
Menu

Capture | Stations | Protocols | Raw Output | Chart

Ethernet | Start | Stop | Refresh

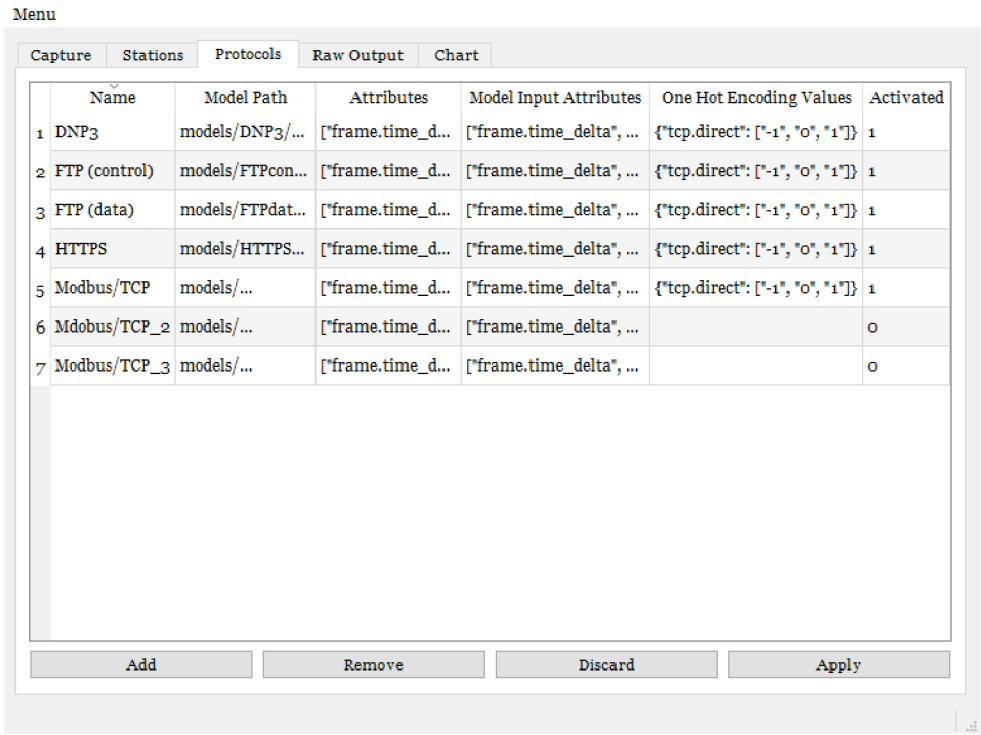
| | Time | IP address A | IP address B | Port A | Port B | Trans. Protocol | Prediction |
|---|---|---|---|---|---|---|---|
| 1 | 2022-05-08 16:08:25.376637 | 10.0.0.249 | 10.0.0.250 | 58174 | 443 | TCP | HTTPS |
| 2 | 2022-05-08 16:08:26.112760 | 10.0.0.249 | 10.0.0.250 | 39238 | 2121 | TCP | FTP (control) |
| 3 | 2022-05-08 16:08:27.276062 | 10.0.0.249 | 10.0.0.250 | 45960 | 64520 | TCP | FTP (data) |
| 4 | 2022-05-08 16:08:28.342632 | 10.0.0.249 | 10.0.0.250 | 39242 | 2121 | TCP | FTP (control) |
| 5 | 2022-05-08 16:08:29.054168 | 10.0.0.249 | 10.0.0.250 | 55712 | 64521 | TCP | FTP (data) |
| 6 | 2022-05-08 16:08:30.099142 | 10.0.0.249 | 10.0.0.250 | 34684 | 64522 | TCP | FTP (data) |
| 7 | 2022-05-08 16:08:30.888666 | 10.0.0.249 | 10.0.0.250 | 34494 | 64523 | TCP | FTP (data) |
| 8 | 2022-05-08 16:08:33.099688 | 10.0.0.249 | 10.0.0.250 | 39250 | 2121 | TCP | FTP (control) |
| 9 | 2022-05-08 16:08:33.991133 | 10.0.0.249 | 10.0.0.250 | 34228 | 64524 | TCP | FTP (data) |
| 10 | 2022-05-08 16:08:37.065933 | 10.0.0.249 | 10.0.0.250 | 35420 | 64525 | TCP | FTP (data) |
| 11 | 2022-05-08 16:08:37.807019 | 10.0.0.249 | 10.0.0.250 | 35666 | 64526 | TCP | FTP (data) |
| 12 | 2022-05-08 16:08:38.464579 | 10.0.0.249 | 10.0.0.250 | 54316 | 64527 | TCP | FTP (data) |
| 13 | 2022-05-08 16:08:38.973384 | 10.0.0.249 | 10.0.0.250 | 53098 | 64528 | TCP | FTP (data) |
| 14 | 2022-05-08 16:09:09.535521 | 10.0.0.249 | 10.0.0.248 | 47745 | 502 | TCP | Modbus/TCP |
| 15 | 2022-05-08 16:09:23.081275 | 10.0.0.249 | 10.0.0.248 | 51351 | 20000 | TCP | DNP3 |

Obr. 6.6: Karta užívateľského rozhrania „Capture”.



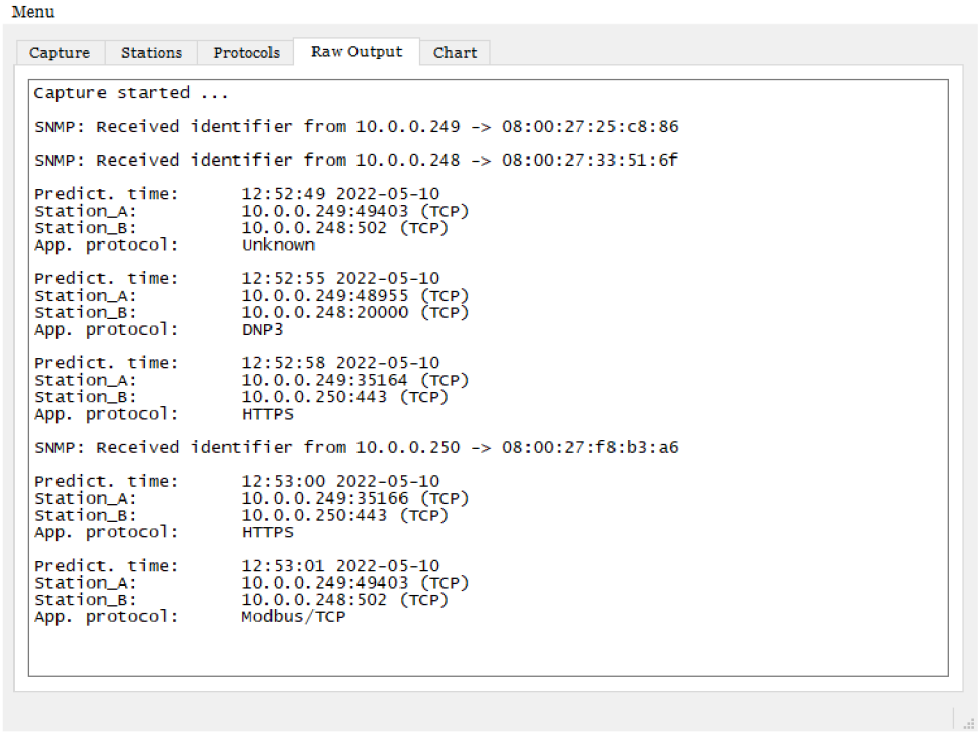

| | Time | IP address | Identifier | OS info |
|---|---|---|---|---|
| 1 | 2022-05-10 15:29:08.634698 | 10.0.0.255 | | |
| 2 | 2022-05-10 15:39:28.639592 | 224.0.0.22 | | |
| 3 | 2022-05-10 15:39:31.018018 | 224.0.0.252 | | |
| 4 | 2022-05-10 15:40:22.241672 | 10.0.0.1 | | |
| 5 | 2022-05-10 15:40:24.719859 | 10.0.0.250 | 08:00:27:f8:b3:a6 | Windows-10-10.0.22000-SP0 |
| 6 | 2022-05-10 15:40:55.492854 | 10.0.0.248 | 08:00:27:33:51:6f | Linux-5.11.0-49-generic-x86_64-with-... |
| 7 | 2022-05-10 15:41:10.073949 | 224.0.0.251 | | |
| 8 | 2022-05-10 15:41:12.688986 | 239.255.255.250 | | |
| 9 | 2022-05-10 15:41:16.955650 | 255.255.255.255 | | |
| 10 | 2022-05-10 15:41:46.808320 | 10.0.0.249 | 08:00:27:25:c8:86 | Linux-5.11.0-49-generic-x86_64-with-... |

Obr. 6.7: Karta užívateľského rozhrania „Stations”.

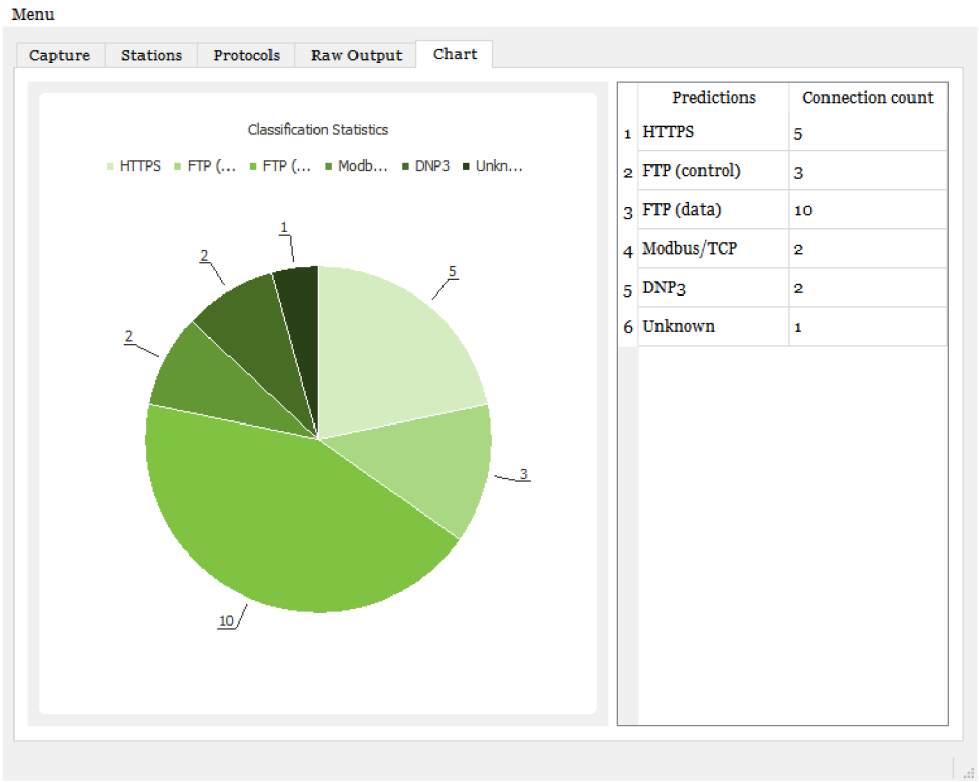



| | Name | Model Path | Attributes | Model Input Attributes | One Hot Encoding Values | Activated |
|---|---|---|---|---|---|---|
| 1 | DNP3 | models/DNP3/... | ["frame.time_d... | ["frame.time_delta", ... | {"tcp.direct": ["-1", "0", "1"]} | 1 |
| 2 | FTP (control) | models/FTPcon... | ["frame.time_d... | ["frame.time_delta", ... | {"tcp.direct": ["-1", "0", "1"]} | 1 |
| 3 | FTP (data) | models/FTPdat... | ["frame.time_d... | ["frame.time_delta", ... | {"tcp.direct": ["-1", "0", "1"]} | 1 |
| 4 | HTTPS | models/HTTPS... | ["frame.time_d... | ["frame.time_delta", ... | {"tcp.direct": ["-1", "0", "1"]} | 1 |
| 5 | Modbus/TCP | models/... | ["frame.time_d... | ["frame.time_delta", ... | {"tcp.direct": ["-1", "0", "1"]} | 1 |
| 6 | Mdobus/TCP_2 | models/... | ["frame.time_d... | ["frame.time_delta", ... | | 0 |
| 7 | Modbus/TCP_3 | models/... | ["frame.time_d... | ["frame.time_delta", ... | | 0 |

Obr. 6.8: Karta užívateľského rozhrania „Protocols”.




Obr. 6.9: Karta užívateľského rozhrania „Raw Output".




Obr. 6.10: Karta užívateľského rozhrania „Chart".

## 6.4 Testovanie systému

Prvým testom systému bolo rozpoznávanie protokolu za behu všetkých simulácií. Bol testovaný model protokolu Modbus/TCP v troch variantoch. Varianty boli učené na rovnakých negatívnych dátach. Pozitívne dáta však boli v jednom prípade len z verejných zdrojov, v druhom prípade len z prenosu zachyteného za behu simulácie a v treťom prípade bol použitý už vytvorený model (viď podkapitola 5.8) učený na všetkých dátach. Nové dva modely boli učené rovnako ako už vytvorený model s výnimkou počtu epoch (viď tabuľka 6.1). Testom je možné porovnať, ako presný je model v klasifikovaní jemu známeho využitia daného protokolu oproti klasifikovaniu využitia neznámeho. Systém bol nastavený tak, aby prešlo medzi každou predikciou daného spojenia aspoň 10 sekúnd. Test bežal pre každý model 40 minút, za ktorých systém v priemere vykonal 210 predikcií. Výsledky testu vzhľadom k výpisu systému boli nasledovné:

- Dáta z verejných zdrojov – model klasifikoval správne len prvých 8 zachytených paketov, čiže len jedna predikcia bola správna,
- Dáta simulácie – model mal všetky predikcie správne, až na prvú predikciu,
- Všetky dáta – rovnaké výsledky ako pri prvom modely.

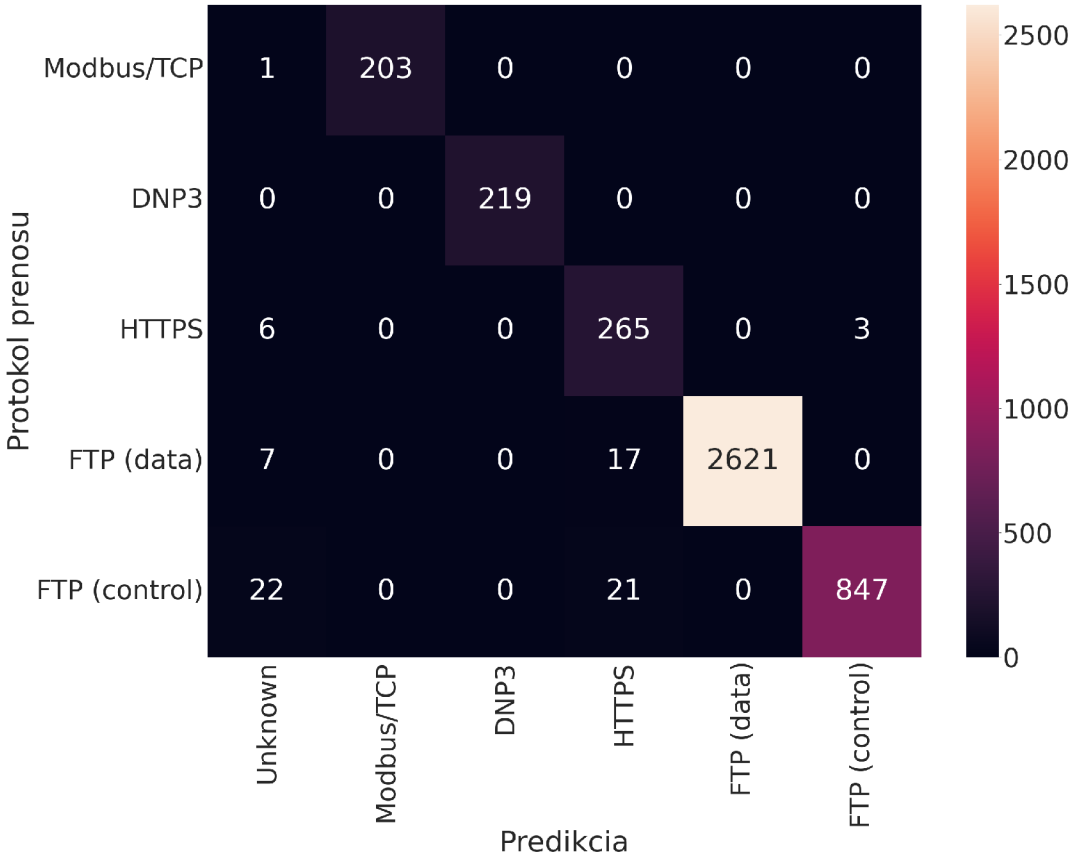
Druhý test využíval tie isté modely ako test prvý, avšak boli evaluované dáta z verejných zdrojov, a to pomocou funkcie systému umožňujúcej spracovať PCAP súbor a hromadne predikovať využitý protokol na všetkých paketoch každého spojenia, ktoré sa v súbore nachádza (bola nastavená neobmedzená hromadná predpoveď). Z dôvodu výpočtovej a pamäťovej náročnosti bol vybraný jeden PCAP súbor z repozitáru [31] so zachytenými Modbus/TCP prenosmi v rozmedzí jednej hodiny. Systém vykonal zakaždým 1244 predikcií s nasledovnými výsledkami:

- Dáta z verejných zdrojov – model mal len jednu predikciu nesprávnu,
- Dáta simulácie – model mal všetky predikcie nesprávne,
- Všetky dáta – rovnaké výsledky ako pri prvom modely.

Tab. 6.1: Nastavenie a presnosť Modbus/TCP modelu s jednotlivými datasetmi.

| **Dáta použité na učenie** | **Počet skrytých vrstiev** | **Počet epoch** | **Presnosť (testovacia evaluácia)** | **Presnosť (Test 1)** | **Presnosť (Test 2)** |
|---|---|---|---|---|---|
| Dáta z verejných zdrojov | 2 | 100 | 99,96 % | 0,48 % | 99,92 % |
| Dáta simulácie | 2 | 10 | 100 % | 99,52 % | 0 % |
| Všetky dáta | 2 | 100 | 99,96 % | 99,52 % | 99,92 % |

Tretí test bol zameraný na rozpoznávaciu schopnosť systému ako celku. Bolo aktivovaných všetkých 5 modelov spomínaných v podkapitole 5.8 a systém bol opäť spustený za behu všetkých simulácií. Na základe tohto testu je možné pozorovať „zmätenie" predikcie systému, a teda mieru nesprávnosti predikcie jednotlivých modelov voči jednotlivým protokolom prenosu. Systém rovnako nastavený ako v prvom teste bežal približne 3 hodiny. Výsledok testu reprezentuje tzv. matica zámeny (z ang. confusion matrix) viditeľná na obrázku 6.11. Na vykreslenie matice boli použité balíčky seaborn [53] a matplotlib [54].




Obr. 6.11: Matica zámeny predikcie systému počas testovania.

# Záver

V rámci práce boli naštudované a teoreticky rozobrané priemyselné protokoly Modbus/TCP, DNP3 a Profinet, vrátane možností ich simulácie. Taktiež boli popísané bežné aplikačné protokoly HTTP, FTP, SMTP ako aj protokol SNMP, ktorý bol vo výslednom systéme použitý na identifikáciu zariadení. Ďalej boli rozobrané vybrané techniky strojového učenia, konkrétne umelé neurónové siete, algoritmus C4.5 a metódy podporných vektorov. Taktiež bola spracovaná základná teória databáz a implementácia SQLite3, ktorá bola vybraná ako najvhodnejšia pre finálne riešenie.

Na zozbieraných a spracovaných dátach boli natrénované a otestované rôzne klasifikačné modely spomenutých algoritmov pre protokol Modbus/TCP. Na základe testov bol vybraný model umelých neurónových sietí ako najvhodnejší na využitie v rozpoznávacom systéme. Ďalej bolo zostavené experimentálne pracovisku vo virtuálnom prostredí, kde boli simulované využitia vybraných protokolov, a to Modbus/TCP, DNP3, HTTPS a FTP. Prenos zachytený v rámci pracoviska bol využitý spolu so zozbieranými dátami pri trénovaní a testovaní finálnych modelov výsledného riešenia. Taktiež bol navrhnutý systém na rozpoznávanie zariadení a použitých protokolov na sieti z pohľadu jeho hlavnej rozpoznávacej činnosti. Systém so všetkými jeho súčasťami bol nakoniec implementovaný aj s grafickým užívateľským rozhraním na základe objektovo orientovaného návrhu. Až na GUI bolo celé riešenie vyhotovené obecne s myšlienkou, že systém bude môcť byť využitý na ľubovoľnú klasifikáciu zachytených paketov s využitím atribútov, ktoré poskytuje nástroj tshark. V priebehu práce bolo preto riešenie premenované z „Network Protocol Classifier" na „Network Packet Classifier" pre vyzdvihnutie jeho univerzality.

Z testovania modelov a systému je možné vyvodiť, že je výrazne odlišnou úlohou naučiť model rozpoznávať konkrétne využitie protokolu, než protokol ako taký. Najmä ak existuje mnoho rôznych spôsobov jeho využitia. O tom vypovedajú najmä výsledky prvého testu výsledného riešenia. Pre model učený na rozličných zozbieraných prenosoch pomocou protokolu Modbus/TCP bola úspešnosť rozpoznania neznámeho simulovaného využitia protokolu prakticky nulová. Na druhú stranu, výsledky testov ukazujú, že systém nemá problém rozpoznať známe využitie daného protokolu. Z výsledkov posledného testu systému je možné vyvodiť, že systém mal najväčší problém s rozpoznaním kontrolného spojenia protokolu FTP, pričom najčastejšie falošne predikoval použitý protokol spojení ako HTTPS. Pre celkové zhodnotenie výsledného riešenia je okrem vyzdvihnutého problému s rivalitou medzi modelmi dobré spomenúť jeho pamäťovú a výpočtovú náročnosť. Pri vysokej intenzite prenosu dokáže ľahko vyplytvať zdroje pomalšieho zariadenia. Riešením tohto problému by bola voľba jedného z programovacích jazykov nižšej úrovne ako aj úspornejší management ukladaných dát.

# Literatúra


[1] Modbus Organization. MODBUS Application Protocol Specification V1.1b3. [online], 2012. URL: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[2] Modbus Organization. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. [online], 2012. URL: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

[3] Doug Lyons. ModRSsim2 download - SourceForge.net. [online], 2021. URL: https://sourceforge.net/projects/modrssim2/.

[4] Rossmann Engineering. EasyModbusTCP/UDP/RTU .NET download - SourceForge.net. [online], 2020. URL: https://sourceforge.net/projects/easymodbustcp/.

[5] Riptide IO Inc. GitHub - riptideio/pymodbus: A full modbus protocol written in python. [online], 2022. URL: https://github.com/riptideio/pymodbus.

[6] Ken Curtis. A DNP3 Protocol Primer. [online], 2005. URL: https://www.dnp.org/Portals/0/AboutUs/DNP3%20Primer%20Rev%20A.pdf.

[7] Nicholas R. Rodofile, Kenneth Radke, and Ernest Foo. DNP3 Network Scanning and Reconnaissance for Critical Infrastructure. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2843043.2843350.

[8] FreyrSCADA. DNP3 Protocol download - SourceForge.net. [online], 2021. URL: https://sourceforge.net/projects/dnp3/.

[9] ChargePoint Inc. GitHub - ChargePoint/pydnp3: Python bindings for opendnp3 libary. [online], 2019. URL: https://github.com/ChargePoint/pydnp3.

[10] Émile Grégoire and Adam Crain. GitHub - dnp3/opendnp3: DNP3 (IEEE-1815) protocol stack. Modern C++ with bindings for .NET and Java. [online], 2022. URL: https://github.com/dnp3/opendnp3.

[11] CDOAN. DNP3 Protocol Simulator - CDOAN. [online], 2020. URL: https://www.cdoancal.com/dnp3.

[12] PROFIBUS & PROFINET International. PROFINET System Description. [online], 2014. URL: http://us.profinet.com/wp-content/uploads/2012/11/PROFINET_SystemDescription_ENG_2014_web.pdf.

[13] MDN contributors. HTTP documentation – DevDocs. [online], 2021. URL: https://devdocs.io/http/.

[14] Jon Postel and Joyce K. Reynolds. File Transfer Protocol. RFC 959, 10 1985. doi:10.17487/RFC0959.

[15] Dr. John C. Klensin. Simple Mail Transfer Protocol. RFC 5321, 10 2008. doi:10.17487/RFC5321.

[16] Dr. Jeff D. Case, Russ Mundy, David Partain, and Bob Stewart. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410, 12 2002. doi:10.17487/RFC3410.

[17] Muhammad Shafiq, Xiangzhan Yu, Asif Laghari, Lu Yao, Nabin Karn, and Foudil Abdessamia. Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms, 10 2016. doi:10.1109/CompComm.2016.7925139.

[18] L´eon Bottou. Stochastic Gradient Learning in Neural Networks. [online], 1991. URL: https://leon.bottou.org/publications/pdf/nimes-1991.pdf.

[19] Anil K. Jain, Jianchang Mao, and Kottappuram M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, 1996. doi:10.1109/2.485891.

[20] Nilima Khanna. J48 Classification (C4.5 Algorithm) in a Nutshell. [online], 2021. URL: https://medium.com/@nilimakhanna1/j48-classification-c4-5-algorithm-in-a-nutshell-24c50d20658e.

[21] Marti A. Hearst, Sue T. Dumais, Edgar Osuna, John C. Platt, and Bernhard Scholkopf. Support Vector Machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998. doi:10.1109/5254.708428.

[22] Oracle. What Is a Database. [online], 2020. URL: https://www.oracle.com/database/what-is-database/.

[23] Oracle. What is a Relational Database. [online], 2021. URL: https://www.oracle.com/database/what-is-a-relational-database/.

[24] Peter Loshin. What is Structured Query Language (SQL)? [online], 2022. URL: https://www.techtarget.com/searchdatamanagement/definition/SQL.

[25] SQLite Home Page, 2022. URL: https://www.sqlite.org/index.html.

[26] Rossmann Engineering. GitHub - rossmann-engineering_EasyModbusTCP.PY Modbus TCP, Modbus UDP and Modbus RTU client library for Python implementations. [online], 2021. URL: https://github.com/rossmann-engineering/EasyModbusTCP.PY.

[27] The Wireshark team. Wireshark - packet analyzer. [online], 2021. URL: https://www.wireshark.org.

[28] F5. Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX, 2022. URL: https://www.nginx.com/.

[29] Software Freedom Conservancy. Selenium, 2022. URL: https://www.selenium.dev/.

[30] Tim Kosse. FileZilla - The free FTP solution, 2022. URL: https://filezilla-project.org/.

[31] Ivo Frazão, Pedro Henriques Abreu, Tiago Cruz, Hélder Araújo, and Paulo Simões. ICS Cybersecurity PCAP repository. [online], 9 2018. doi:10.1007/978-3-030-05849-4_19.

[32] Information Trust Institute (UIUC). ICS Security Tools, Tips, and Trade. [online], 2021. URL: https://github.com/ITI/ICS-Security-Tools.

[33] Antoine Lemay. Modbus_dataset. [online], 2016. URL: https://github.com/antoine-lemay/Modbus_dataset.

[34] Netresec AB. Public PCAP files for download. [online], 2021. URL: https://www.netresec.com/?page=PcapFiles.

[35] Ross Jacobs. Tshark: tshark.dev. [online], 2019. URL: https://tshark.dev/.

[36] The Wireshark team. mergecap - the wireshark network analyzer. [online], 2021. URL: https://www.wireshark.org/docs/man-pages/mergecap.html.

[37] The pandas development team. pandas-dev/pandas: Pandas, February 2020. doi:10.5281/zenodo.3509134.

[38] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant.

Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi:10.1038/s41586-020-2649-2.

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[40] Plotly Technologies Inc. Collaborative data science, 2015. URL: https://plot.ly.

[41] Google LLC. Welcome To Colaboratory - Colaboratory, 2022. URL: https://colab.research.google.com/.

[42] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: https://www.tensorflow.org/.

[43] Sefik I. Serengil. ChefBoost: A Lightweight Boosted Decision Tree Framework, 10 2021. doi:10.5281/zenodo.5576203.

[44] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderSVM: A Fast SVM Library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801, 2018. URL: https://github.com/Xtra-Computing/thundersvm/blob/master/thundersvm-full.pdf.

[45] Jasper Bongertz. The Megalodon Challenge - Packet-Foo - Network Packet Capture and Analysis, 2015. URL: https://blog.packet-foo.com/2015/07/the-megalodon-challenge/.

[46] The Wireshark team. SampleCaptures - The Wireshark Wiki, 2020. URL: https://wiki.wireshark.org/SampleCaptures.

[47] Mauro Conti, Denis Donadel, and Federico Turrin. A survey on industrial control system testbeds and datasets for security research. *IEEE Communications Surveys Tutorials*, 23(4):2248–2294, 2021. doi:10.1109/COMST.2021.3094360.

[48] Obinna Igbe, Ihab Darwish, and Tarek Saadawi. Deterministic dendritic cell algorithm application to smart grid cyber-attack detection. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 199–204, 2017. doi:10.1109/CSCloud.2017.12.

[49] Dor Green. KimiNewt_pyshark: Python wrapper for tshark, allowing python packet parsing using wireshark dissectors, 2021. URL: https://github.com/KimiNewt/pyshark.

[50] Ilya Etingof. etingof_pysnmp: Python SNMP library, 2019. URL: https://github.com/etingof/pysnmp.

[51] Yueqiu Jiang, Kun Sun, Wenbo Zhang, and Daozhu Zhang. A SNMPv3 Replay Protection Scheme Used in Space Network Based on Random Number. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 2, pages 378–380, 2009. doi:10.1109/HIS.2009.189.

[52] Riverbank Computing Limited. Riverbank Computing - Introduction, 2022. URL: https://www.riverbankcomputing.com/software/pyqt/.

[53] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi:10.21105/joss.03021.

[54] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

# Zoznam symbolov a skratiek

| | |
|---|---|
| **ADU** | Application Data Unit |
| **AES** | Advanced Encryption Standard |
| **ANN** | Artificial Neural Network |
| **CPU** | Central Processing Unit |
| **DBMS** | Database Management System |
| **DHCP** | Dynamic Host Configuration Protocol |
| **DNP3** | Distributed Network Protocol 3 |
| **ESMTP** | Extended Simple Mail Transfer Protocol |
| **FTP** | File Transfer Protocol |
| **FTPS** | File Transfer Protocol Secure |
| **GPU** | Graphics Processing Unit |
| **GUI** | Graphical User Interfac |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **IO** | Input/Output |
| **IMAP** | Internet Message Access Protocol |
| **IP** | Internet Protocol |
| **MIB** | Management Information Base |
| **MITM** | Man In The Middle |
| **NAT** | Network Address Translation |
| **NPC** | Network Packet Classifier |
| **OID** | Object Identifier |
| **PCAP** | Packet Capture |
| **PDU** | Protocol Data Unit |

| | |
|---|---|
| **POP3** | Post Office Protocol 3 |
| **QUIC** | Quick UDP Internet Connections |
| **ReLU** | Rectified Linear Unit |
| **SARSA** | State–action–reward–state–action |
| **SMTP** | Simple Mail Transfer Protocol |
| **SNMP** | Simple Network Management Protocol |
| **SQL** | Structured Query Language |
| **SSL** | Secure Sockets Layer |
| **SVM** | Support Vector Machine |
| **TCP** | Transmission Control Protocol |
| **TLS** | Transport Layer Security |
| **UDP** | User Datagram Protocol |
| **URL** | Uniform Resource Locator |
| **VM** | Virtual Machine |

# A Obsah elektronickej prílohy

Elektronická príloha obsahuje implementáciu rozpoznávacieho systému, a teda aplikáciu Network Packet Classifier, vrátane ďalších súborov súvisiacich s aplikáciou. Pre požiadavky na spustenie aplikácie, spôsob jej použitia či ďalšie užitočné informácie viď súbor *README.md*. Aplikácia bola testovaná a je plne funkčná na operačných systémoch Ubuntu 21.04 a Windows 10.

```
/ ..... koreňový adresár priloženého archívu
└── NPC
    ├── export ..... vzorový export databázy
    │   ├── app_protocols.csv
    │   ├── connections.csv
    │   └── stations.csv
    ├── models ..... modely spomenuté v rámci práce
    │   ├── DNP3
    │   │   └── DNP3_1_100ep_2048bs.h5
    │   ├── FTPcon
    │   │   └── FTPcon_1_50ep_2048bs.h5
    │   ├── FTPdata
    │   │   └── FTPdata_1_100ep_2048bs.h5
    │   ├── HTTPS
    │   │   └── HTTPS_1_200ep_2048bs.h5
    │   └── MODBUS_TCP
    │       ├── VAR_TEST
    │       │   ├── MODBUS_VAR_TEST_CAP_1_100ep_2048bs.h5
    │       │   └── MODBUS_VAR_TEST_SIM_1_10ep_2048bs.h5
    │       ├── MODBUS_1_100ep_2048bs.h5
    │       ├── MODBUS_2_200ep_2048bs.h5
    │       └── MODBUS_3_200ep_2048bs.h5
    ├── capture_control.py
    ├── config.json ..... konfiguračný súbor aplikácie
    ├── database_control.py
    ├── LICENSE ..... súbor s textom licencie
    ├── main.py ..... skript na spustenie aplikácie
    ├── npc.py
    ├── NPC_database.db ..... súbor databázy
    ├── packets_classifier.py
    ├── predict_entity.py
    ├── README.md ..... súbor s užitočnými informáciami o aplikácii
    ├── requirements.txt ..... súbor s požadovanými Python balíčkami
    ├── snmp.py
    └── ui_main_window.py
```