UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2018 Lukáš Kutík

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Generátor síťového provozu pro testování OpenFlow funkcionalit

Lukáš Kutík

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš Kutík**
Osobní číslo:	**I14124**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Generátor síťového provozu pro testování OpenFlow funkcionalit**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce bude analyzovat existující nástroje pro generování síťového provozu. Autor se zaměří zejména na aplikace v oblasti IoT a Smart Grid sítí. V praktické části pak bude implementován multiplatformní grafický nástroj pro generování síťového provozu. Tento nástroj bude implementován jako aplikace typu klient-server a bude umožňovat dynamické nastavení polí podporovaných protokolem OpenFlow 1.3. Aplikace bude obsahovat předpřipravené šablony pro generování provozu různých IoT protokolů (například protokoly normy IEC 61850).

Rozsah grafických prací:

Rozsah pracovní zprávy: **35**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

ONF Open Networking Foundation, OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04), June 25, 2012 [online]. Dostupný z: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf OASIS: MQTT Version 3.1.1 [online]. 2014 [cit. 2016-10-20]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html Core IEC Standards [online]. IEC, 2016 [cit. 2016-10-20]. Dostupné z: http://www.iec.ch/smartgrid/standards/

Vedoucí bakalářské práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2017**

Termín odevzdání bakalářské práce: **12. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 23. 3. 2018

Lukáš Kutík

PODĚKOVÁNÍ

Mé poděkování patří Ing. Soně Neradové Ph.D. za cenné rady, trpělivost a ochotu, kterou mi věnovala v průběhu zpracování bakalářské práce.

ANOTACE

Tato práce se zabývá internetem věcí a nástroji pro generování síťového provozu. V teoretické části jsou vybrány některé již existující nástroje pro generování síťového provozu, které jsou zde popsány a posléze i vzájemně porovnány. Práce také představuje internet věcí a některé v něm používané protokoly. Praktickou část tvoří vlastní implementace aplikací pro generování i zachytávání síťového provozu.

KLÍČOVÁ SLOVA

Generátor síťového provozu, aplikace, Java, internet věcí, chytré sítě

TITLE

Network traffic generator for testing OpenFlow functionalities

ANNOTATION

This thesis discusses Internet of Things and tools for generating network traffic. In the theoretical part of the paper, several existing tools for generating network traffic are described and compared with each other. The thesis also introduces Internet of Things and several protocols, that are involved in Internet of Things. The practical part comprises of implementation of applications for generating and capturing network traffic.

KEYWORDS

Network traffic generator, application, Java, Internet of Things, Smart Grid

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
ARP	Address Resolution Protocol
BSON	Binary JSON
CoAP	Constrained Application Protocol
DNS	Domain Name System
GNU GPLv3	GNU General Public License
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol version 6
IGMP	Internet Group Management Protokol
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
LLAP	Lightweight Local Automation Protocol
LLC	Logical Link Control
MAC	Media Access Control
MLD	Multicast Listener Discovery
MQTT	MQ Telemetry Transport
MQTT-SN	MQTT For Sensor Networks
SIP	Session Initiation Protocol
SNAP	Subnetwork Access Protocol
TCP	Transmission Control Protocol
TR	Technical Report
TS	Technical Specification
TTL	Time To Live
UDP	User Datagram Protocol
UML	Unified Modeling Language

ÚVOD

Generátor síťového provozu nahrazuje reálný síťový tok a umožňuje testovat a měřit síťové parametry. Představuje užitečný nástroj pro správce sítí umožňující uplatnit získané hodnoty pro úpravu síťové infrastruktury. Cílem teoretické části této práce je vyhledat softwarové síťové generátory a porovnat jejich funkcionality.

Praktická část se zaměří na vytvoření vlastního síťového generátoru. Cílem je vytvořit multiplatformní grafickou aplikaci fungující jako aplikace typu klient-server a umožňující dynamické nastavení polí různých protokolů. Hlavní motivací pro tvorbu takového nástroje je zahrnout podporu vybraných protokolů patřících do internetu věcí, což je koncept, který v poslední době zažívá obrovský růst díky technologickému pokroku a zasahuje do všech oblastí našeho života.

Tato práce se v první kapitole věnuje několika vybraným generátorům síťového provozu a obsahuje rovněž jejich vzájemné porovnání. Druhou kapitolu tvoří představení internetu věcí a vybraných protokolů patřících do internetu věcí. Třetí kapitola navazuje popisem tzv. chytrých sítí, což je koncept související s internetem věcí. Poslední kapitola se skládá z popisu aplikace vytvořené v praktické části a z dokumentace k této aplikaci.

1 VYBRANÉ SÍŤOVÉ GENERÁTORY

Tato kapitola představuje vybrané existující nástroje pro generování síťového provozu. V jednotlivých podkapitolách jsou postupně popsány a zhodnoceny následující síťové generátory: Ostinato, Scapy, Colasoft Packet Builder.

1.1 Generátory síťového provozu

Generátory síťového provozu jsou nástroje, které slouží k simulaci zátěže sítě a síťových prvků tím, že do nich posílají generovaný síťový provoz. Používají se proto pro testování sítě nebo síťových prvků, a využijí je tak především správci sítě.

1.2 Ostinato

Ostinato[1] je multiplatformní generátor a analyzátor síťového provozu. Jedná se o otevřený software licencovaný pod GNU GPLv3. Nejnovější verze je verze 0.8 z června roku 2016. Od určité verze je Ostinato placené, ale některé starší verze lze stáhnout zdarma (Ostinato Network Traffic Generator, c2017).

1.2.1 Instalace

Na operačních systémech Windows stačí stažený soubor rozbalit a poté spustit soubor *ostinato.exe*. Pro Windows je však ještě nutné mít nainstalovaný WinPcap[2], což je nástroj pro linkovou vrstvu síťového rozhraní umožňující zachytávání a odesílání síťových paketů (WinPcap, c2013).

1.2.2 Architektura

Ostinato se skládá ze dvou důležitých částí. Tou první je ovladač, který má na starost komunikaci s uživatelem. Ovladačem může být buďto grafické uživatelské rozhraní, nebo skript napsaný v programovacím jazyce Pythonu s využitím Python API[3]. Dále bude popisována verze s grafickým uživatelským rozhraním.

[1] Dostupné z http://ostinato.org/downloads
[2] Dostupné z https://www.winpcap.org/install/default.htm
[3] Dostupné z https://gumroad.com/l/ostpyold

Druhou nezbytnou částí je tzv. drone, který přijímá a vykonává instrukce zadané uživatelem přes ovladač. Ten si naopak z dronu bere různá data a tvoří z nich statistiky. Drone je při spuštění Ostinata automaticky spuštěn na pozadí, a uživatel se tak o něj nemusí starat. Zároveň ale drone potřebuje správcovská práva pro čtení síťových karet na daném stroji, proto je program nutné spustit s právy správce.

Nutno podotknout, že se tato architektura liší od klient-server architektury a Ostinato si s dronem nevyměňuje žádné pakety (Architecture · Ostinato User Guide, c2017). Drone dokonce může být spuštěn na jiném stroji než Ostinato. Komunikace však není šifrovaná ani autentifikovaná (How to generate network packets - Ostinato Packet/Traffic Generator, c2017).

Program Ostinato byl vyvinut v Pythonu, což je široce rozšířený vyšší programovací jazyk, který v roce 1991 navrhl Guido van Rossum (Technology at Ostinato Network Traffic Generator, c2017). Pro grafické uživatelské rozhraní byla využita knihovna Qt.

1.2.3 Funkce

Nástroj dokáže vytvářet pakety a posílat více proudů paketů s různými protokoly i intervaly mezi jednotlivými pakety. Díky grafickému uživatelskému rozhraní je snadno pochopitelný a snadno použitelný.

Ostinato podporuje následující protokoly:

- Ethernet/802.3/LLC SNAP,
- VLAN (s Q-in-Q),
- ARP, IPv4, IPv6, IP Tunneling,
- TCP, UDP, ICMPv4, ICMPv6, IGMP, MLD,
- Jakýkoliv textový protokol (HTTP, SIP atd.).

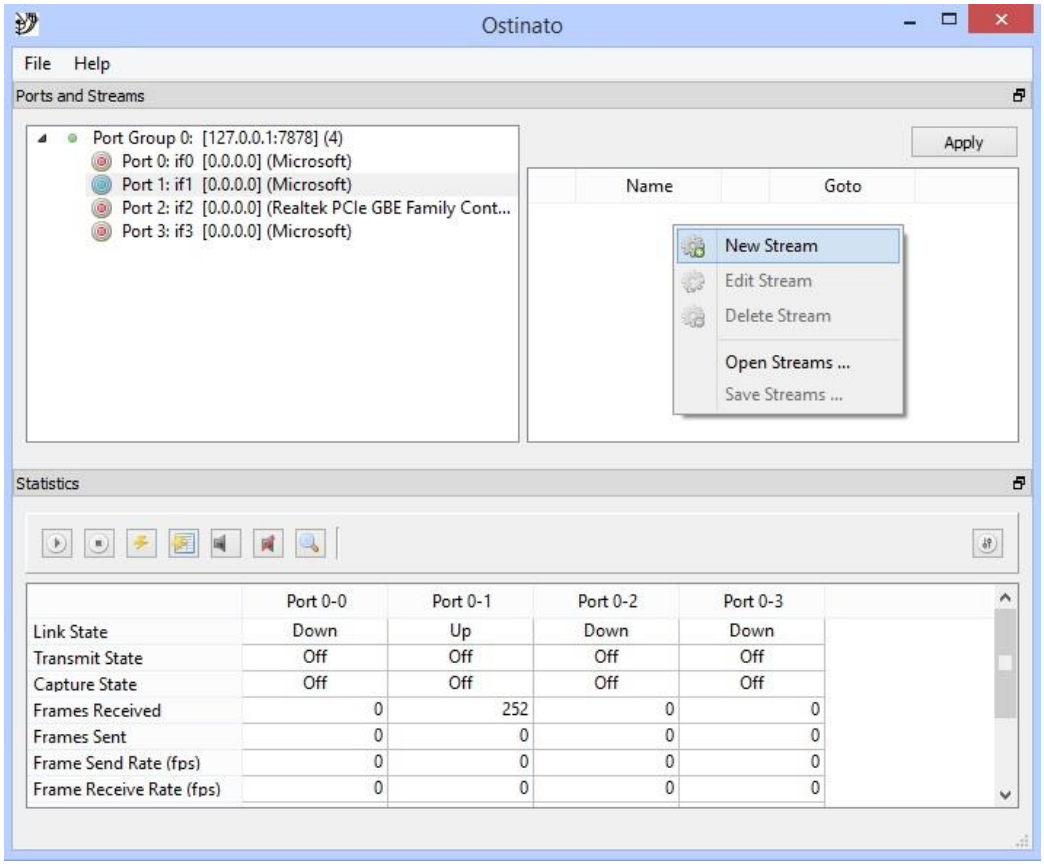
Ostinato umožňuje měnit libovolná pole, která daný protokol obsahuje. Další výhodou je, že se výborně doplňuje s velmi rozšířeným nástrojem Wireshark, který umožňuje zkoumat síťový provoz na daných rozhraních, a používá se proto pro analýzu počítačových sítí a ladění případných problémů. Toto propojení dvou zmíněných nástrojů lze vidět v Ostinatu, kde je jednou z možností zobrazit si přijaté pakety právě přímo ve Wiresharku.

Nástroj Ostinato nepodporuje stavové TCP spojení, a je tudíž bezstavový, což znamená, že si neuchovává žádné informace o předchozí vzájemné komunikaci. Rovněž nemůže být použit

pro generování a posílání falešného síťového provozu na webové stránky (Ostinato Network Traffic Generator, c2017).

1.2.4 Ukázka použití

Následuje obrázek (Obrázek 1) hlavního okna programu Ostinato, což je první okno, které se zobrazí po spuštění programu.

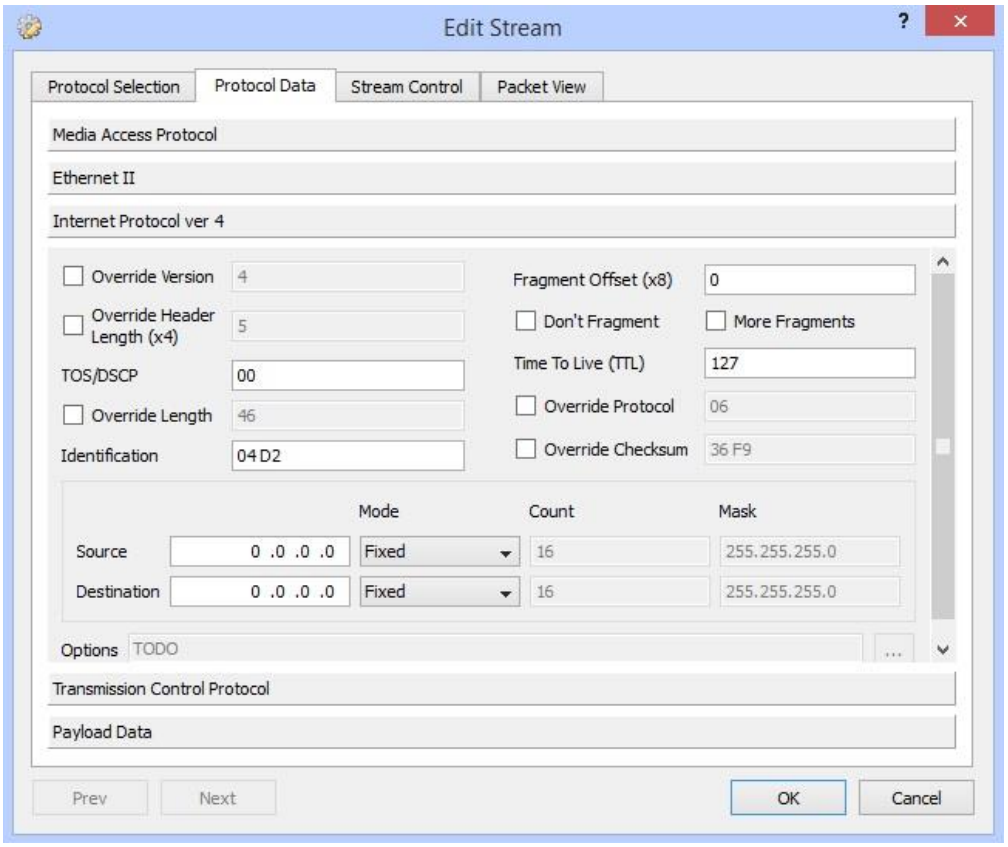


Obrázek 1 – Úvodní okno programu Ostinato

Zdroj: vlastní

Okno je rozděleno do dvou částí. Horní část okna obsahuje porty a datové proudy, ta dolní pak statistiky týkající se jednotlivých portů. Pro vyváření paketů se používá horní část způsobem, jenž je znázorněn na obrázku (Obrázek 1). Nejprve je v její levé části zvolen port, který bude použit. Poté lze v pravém okně pomocí levého tlačítka myši vyvolat nabídku, díky které lze vytvořit nový datový proud nebo upravit či odstranit proud existující.

Po vytvoření nového proudu lze dvojklikem otevřít jeho úpravu. Následující obrázek (Obrázek 2) zobrazuje podobu okna pro úpravu proudu.



Obrázek 2 – Úprava proudu v programu Ostinato

Zdroj: vlastní

V tomto okně se nachází vše potřebné pro vytvoření uživatelem požadovaného proudu paketů. Na obrázku (Obrázek 2) je vidět možnost nastavení různých polí protokolu IPv4. To je však pouze jeden z řady protokolů, které program Ostinato umožňuje nastavit. Nevýhodou je, že uživatel není informován v případě špatně vyplněného pole. Při stisknutí tlačítka *OK* se totiž špatně zadané hodnoty automaticky změní na určitou platnou hodnotu, ale děje se to bez jakéhokoliv informování uživatele.

Po nastavení proudu již stačí pouze v hlavním okně v části statistik zapnout odesílání a zachytávání paketů. Výsledek zachytávání pro daný port je poté možné otevřít v programu Wireshark.

1.3 Scapy

Scapy[4] je interaktivní nástroj pro manipulaci s pakety vytvořený v programovacím jazyce Pythonu. Na rozdíl od Ostinata, které má grafické uživatelské rozhraní, Scapy pracuje v příkazovém řádku.

1.3.1 Instalace

Zde je popsána instalace pro operační systém GNU/Linux, konkrétně pro distribuci Ubuntu 16.04.3 LTS s jádrem verze 4.10. Stažený soubor stačí pouze rozbalit a poté lze Scapy používat z rozbaleného adresáře jednoduchým příkazem:

```
./run_scapy
```

Pro správný běh programu jsou zapotřebí správcovská práva, proto je třeba použít příkazu sudo:

```
sudo /.run_scapy
```

Před samotným spuštěním je však nutné mít nainstalovaný Python a tcpdump. Tcpdump slouží k vypisování popisu obsahu paketu na daném síťovém rozhraní (Ubuntu Manpage: tcpdump - dump traffic on a network, c2010).

Rovněž je třeba se ujistit, že je na daném systému nastavena možnost CONFIG_PACKET. Toto lze ověřit v souboru s konfigurací jádra dané distribuce (Scapy portability page). Na testovacím stroji byla cesta k onomu souboru následující:

```
/boot/config-4.10.0-38-generic
```

1.3.2 Funkce

Scapy umožňuje vytvářet a poté posílat vlastní pakety, přičemž lze po odeslání paketu po určitou dobu čekat na odpověď. Pokud je při tvorbě paketu vynecháno sestavení některé z vrstev, Scapy tuto část paketu vytvoří automaticky a pole naplní výchozími (defaultními) hodnotami. Nástroj dále dokáže zachytávat internetovou komunikaci, zobrazovat zachycené pakety ze souboru s příponou *.pcap či přepsat zachycený paket před jeho následným odesláním.

Mezi pokročilejší funkce patří například možnost vytvořit si vlastní nový protokol, přesněji řečeno novou vrstvu.

[4] Dostupné z: http://www.secdev.org/projects/scapy/

1.3.3 Ukázka použití

V této podkapitole je představeno pouze základní vytváření a odesílání paketů nástrojem Scapy. Ukázky byly vytvořeny na distribuci Ubuntu 16.04.3 LTS s jádrem verze 4.10.

Následuje ukázka práce s IP protokolem:

```
>>> ip = IP()
>>> ip.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= hopopt
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
>>> ip.ttl = 80
>>> ip.dst = "192.168.0.12"
>>> ip.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 80
  proto= hopopt
  chksum= None
  src= 192.168.0.101
  dst= 192.168.0.12
  \options\
```

Po vytvoření proměnné *ip* lze přes tuto proměnou zobrazit strukturu IP protokolu. Přes názvy daných polí pak lze změnit hodnoty polí protokolu. V ukázce je tímto způsobem změněno pole TTL a cílová IP adresa.

Obdobně lze pracovat s libovolným jiným protokolem, což je dokázáno následující ukázkou, která pracuje s protokolem TCP:

```
>>> tcp = TCP()
>>> tcp.display()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
```

```
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
>>> tcp.dport = 8888
>>> tcp.display()
###[ TCP ]###
  sport= ftp_data
  dport= 8888
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
```

Zobrazit lze i oba protokoly najednou následujícím příkazem, který zobrazí oba nastavené protokoly uložené v proměnných *tcp* a *ip*:

```
(tcp/ip).display()
```

Příkaz, který výše vytvořený paket odešle:

```
>>> send(ip/tcp)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
```

V tomto případě byl odeslán paket složený z IP a TCP protokolu. Byla tedy nastavena pouze síťová a transportní vrstva. Nebyla zde nastavena spojová vrstva. Program ji v takovém případě vytvoří bez účasti uživatele a doplní defaultními hodnotami. Z tohoto důvodu lze v předchozí ukázce vidět upozornění, že cílová MAC adresa nebyla nalezena a byla nahrazena všesměrovou adresou. Paket byl nicméně odeslán úspěšně.

Nevýhodou nástroje Scapy je nedostatečné ověřování uživatelem doplňovaných hodnot, jak ilustruje následující ukázka:

```
>>> ether = Ether()
>>> ether.display()
###[ Ethernet ]###
WARNING: Mac address to reach destination not found. Using broadcast.
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:00:00:00
  type= 0x9000
>>> ether.src = 12:12:12:12:12:12
  File "<console>", line 1
    ether.src = 12:12:12:12:12:12
                  ^
SyntaxError: invalid syntax
>>> ether.src = "xx:qq:zz:00:00:00"
>>> ether.display()
```

```
###[ Ethernet ]###
WARNING: Mac address to reach destination not found. Using broadcast.
  dst= ff:ff:ff:ff:ff:ff
  src= xx:qq:zz:00:00:00
  type= 0x9000
>>> send(ether)
WARNING: Mac address to reach destination not found. Using broadcast.
WARNING: Mac address to reach destination not found. Using broadcast.
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "scapy/sendrecv.py", line 258, in send
    realtime=realtime, return_packets=return_packets)
  File "scapy/sendrecv.py", line 236, in __gen_send
    s.send(p)
  File "scapy/arch/linux.py", line 395, in send
    sx = str(ll(x))
  File "scapy/packet.py", line 272, in __str__
    return self.build()
  File "scapy/packet.py", line 349, in build
    p = self.do_build()
  File "scapy/packet.py", line 339, in do_build
    pay = self.do_build_payload()
  File "scapy/packet.py", line 331, in do_build_payload
    return self.payload.do_build()
  File "scapy/packet.py", line 336, in do_build
    pkt = self.self_build()
  File "scapy/packet.py", line 327, in self_build
    p = f.addfield(self, p, val)
  File "scapy/fields.py", line 70, in addfield
    return s+struct.pack(self.fmt, self.i2m(pkt,val))
  File "scapy/layers/l2.py", line 112, in i2m
    return MACField.i2m(self, pkt, self.i2h(pkt, x))
  File "scapy/fields.py", line 183, in i2m
    return mac2str(x)
  File "scapy/utils.py", line 298, in mac2str
    return "".join(map(lambda x: chr(int(x,16)), mac.split(":")))
  File "scapy/utils.py", line 298, in <lambda>
    return "".join(map(lambda x: chr(int(x,16)), mac.split(":")))
ValueError: invalid literal for int() with base 16: 'xx'
```

V ukázce je vidět, jak program Scapy reaguje při různých typech chyb. V případě syntaktické chyby (v tomto případě opomenutí uvozovek u zdrojové MAC adresy) se na obrazovku vypíše varování. Když se však do zdrojové MAC adresy zadá výraz, který je syntakticky správně, žádné varování se nevypíše a hodnota se do proměnné uloží bez ohledu na to, zda se jedná o nesprávnou hodnotu (v této ukázce například nebyla zadána validní hexadecimální hodnota). K chybě poté dojde až při samotném odesílání paketu, kdy se kvůli této chybě nepovede paket odeslat.

Scapy lze rovněž používat pomocí skriptů vytvořených v Pythonu.

1.4 Colasoft Packet Builder

Colasoft Packet Builder[5] je další nástroj umožňující vytváření síťových paketů. Kromě toho lze nástroj použít pro testování ochrany sítě proti útočníkům a narušitelům sítě. Colasoft Packet Builder je ale omezen pouze na operační systémy Windows. Nejnovější verzí je verze 2.0, která byla vydána 21. června 2016.

1.4.1 Instalace

Stažením z webových stránek produktu se získá jediný soubor s názvem *pkbuilder_2.0.0.212.exe*, který stačí spustit a spustí se instalace. Po úspěšné instalaci lze již program jednoduše spustit vzniklým spustitelným souborem.

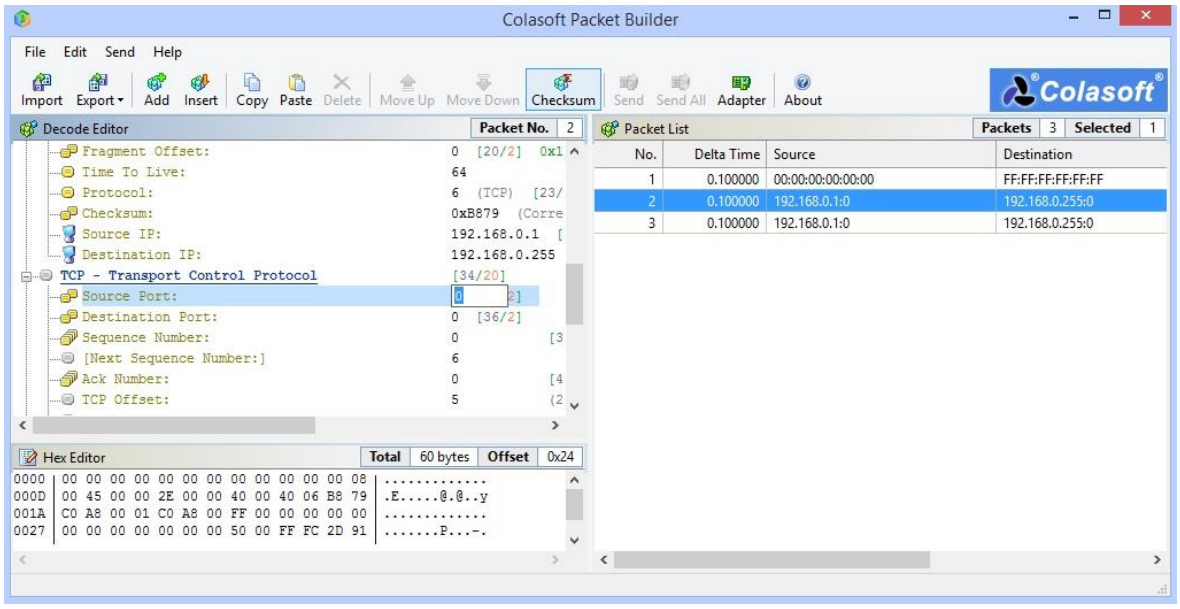
1.4.2 Funkce

Nástroj obsahuje dva editory pro úpravu paketu. Prvním je Hex Editor, ve kterém lze upravovat hexadecimální data. Druhým editorem je pak Decode Editor, který uživateli umožňuje upravovat data v libovolných polích daného paketu podle zvoleného protokolu. Oba editory jsou součástí grafického uživatelského rozhraní, a jejich užití je tedy velmi intuitivní. Uživatel si může zvolit jednu z následujících šablon: ARP paket, IP paket, TCP paket nebo UDP paket. Parametry pak lze měnit v obou zmíněných editorech (Colasoft Packet Builder, c2001-2017).

1.4.3 Ukázka použití

Po spuštění programu Colasoft Packet Builder se zobrazí okno, které je rozděleno do několika částí. Na pravé straně okna se nachází Packet list, který zobrazuje všechny vytvořené pakety. Po kliknutí na některý z těchto paketů se tento paket zobrazí v Decode Editoru i Hex Editoru v levé části obrazovky. Zde pak uživatel může pakety libovolně upravovat, jak je vidět na následujícím obrázku (Obrázek 3).

[5] Dostupné z: http://www.colasoft.com/download/products/download_packet_builder.php

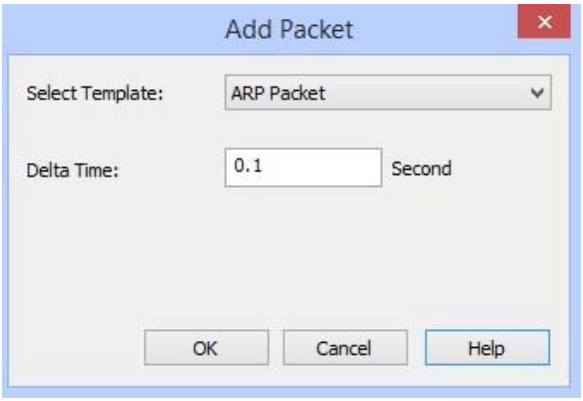


Obrázek 3 – Colasoft Packet Builder – úprava paketu

Zdroj: vlastní

V případě zadání špatné hodnoty do některého z polí se zobrazí okno s chybovou hláškou a uživatel hodnotu musí opravit.

Nové pakety se pak vytvářejí stisknutím tlačítka *Add* na horní liště programu. To vyvolá okno pro tvorbu paketu zachycené následujícím obrázkem (Obrázek 4).



Obrázek 4 – Colasoft Packet Builder – přidání paketu

Zdroj: vlastní

Zmíněné okno uživateli umožňuje nastavit šablonu paketu a Delta Time. Na výběr jsou tyto šablony: ARP paket, IP paket, TCP paket, UDP paket. Delta Time představuje časový interval v sekundách mezi novým paketem a posledním paketem v Packet Listu. Po stisknutí tlačítka *OK* se toto okno zavře a uživatelem vybraný paket se přidá do seznamu v okně Packet List.

Všechny pakety zobrazené v Packet Listu lze odeslat stisknutím tlačítka *Send All*. Pokud je cílem odeslat pouze jeden určitý paket, lze označit v seznamu paketů pouze tento konkrétní paket a následně ho odeslat tlačítkem *Send*. Před odesláním se ještě zobrazí okno, kde je možné zvolit, kolik paketů se pošle a jaký mezi nimi bude interval.

Program dále umožňuje i import a export paketů použitím souborů typu libpcap (soubory s příponou *.cap nebo *.pcap). Tyto soubory pak lze otevřít v některých dalších programech pracujících s pakety, například ve Wiresharku.

1.5 Porovnání nástrojů

Souhrn důležitých vlastností jednotlivých zkoumaných nástrojů je zobrazen v následující tabulce (Tabulka 1).

Tabulka 1 – Srovnání nástrojů Ostinato, Scapy a Colasoft Packet Builder

	Ostinato	**Scapy**	**Colasoft Packet Builder**
Platformy	Multiplatformní	Multiplatformní	Windows
GUI	Ano	Ne	Ano
Cena	Starší verze zdarma	Zdarma	Zdarma
Nabídka protokolů	Velmi rozšířená	Pouze nejrozšířenější protokoly	Pouze nejrozšířenější protokoly
Vytváření vlastních protokolů	Ne	Ano	Ne
Programovací jazyk	Python	Python	

Zdroje: (How to generate network packets – Ostinato Packet/Traffic Generator, c2017, Colasoft Packet Builder, c2001-2017, Architecture · Ostinato User Guide, c2017, Ostinato Network Traffic Generator, c2017, Scapy portability page, Technology at Ostinato Network Traffic Generator)

2 INTERNET VĚCÍ

Tato kapitola se věnuje internetu věcí (anglicky Internet of Things, zkratka IoT). Internet věcí je koncept, jehož cílem je propojování různých elektronických zařízení s internetem. To se týká například mobilních telefonů, kávovarů, televizorů, ledniček, ale i senzorů a spousty dalších zařízení. Takto propojená zařízení přes internet věcí pak mohou komunikovat nejen s lidmi, ale také vzájemně mezi sebou.

V následujících podkapitolách jsou představeny některé komunikační protokoly používané pro běh internetu věcí.

2.1 MQTT

MQTT (MQ Telemetry Transport) je nenáročný komunikační protokol, který je navržen tak, aby byl lehce implementovatelný, což ho činí vhodným pro použití v mnoha situacích. Využití si najde v omezených prostředích, jakým je například vzájemná komunikace mezi stroji v rámci internetu věcí. MQTT tedy umožňuje jednoduché a nenáročné přenosy malého množství dat prostřednictvím běžné TCP/IP internetové sítě.

2.1.1 Architektura

MQTT používá architekturu typu klient-server.

Klient je zařízení používající protokol MQTT. Klientovy úkony jsou následující:

- otevírá síťové připojení k serveru,
- vydává zprávy, o které se mohou zajímat ostatní klienti,
- odebírá zprávy, o které má zájem,
- ruší odebírání zpráv, pokud již o tyto zprávy nemá zájem,
- zavírá síťové připojení k serveru.

Server je potom program či zařízení, které slouží jako prostředník mezi klienty, kteří vydávají zprávy a klienty, kteří odebírají zprávy. Server:

- přijímá síťové připojení od klientů,
- přijímá zprávy vydané klienty,
- zpracovává klientské požadavky na odebírání a zrušení odebírání,

- danému klientovi posílá dál zprávy, které se shodují s jeho odběrem,
- zavírá síťové připojení od klienta (MQTT Version 3.1.1., 2014).

Příklad takového vztahu může být například snímač či měřící jednotka, která vydá zprávu obsahující výsledek měření. Server pak tuto zprávu odešle klientům, kteří odebírají tyto zprávy. V tomto případě může být takovým klientem například řídící jednotka, která hodnoty zpracuje či zobrazí (VOJÁČEK, 2017).

2.1.2 Struktura MQTT zpráv

MQTT paket se skládá ze vždy přítomné **hlavičky pevné délky** (16 bitů), **hlavičky proměnné délky**, která nemusí být přítomna a **datového obsahu** (payload), který rovněž nemusí být přítomen.

Hlavička pevné délky se poté skládá z následujících polí:

- **MQTT Control Packet type** (4 bity) je pole obsahující typ MQTT zprávy. Například pro *Publish Message* je stanovena hodnota 3.
- **Příznaky specifické pro každý MQTT typ** (4 bity)
- **Délka paketu** (8 bitů) určuje délku paketu v bajtech (MQTT Version 3.1.1., 2014 a Understanding the MQTT Protocol Packet Structure, 2018).

Další pole se liší podle typu MQTT zprávy. *Publish Message* má například tato další pole:

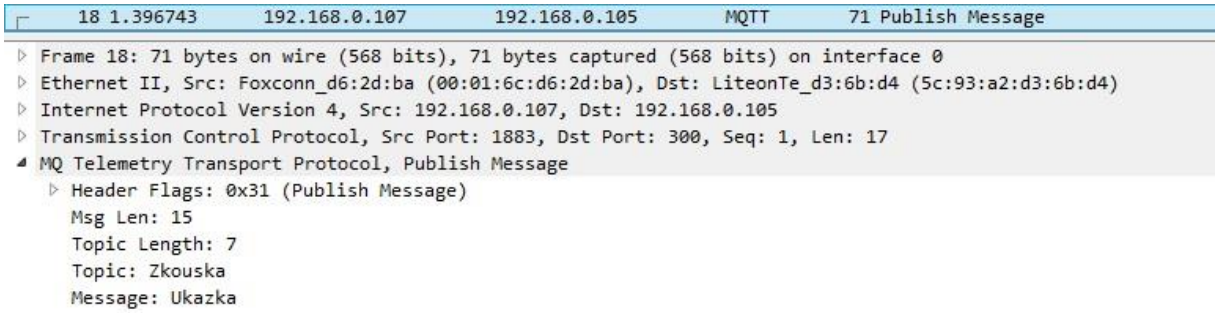
- **Délka tématu** (16 bitů) je pole, které udává délku tématu v bajtech.
- **Téma** je téma zprávy a má proměnnou délku.
- **Zpráva** je rovněž pole proměnné délky.

<table>
<tr><td>MQTT Control Packet Type (4 bity)</td><td>Příznaky specifické pro každý MQTT typ (4 bity)</td><td>Délka paketu (8 bitů)</td><td>Délka tématu (16 bitů)</td></tr>
<tr><td colspan="4">Téma</td></tr>
<tr><td colspan="4">Zpráva</td></tr>
</table>

Obrázek 5 – Pole MQTT zprávy typu Publish Message

Zdroj: vlastní

Pro ukázku následuje MQTT zpráva právě tohoto typu (Publish Message), která byla zachycena programem Wireshark.



Obrázek 6 – MQTT zpráva zachycená Wiresharkem

Zdroj: vlastní

Přenášené zprávy jsou tříděny do témat, přičemž každá zpráva patří právě do jednoho tématu. Odběratel (klient, který odebírá) pak musí předem znát jméno tématu, které chce odebírat, aby se mohl přihlásit u serveru k odběru. Odběratel pak od serveru dostává jen ta témata, ke kterým se přihlásil, přičemž nemusí znát umístění ani adresu vydavatele zpráv. Stačí znát pouze komunikační adresu serveru.

Témata jsou reprezentována řetězcem v UTF-8 kódování. Témata jsou hierarchická a oddělená lomítky, přičemž hierarchie není pevně dána, ale záleží jen na programátorovi a jeho návrhu a aplikaci.

Obsah zpráv není definovaný, a v zásadě tak obsahem mohou být data libovolného formátu, protože server tato data nijak neposuzuje, ale pouze je přeposílá. Pochopení dat záleží jen na odběrateli. Nejčastěji formáty dat však jsou JSON (JavaScript Object Notation), BSON (Binary JSON) nebo textové zprávy (VOJÁČEK, 2017).

2.2 MQTT-SN

MQTT-SN (MQTT For Sensor Networks) je otevřený a odlehčený protokol, který také funguje na principu vydavatel/odběratel. Protokol je navržen speciálně pro vzájemnou komunikaci mezi stroji a také pro mobilní aplikace (IoT Standards and Protocols).

2.3 CoAP

CoAP (Constrained Application Protocol) je protokol aplikační vrstvy, který je určený k používání v sítích či síťových zařízeních v internetu věcí, které mají omezené prostředky. CoAP je navržen tak, aby byl možný snadný překlad do HTTP (Hypertext Transfer Protocol) pro zjednodušenou integraci s webem. Zároveň tento protokol splňuje další speciální požadavky, jako například podporu multicastu, velmi nízkou režii či jednoduchost (IoT Standards and Protocols).

2.4 LLAP

LLAP (Lightweight Local Automation Protocol) protokol vytváří jednoduché krátké zprávy, které jsou posílány mezi inteligentními objekty za použití obyčejného textu. LLAP tedy může použít jakékoliv komunikační médium. Mezi jeho hlavní výhody tedy patří jeho srozumitelnost a také platí, že tento protokol bude fungovat na všem v současnosti i na všem v budoucnosti. (IoT Standards and Protocols).

3 SMART GRID

Smart Grid (do češtiny překládáno jako chytré nebo inteligentní sítě) jsou komunikační sítě, které umožňují regulovat výrobu a spotřebu elektrické energie v reálném čase. Jejich základním principem je vzájemná obousměrná komunikace mezi výrobními zdroji elektrické energie a spotřebiči nebo spotřebiteli o okamžitých možnostech výroby a spotřeby energie (Jak funguje smart grid?, c2012-2017).

3.1 Motivace

Hlavní motivací pro vývoj chytrých sítí je stárnoucí technologie současných elektrických sítí, která pomalu přestává stačit vývoji jiných technologií.

Například ve Spojených státech amerických byla elektrizační soustava vytvořena v 90. letech 19. století a technologie se v dalších desetiletích postupně zdokonalovala. Dnes se tato soustava skládá z více než 9200 jednotek generujících elektřinu, které jsou propojeny s více než 480000 kilometry elektrického vedení. Přestože jsou elektrické sítě považovány za technologický zázrak, blíží se dosažení jejich maximálního využití. Proto vznikl koncept nového typu elektrických sítí. Takového typu, který zvládne velký nárůst technologií závislých na elektřině a takového, který může zautomatizovat a zvládnout rostoucí složitost a potřebu elektřiny ve 21. století (What is the Smart Grid?).

3.2 Výhody

Chytré sítě představují bezprecedentní příležitost přesunout energetiku do nové éry spolehlivosti, dostupnosti a účinnosti, což přispěje k ekonomickému zdraví a ke zdraví životního prostředí. Výčet některých výhod spojených s chytrými sítěmi:

- účinnější přenos elektřiny,
- rychlejší obnovení elektřiny po poruchách,
- snížení cen pro provoz a správu součástí infrastruktury,
- snížení cen pro spotřebitele,
- lepší pokrytí poptávky v době špičky,
- zlepšené začlenění rozsáhlých systémů pracujících s obnovitelnou energií,

- lepší začlenění spotřebitelů vlastnících vlastní generátory energie (například solární panely),
- vylepšené zabezpečení.

V současné době mohou poruchy elektřiny (například výpadek proudu) způsobit dominový efekt – řadu selhání, která mohou ovlivnit bankovnictví, komunikační prostředky, dopravu i bezpečnost. Chytré sítě přidávají odolnost a jsou lépe připraveny vypořádat se s mimořádnými událostmi, jako jsou prudké bouře, zemětřesení či teroristické útoky. Díky obousměrné interaktivní komunikaci umožňují chytré sítě v případě výpadku některého zařízení automatické přesměrování přes jiná zařízení. Chytré sítě rovněž dokáží zaznamenat a izolovat výpadky před jejich nebezpečným rozšířením. Nové technologie dále pomáhají zajistit rychlou a strategickou obnovu po krizové situaci. Strategická obnova znamená, že se elektřina první dostane k základním složkám integrovaného záchranného systému.

Spotřebitelé díky chytrým sítím získají více informací o elektrické energii a díky domu také větší kontrolu nad ní. Mohou v reálném čase získávat údaje o jejich vlastní spotřebě elektřiny (například kdy a kolik elektřiny spotřebovávají), což jim umožní lépe hospodařit, a šetřit tak náklady za elektřinu (What is the Smart Grid?).

3.3 Technické normy

Přes 100 technických norem IEC (International Electrotechnical Commission) bylo identifikováno jako relevantní pro chytré sítě. Následuje seznam těch nejdůležitějších norem:

- IEC/TR 62357,
- IEC 61970,
- IEC 61850,
- IEC 61968,
- IEC/TS 62351,
- IEC 62056,
- IEC 61508 (Core IEC Standards, c2017).

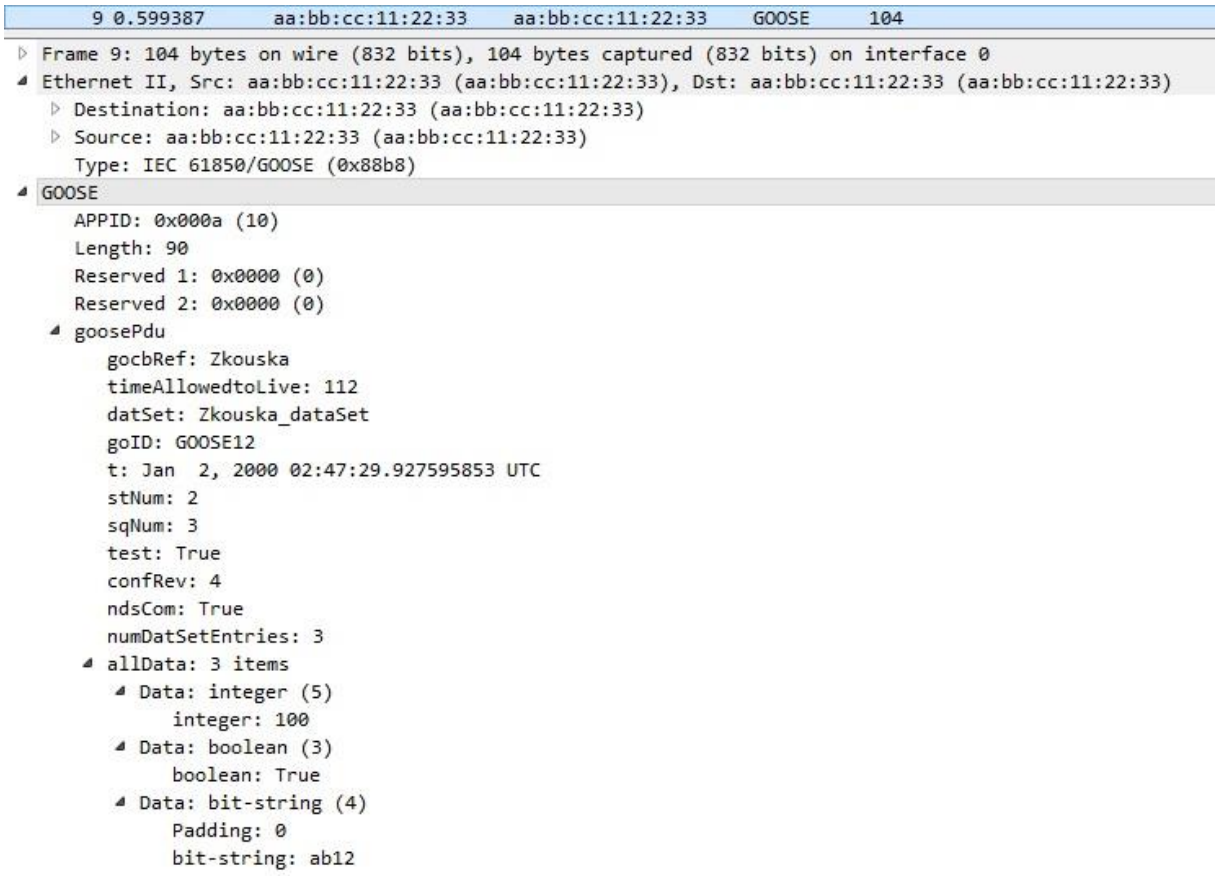
3.4 GOOSE

GOOSE (Generic Object Oriented Substation Event) je protokol normy IEC 61850. Je používaný pro rychlý přenos událostí, jako jsou příkazy, varovné signály či indikace, ve formě zpráv. GOOSE zprávu odeslanou jedním zařízením lze zachytit více zařízeními. Protokol využívá výhod Ethernetu a podporuje chování v reálném čase (ZHANG a GUNTER).

3.4.1 Struktura GOOSE zprávy

GOOSE zpráva souvisí s fyzickou, linkovou a aplikační vrstvou referenčního modelu ISO/OSI. Struktura zprávy se skládá z části, která je pevně stanovená, co se týče délky i obsahu a dále z části, která je proměnlivá ve smyslu délky i obsahu. To znamená, že uživatel definuje, jaká data budou přenášena.

Na následujícím obrázku (Obrázek 7) lze prozkoumat strukturu GOOSE zprávy zachycené programem Wireshark.



Obrázek 7 – GOOSE zpráva zachycená Wiresharkem

Zdroj: vlastní

GOOSE zpráva je zapouzdřena v ethernetovém rámci. V ethernetové hlavičce proto musí být nastaveno pole *Typ* na hodnotu 0x88b8, což označuje právě GOOSE protokol.

Protokol GOOSE obsahuje tato pole s pevně stanovenou délkou:

- **Application identifier** (16 bitů) identifikuje spojení s aplikací.
- **Length** (16 bitů) označuje velikost GOOSE zprávy.
- **Reserved 1** (16 bitů) je nastaveno na hodnotu nula a je rezervováno pro budoucí použití.
- **Reserved 2** (16 bitů) je nastaveno na hodnotu nula a je rezervováno pro budoucí použití.

Dále následují pole, která mají délku závislou na délce hodnoty daného pole v bajtech. Všechna tato pole poté mají formát, ve kterém první bajt tvoří tzv. TAG, který jednoznačně identifikuje dané pole a druhý bajt je tvořen velikostí daného pole v bajtech, aby se vyznačila hranice mezi koncem aktuálního pole a začátkem toho následujícího. Poté následují samotná data daného

pole. V tomto formátu musejí být následující pole: **GOOSE Control block reference, TimeAllowedtoLive, Data set, GOOSE ID, Time, Status number, Sequence number, Test bit, Configuration revision, Needs commissioning a Number of data set entries.**

Poslední část GOOSE zprávy pak tvoří **uživatelsky definovaná data**. Do protokolu lze díky tomu přidat libovolné množství dalších polí. Data jsou ale omezena na datové typy boolean, celé číslo a textový řetězec. O jaký datový typ se jedná lze rozpoznat díky hodnotě TAG (například pro boolean je to hodnota 0x83). Poté následuje opět velikost samotné hodnoty v bajtech pro rozpoznání konce daného pole a na konec samotná data (KRIGER, BEHARDIEN a RETONDA-MODIYA, 2013).

Následuje grafické znázornění struktury GOOSE zprávy.

<table>
<tr><td colspan="2">Application identifier (16 bitů)</td><td colspan="2">Length (16 bitů)</td></tr>
<tr><td colspan="2">Reserved 1 (16 bitů)</td><td colspan="2">Reserved 2 (16 bitů)</td></tr>
<tr><td>GOOSE Control Block Reference</td><td>TimeAllowedtoLive</td><td>Data set</td><td>GOOSE ID</td></tr>
<tr><td>Time</td><td>Status number</td><td>Sequence number</td><td>Test bit</td></tr>
<tr><td>Configuration revision</td><td>Needs commissioning</td><td>Number of data set entries</td><td>User-defined data</td></tr>
</table>

Obrázek 8 – Pole GOOSE zprávy

Zdroj: vlastní

4 GENERÁTOR SÍŤOVÉHO PROVOZU PRO TESTOVÁNÍ OPENFLOW FUNKCIONALIT

4.1 Popis aplikace

4.1.1 Charakteristika aplikace

Generátor síťového provozu pro testování OpenFlow funkcionalit je multiplatformní grafický nástroj sloužící ke generování a zachytávání síťového provozu. Nástroj funguje na architektuře typu klient-server, a proto je rozdělen na dvě aplikace. Jedna aplikace (generátor, klient) slouží k vytváření vlastních paketů a jejich následnému odesílání, druhá aplikace poté funguje jako server a slouží k zachytávání komunikace. Obě části mají grafické uživatelské rozhraní sloužící k intuitivnímu ovládání aplikace.

Klient (generátor) obsahuje předpřipravené šablony pro generování následujících IoT protokolů: GOOSE, MQTT. Další podporované protokoly: Ethernet, IPv4, IPv6, UDP, TCP, ICMP, ICMPv6. U všech zmíněných protokolů lze dynamicky nastavit všechna jejich pole, přičemž je kontrolována validita zadávaných dat.

Server kromě samotného zachytávání paketů umožňuje také nastavit filtr pro zachytávání. Lze tedy nastavit zachytávání pouze určitého protokolu, přičemž na výběr jsou protokoly podporované generátorem. Filtrovat dále lze na základě MAC adres, IPv4 adres, IPv6 adres či portů.

4.1.2 Využité nástroje

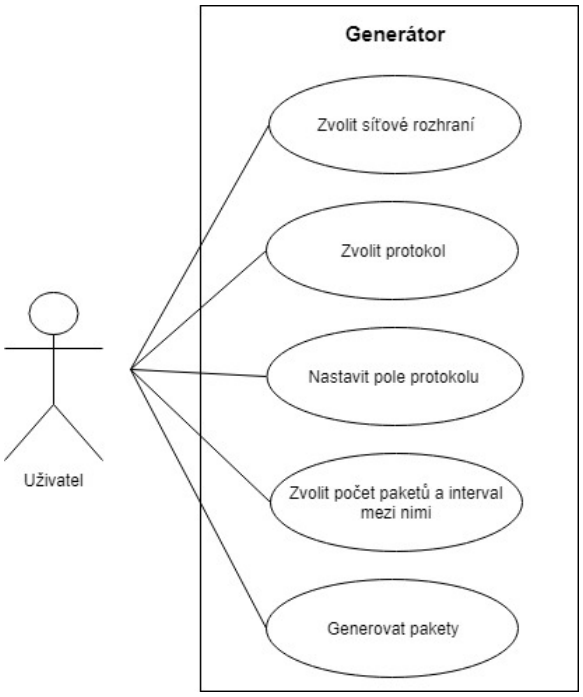
Aplikace je napsána v programovacím jazyce Java. Java je moderní objektově orientovaný programovací jazyk. Vyvinula ho firma Sun Microsystems a přestavila ho 23. května 1995. Pro tuto aplikaci byl vybrán hlavně kvůli tomu, že se jedná o multiplatformní programovací jazyk, který je zároveň snadno přenositelný. Mezi další vhodné vlastnosti patří generační správa paměti či podpora tvorby vícevláknových aplikací (Java (programovací jazyk), 2017).

Kvůli grafickému uživatelskému rozhraní je pak použita platforma JavaFX, která nahrazuje zastaralý Swing jako nástroj pro tvorbu grafického uživatelského rozhraní. Stejně jako Java, také JavaFX se řídí heslem „Write once, run anywhere“, což znamená, že aplikace naprogramované na platformě JavaFX mohou být spuštěny na kterémkoliv prostředí, které obsahuje Java Virtual Machine (JavaFX, 2017).

Základem aplikace je knihovna Pcap4J[6] verze 1.7.3, což je knihovna pro Javu, která slouží k zachytávání, vytváření a posílání paketů. Pcap4J zapouzdřuje jiné knihovny pro odchytávání síťové komunikace. Jmenovitě knihovnu libpcap[7] pro unixové systémy a WinPcap[8] pro operační systémy Windows. Pro správný běh aplikací je tedy nutné, aby na daném operačním systému byla nainstalována odpovídající knihovna.

4.1.3 Use case diagramy

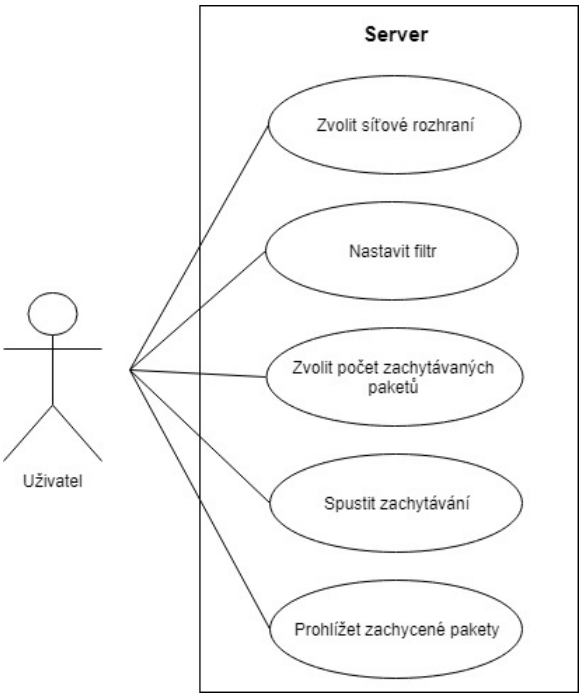
Následující obrázky zobrazují use case diagramy generátoru (klienta) a serveru.



Obrázek 9 – Use case diagram generátoru

Zdroj: vlastní

[6] Dostupné z: https://github.com/kaitoy/pcap4j#download
[7] Dostupné z: http://www.tcpdump.org/#latest-releases
[8] Dostupné z: https://www.winpcap.org/install/default.htm



Obrázek 10 – Use case diagram serveru

Zdroj: vlastní

4.2 Podporované protokoly

Tato podkapitola obsahuje výčet podporovaných protokolů formou tabulky a poté postupně popisuje dané protokoly. Protokoly GOOSE a MQTT byly popsány v kapitolách Internet věcí a Smart Grid.

Následuje tabulka (Tabulka 2) aplikací podporovaných protokolů:

Tabulka 2 – Podporované protokoly v aplikaci

Zkratka protokolu	**Název**	**Vrstva podle ISO/OSI modelu**
Ethernet	Ethernet	2
GOOSE	Generic Object Oriented Substation Events	2
ICMP	Internet Control Message Protocol	3
ICMPv6	ICMP version 6	3
IPv4	Internet Protocol version 4	3
IPv6	Internet Protocol version 6	3
MQTT	Message Queuing Telemetry Transport	7
TCP	Transmission Control Protocol	4
UDP	User Datagram Protocol	4

Zdroj: vlastní

4.2.1 Ethernet

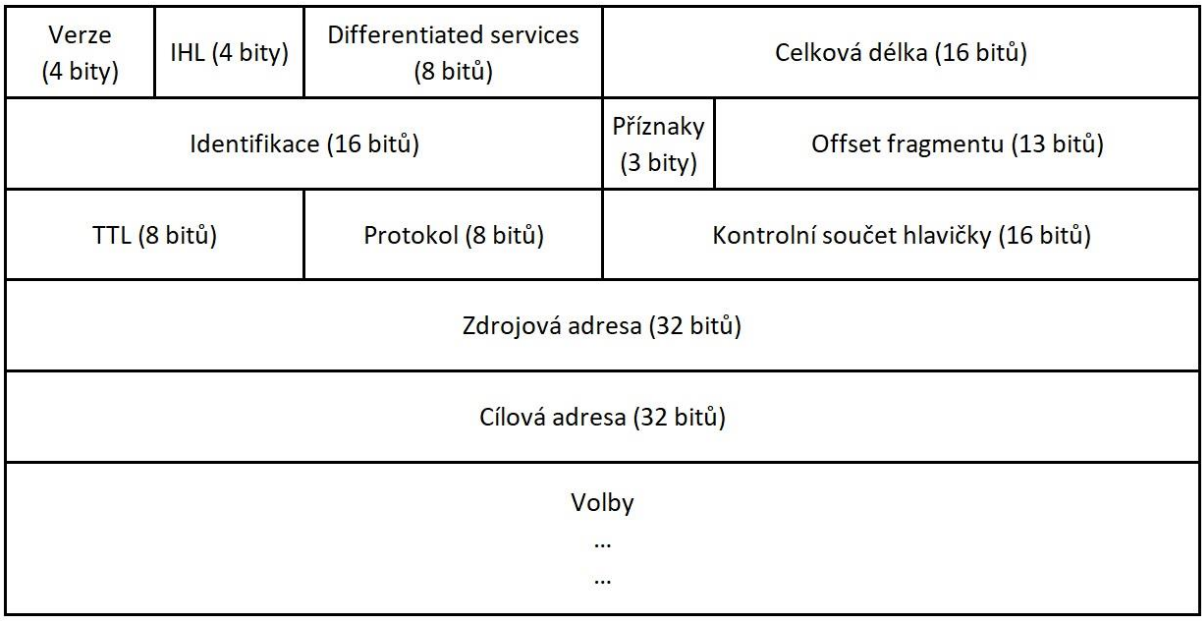
Ethernet je protokol týkající se fyzické i spojové vrstvy (1. a 2. vrstva podle ISO/OSI modelu). Tento protokol popisuje, jak mohou síťová zařízení formátovat data pro přenos k ostatním zařízením na stejném síťovém segmentu. Každý ethernetový rámec je následně zabalen do paketu, který připojuje další bajty informací (ROUSE, 2015).

Vytvořený generátor umožňuje nastavit následující pole tohoto protokolu:

- **Cílová MAC adresa** (48 bitů) je hodnota, která identifikuje příjemce.
- **Zdrojová MAC adresa** (48 bitů) je hodnota, která identifikuje odesílatele.
- **Typ** (16 bitů) indikuje, jaký protokol je zapouzdřený v datovém poli rámce (The Ethernet Frame Structure and The Ethernet Frame Fields).

4.2.2 IPv4

IPv4 je čtvrtou verzí protokolu IP a je jedním ze základních protokolů. IPv4 patří do síťové vrstvy podle ISO/OSI modelu a je používán v sítích s přepojováním paketů. Nezaručuje přitom doručení ani správné pořadí paketů a nezabraňuje jejich vícenásobnému doručení. Tyto úkony mají na starost protokoly vyšších vrstev (například TCP). IPv4 identifikuje síťová zařízení pomocí IPv4 adres a poskytuje spojení mezi nimi. IPv4 adresa je dlouhá 4 bajty. Kvůli nedostatečnému adresnímu prostoru se postupně přechází na novější protokol IPv6.



Obrázek 11 – Hlavička IPv4 protokolu

Zdroj: vlastní

IPv4 protokol obsahuje následující pole:

- **Verze** (4 bity) určuje verzi protokolu a pro protokol IPv4 je pole nastaveno na hodnotu 4.
- **IHL** (Internet Header Length, 4 bity) uchovává informaci o délce hlavičky.
- **Differentiated services** (8 bitů) je pole určující prioritu daného paketu.
- **Celková délka** (16 bitů) je délka datagramu v bajtech.
- **Identifikace** (16 bitů) je pole, které nachází uplatnění při fragmentaci datagramu, kdy se na základě tohoto pole identifikují fragmenty patřící k sobě.
- **Příznaky** (3 bity) slouží pro řízení fragmentace. První bit musí být vždy nulový, druhý bit zakazuje tento datagram fragmentovat, a třetí bit indikuje, jestli následují další fragmenty.
- **Offset fragmentu** (13 bitů) udává, na jaké pozici v původním datagramu začíná tento fragment.
- **TTL** (Time To Live, 8 bitů) představuje ochranu proti zacyklení. Když směrovač přijme datagram, sníží hodnotu tohoto pole o 1. V případě, že hodnota klesne na nulu, datagram je zahozen.

- **Protokol** (8 bitů) určuje, kterému protokolu vyšší vrstvy se mají data předat.
- **Kontrolní součet hlavičky** (16 bitů) je pole používané pro kontrolu chyb. Router spočítá kontrolní součet a pokud nesouhlasí s hodnotou tohoto pole, je datagram zahozen.
- **Zdrojová adresa** (32 bitů) je IPv4 adresa síťového rozhraní, které datagram odeslalo.
- **Cílová adresa** (32 bitů) je IPv4 adresa síťového rozhraní, kterému je datagram určen.
- **Volby** je pole obsahující různé rozšiřující informace či požadavky (IPv4, 2017).

4.2.3 IPv6

IPv6 je nejnovější verzí protokolu IP, což je protokol poskytující identifikaci pro zařízení připojená k počítačové síti. Tento protokol rovněž směruje dopravu po internetu a také patří do síťové vrstvy. IPv6 vznikl za účelem nahradit protokol IPv4, který se začal potýkat s problémem vyčerpání IPv4 adres. IPv6 adresy jsou dlouhé 128 bitů, což dává k dispozici dostatečně velký adresní prostor, který obsahuje 2^{128} adres. Adresní prostor u IPv6 je zhruba 10^{28}krát větší než adresní prostor u IPv4 (IPv6, 2017).

Obrázek 12 – Hlavička IPv6 protokolu

Zdroj: vlastní

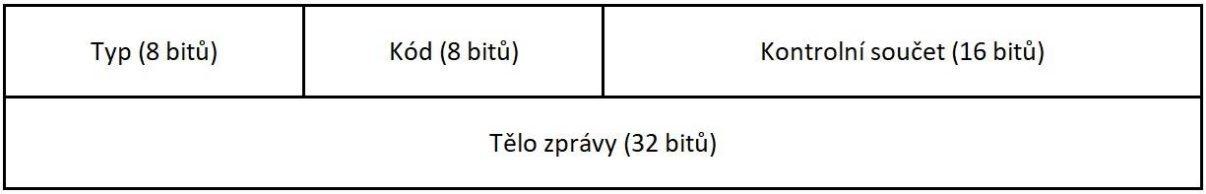
IPv6 protokol obsahuje následující pole:

- **Verze** (4 bity) určuje verzi protokolu a pro protokol IPv6 je pole nastaveno na hodnotu 6.

- **Třída provozu** (8 bitů) je pole určující prioritu daného paketu.
- **Značka toku** (20 bitů) slouží jako informace pro směrovače a přepínače, že by pakety se stejnou hodnotou tohoto pole měly zůstat na stejné cestě, aby nedošlo ke změně jejich pořadí.
- **Délka dat** (16 bitů) je délka těla paketu v bajtech.
- **Další hlavička** (8 bitů) určuje další vnořený protokol.
- **Zdrojová adresa** (128 bitů) je IPv6 adresa síťového rozhraní, které datagram odeslalo.
- **Cílová adresa** (128 bitů) je IPv6 adresa síťového rozhraní, kterému je datagram určen.
- **Hop limit** (8 bity) číselně definuje počet povolených přechodů síťovými prvky. Každý přechod znamená snížení čísla o 1 (IPv6 packet, 2018).

4.2.4 ICMP

ICMP je síťový protokol, který přenáší informace o stavu samotné sítě. ICMP zprávy jsou používány pro diagnostiku a kontrolu. Dále mohou být generovány jako reakce na nějakou chybu v operacích IP protokolu.



Obrázek 13 – Hlavička ICMP protokolu

Zdroj: vlastní

Hlavička ICMP protokolu obsahuje tato pole:

- **Typ** (8 bitů)
- **Kód** (8 bitů)
- **Kontrolní součet** (16 bitů) slouží ke kontrole integrity dat.
- **Tělo zprávy** (32 bitů) se liší na základě typu a kódu ICMP zprávy

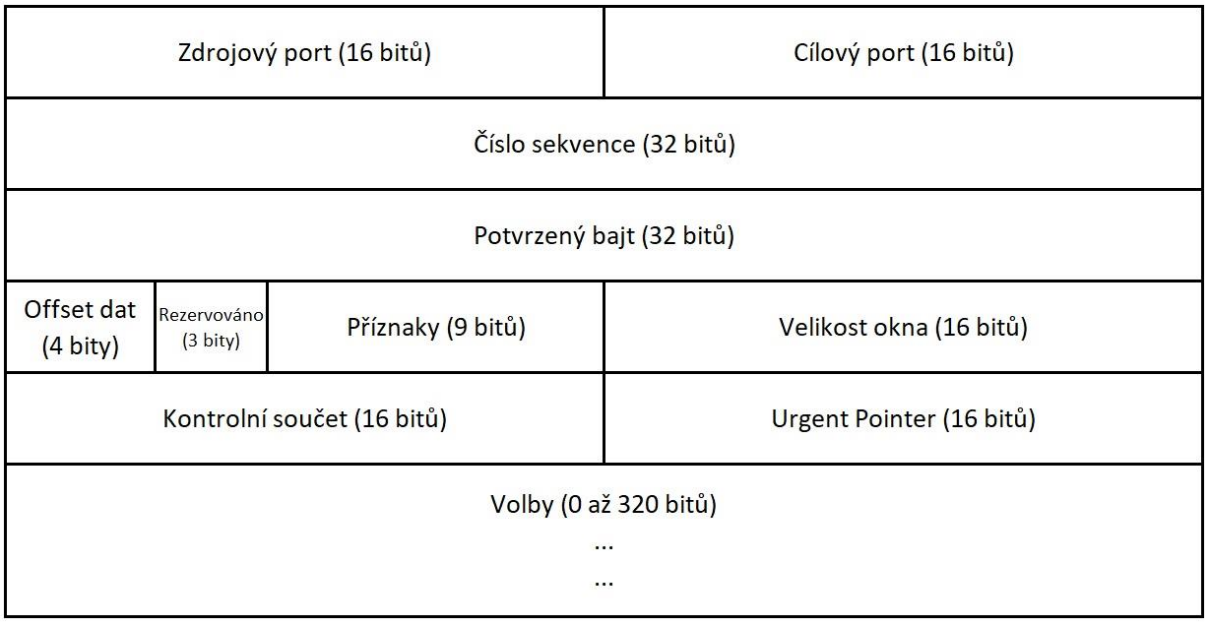
Mezi nejdůležitější ICMP zprávy patří *Echo Request* a *Echo Reply*. *Echo Request* je zpráva typu 8 a *Echo Reply* typu 0. Právě tyto dva typy zpráv využívá velmi rozšířený nástroj ping, který tyto zprávy používá k prověření spojení mezi dvěma síťovými rozhraními (Internet Constrol Message Protocol, 2018).

4.2.5 ICMPv6

ICMPv6 je nová verze protokolu ICMP používaná s protokolem IPv6. Struktura hlavičky se oproti ICMP nezměnila, skládá se tedy rovněž z typu, kódu, kontrolního součtu a těla zprávy. Do protokolu však byly přidány nové typy zpráv a bylo změněno jejich pořadí. Například *Echo Request* je zde zpráva typu 128 a *Echo Reply* typu 129 (Internet Constrol Message Protocol version 6, 2018).

4.2.6 TCP

TCP je protokol transportní vrstvy jedná se o jeden z hlavních protokolů ze sady protokolů TCP/IP, což se ostatně odráží i v názvu TCP/IP. Díky TCP mezi sebou zařízení připojená k síti vytvářejí spojení, přes které si poté mohou obousměrně posílat data. TCP zajišťuje spolehlivé doručování ve správném pořadí a s kontrolou chyb. Na TCP se spoléhají hlavní internetové aplikace a služby, jako například WWW (World Wide Web), e-mail, vzdálená správa či přenos souborů (Transmission Control Protocol, 2017).



Obrázek 14 – Hlavička TCP protokolu

Zdroj: vlastní

TCP obsahuje tato pole:

- **Zdrojový port** (16 bitů) identifikuje zdrojový port.
- **Cílový port** (16 bitů) identifikuje cílový port.
- **Číslo sekvence** (32 bitů) slouží k označení pořadí zpráv.
- **Potvrzený bajt** (32 bitů) značí číslo sekvence zprávy, která byla přijata.
- **Offset dat** (4 bity) určuje velikost hlavičky TCP protokolu.
- **Rezervováno** (3 bity) je pole rezervované pro případné budoucí využití.
- **Příznaky** (9 bitů). Seznam všech devíti příznaků: NS, CWR, ECE, URG, ACK, PSH, RST, SYN a FIN.
- **Velikost okna** (16 bitů) určuje kolik bajtů je odesílatel momentálně ochoten přijmout.
- **Kontrolní součet** (16 bitů) slouží k identifikaci poškozených zpráv.
- **Urgent Pointer** (16 bitů) je pole označující naléhavou část dat.
- **Volby** (0 až 320 bitů) je pole obsahující různé rozšiřující informace (MITCHELL, 2017).

4.2.7 UDP

UDP je dalším ze základních členů sady protokolů TCP/IP a stejně jako TCP patří do transportní vrstvy. U UDP není vyžadována předchozí komunikace pro vytvoření nového komunikačního kanálu. Není zde ale žádná záruka doručení, záruka zachování pořadí či ochrana před vícenásobným doručením. UDP je tak protokol vhodný pro případy, kdy není kontrola a oprava chyb potřeba nebo kdy tyto úkony provádí aplikace používající UDP. Výhodou UDP je, že snižuje režii, a využijí ho tak hlavně aplikace, u kterých se počítá se ztrátami datagramů a je vhodnější ztrácet pakety než ztrácet čas jejich opětovným posíláním. UDP tak využívají například DNS (Domain Name System), streamování médií či online hry.

Zdrojový port (16 bitů)	Cílový port (16 bitů)
Délka (16 bitů)	Kontrolní součet (16 bitů)

Obrázek 15 – Hlavička UDP protokolu

Zdroj: vlastní

Protokol UDP obsahuje tato pole:

- **Zdrojový port** (16 bitů) identifikuje zdrojový port.
- **Cílový port** (16 bitů) identifikuje cílový port.
- **Délka** (16 bitů) je pole reprezentující délku UDP hlavička a UDP dat v bajtech.
- **Kontrolní součet** (16 bitů) slouží ke kontrole chyb (User Datagram Protocol, 2017).

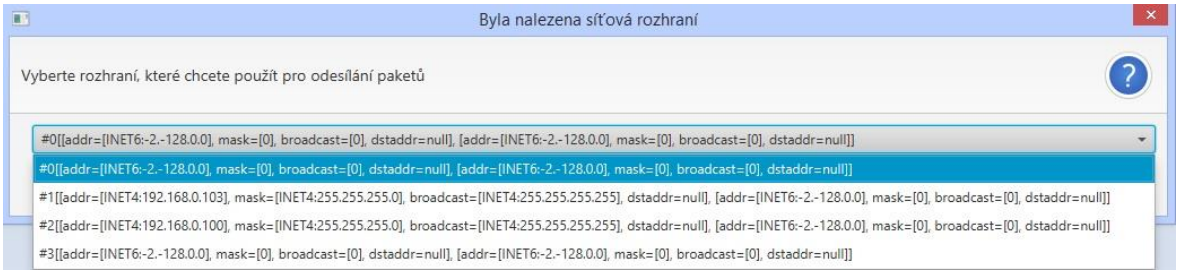
4.3 Generátor (klient)

Na úvod je nutno dodat, že aplikace generátoru musí být spuštěna s právy správce, protože aplikace potřebuje práva pro čtení síťových zařízení. Například na operačním systému Windows 8 je tak například nutné spustit příkazový řádek jako správce a samotný program poté spustit následujícím příkazem:

```
java -jar nazev_programu.jar
```

Úvodní okno aplikace nabízí možnost stisknutí tlačítka *Generovat pakety*. Před samotným generováním je však ještě nutno zvolit síťové rozhraní, které bude použito k odesílání paketů.

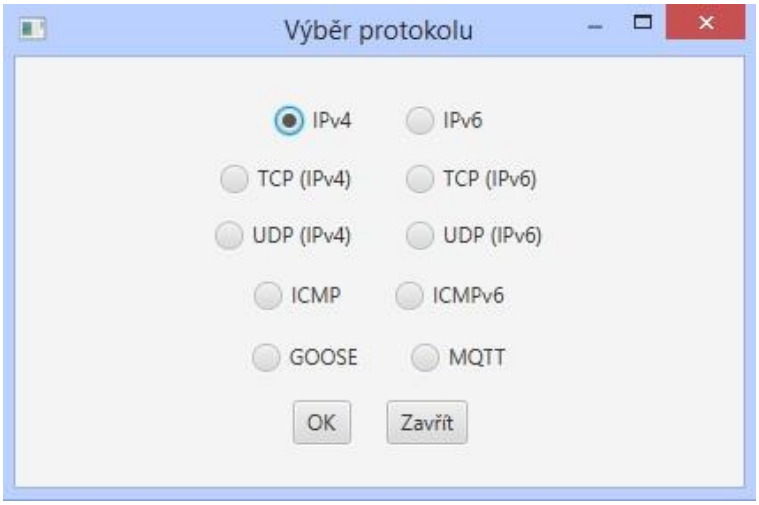
Po stisknutí tlačítka *Generovat pakety* se proto objeví nabídka dostupných síťových rozhraní na daném stroji, ze které se musí zvolit jedna z možností (Obrázek 16).



Obrázek 16 – Generátor – volba rozhraní

Zdroj: vlastní

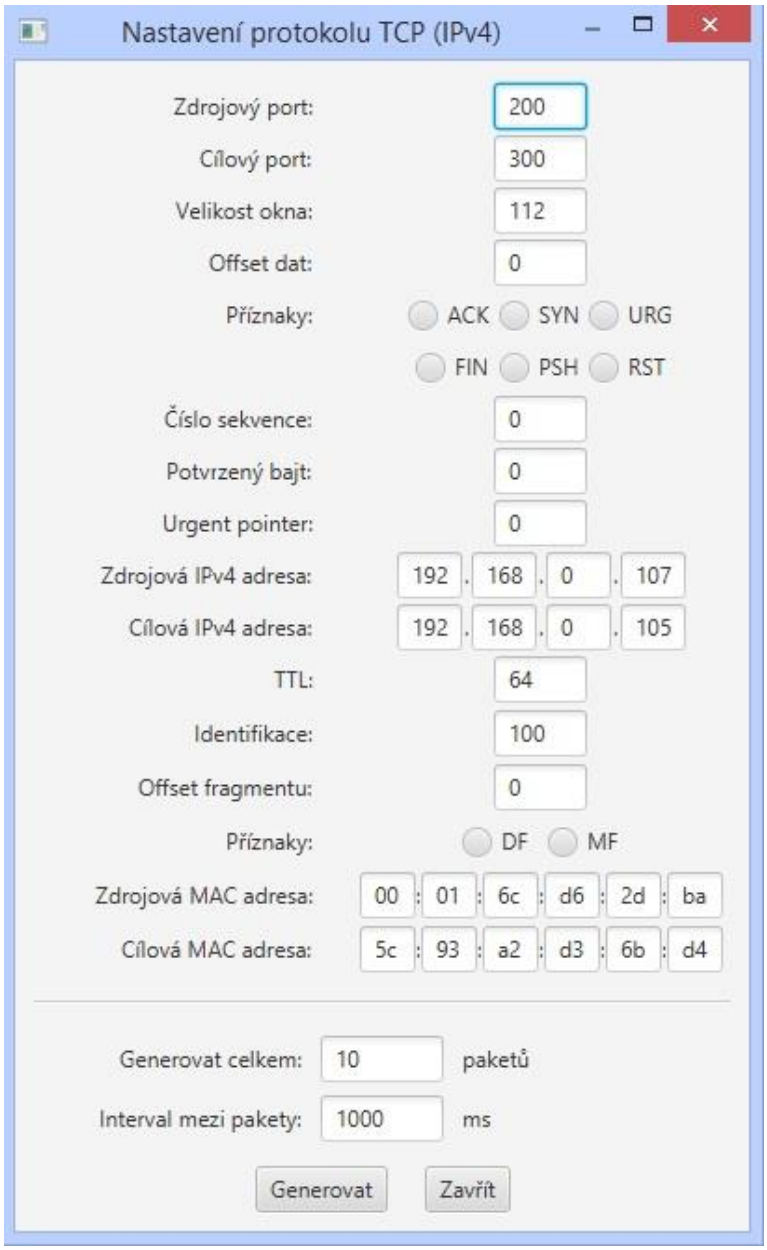
Po zvolení jednoho ze síťových rozhraní následuje volba protokolu, který bude generován. Okno pro výběr protokolu je zachyceno na následujícím obrázku (Obrázek 17).



Obrázek 17 – Generátor – výběr protokolu

Zdroj: vlastní

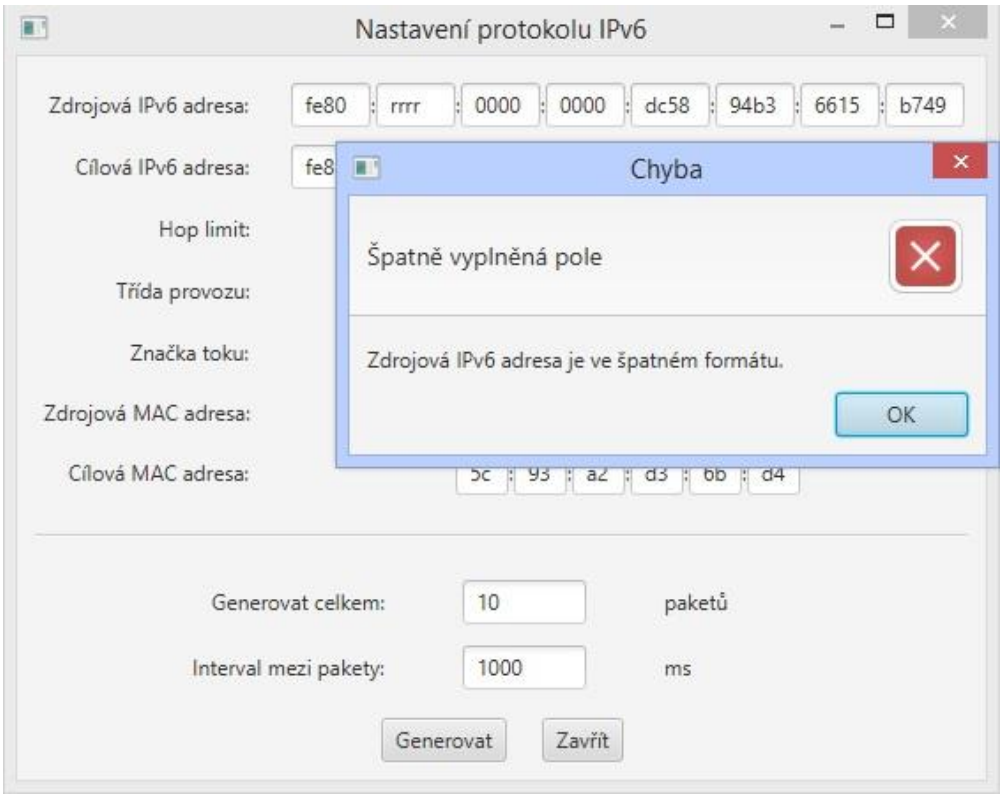
Poté se zobrazí okno, ve kterém lze nastavit všechna pole daného vybraného protokolu. Také zde lze nastavit počet paketů, který bude generován a jaký bude interval mezi jednotlivými pakety. Pakety se začnou generovat po stisknutí tlačítka *Generovat*. Na následujícím obrázku (Obrázek 18) je pro ukázku zachyceno okno pro vyplňování polí protokolu TCP.



Obrázek 18 – Generátor – nastavení polí protokolu TCP

Zdroj: vlastní

Všechna pole musí být vyplněna povolenou hodnotou a žádné pole nesmí být zanecháno nevyplněné. Při špatném vyplnění dochází po stisknutí tlačítka *Generovat* k informování uživatele chybovým oknem. Příklad takového chybového okna je zachycen na následujícím obrázku, kde byla při nastavování protokolu IPv6 špatně vyplněna zdrojová IPv6 adresa, neboť není v hexadecimálním formátu. Paket se nepodaří odeslat, dokud nedojde k nápravě.



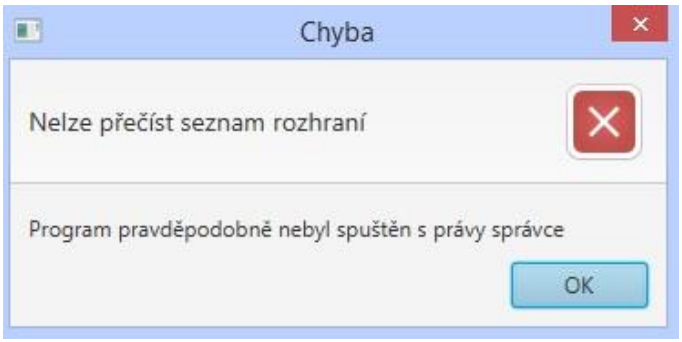
Obrázek 19 – Generátor – chybové okno

Zdroj: vlastní

Po úspěšném odeslání paketů je poté možno tyto pakety zachytit na cílovém stroji předem spuštěným serverem. Případně lze využít některý z dalších nástrojů sloužících k zachytávání komunikace, jako je například velmi rozšířený program Wireshark.

4.4 Server

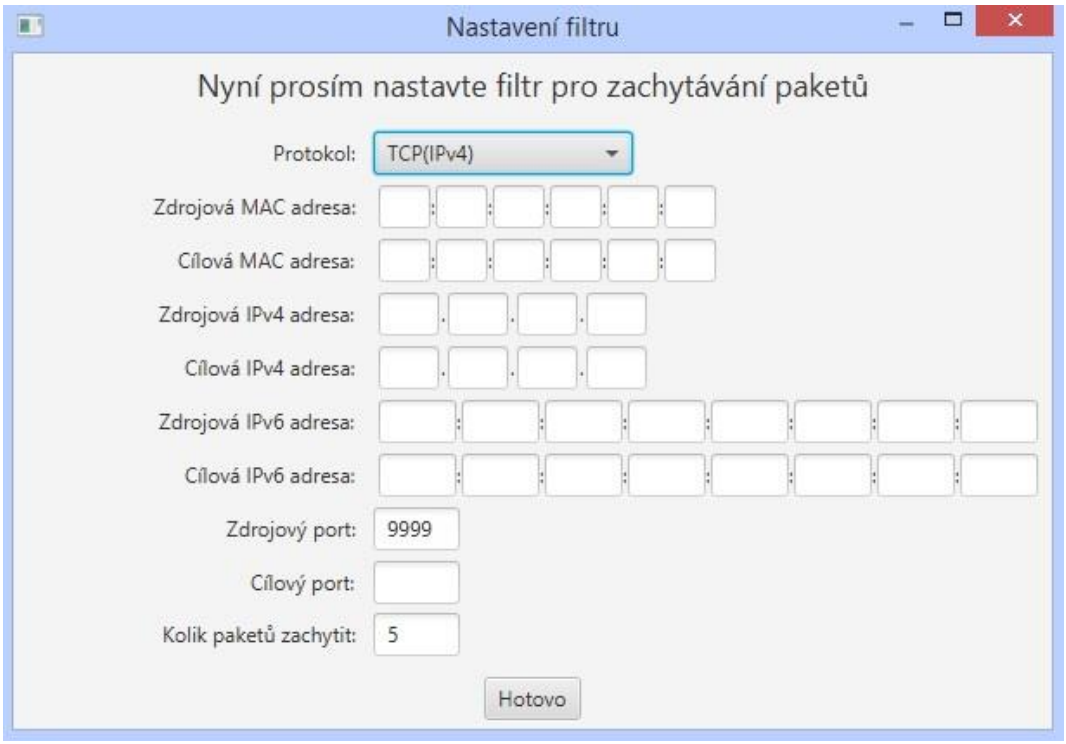
Serverová část aplikace slouží k zachytávání síťového provozu. Stejně jako u generátoru, také zde je nutno zvolit z dostupných síťových rozhraní to rozhraní, které bude naslouchat a zachytávat komunikaci. Právě kvůli síťovým rozhraním je i zde nutno spouštět aplikaci s právy správce. Jinak by nebylo možné načíst síťová rozhraní. V takovém případě se objeví následující okno (Obrázek 20), které uživatele informuje o neúspěšném načtení seznamu rozhraní a doporučí uživateli spustit aplikaci s právy správce.



Obrázek 20 – Chyba při načítání síťových rozhraní

Zdroj: vlastní

Volba rozhraní pak probíhá identicky jako u aplikace generátoru (viz Obrázek 16). Po volbě rozhraní následuje nastavení filtru. Pokud je cílem zachytávat veškerou komunikaci, filtr se nechá nevyplněn. Filtr je však velmi užitečný, je-li cílem zachytávat pakety podle určitých kritérii. Aplikace umožňuje zachytávat pakety na základě protokolu, MAC adres, IPv4 adres, IPv6 adres či portů. Uživatel si rovněž volí, kolik paketů má server zachytit. Na obrázku (Obrázek 21) je znázorněno nastavení filtru na zachycení pěti paketů s protokolem TCP a zdrojovým portem 9999.

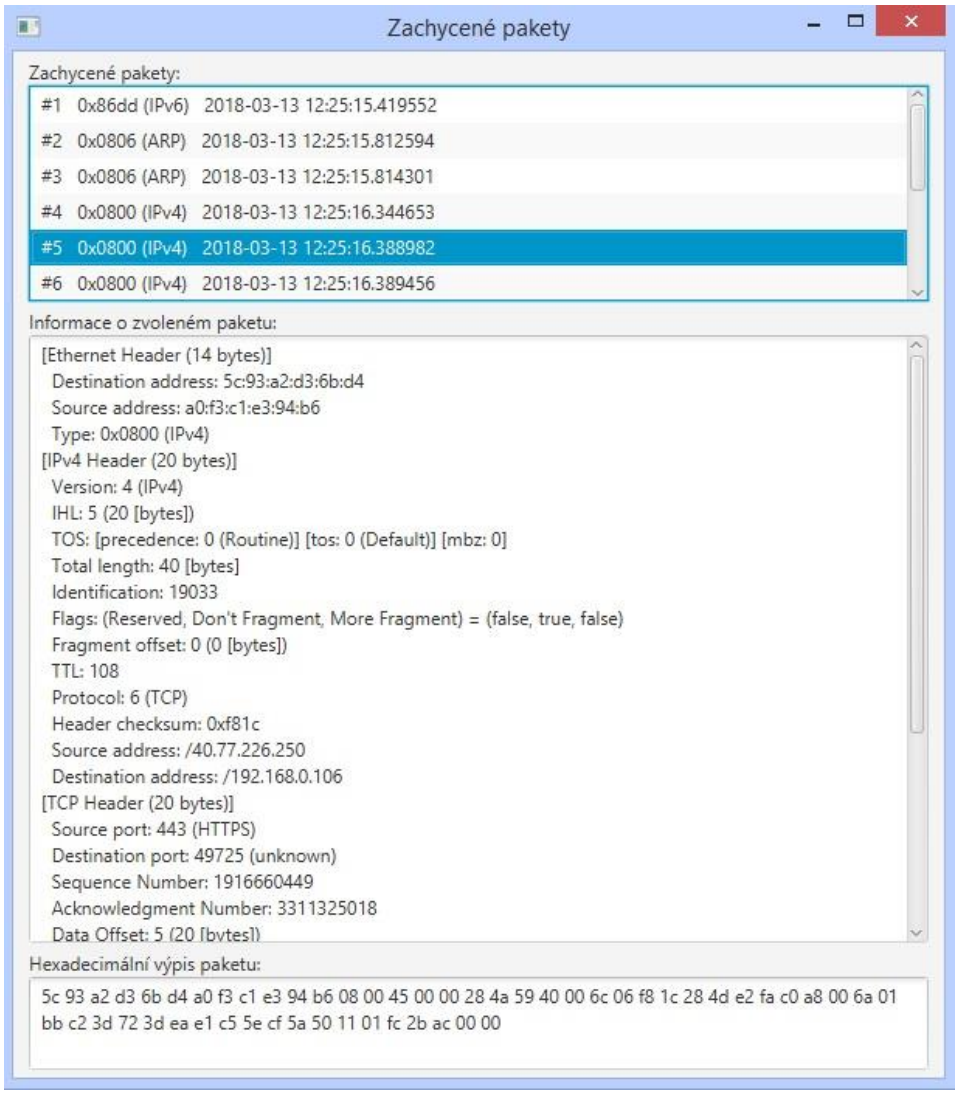


Obrázek 21 – Server – nastavení filtru

Zdroj: vlastní

I zde jsou všechny uživatelské vstupy kontrolovány a uživatel je v případě zadání nesprávného formátu informován chybovým oknem.

Po zachycení daného počtu paketů se zobrazí okno se zachycenými pakety. Příklad takového okna lze vidět na následujícím obrázku (Obrázek 22).



Obrázek 22 – Server – zachycené pakety

Zdroj: vlastní

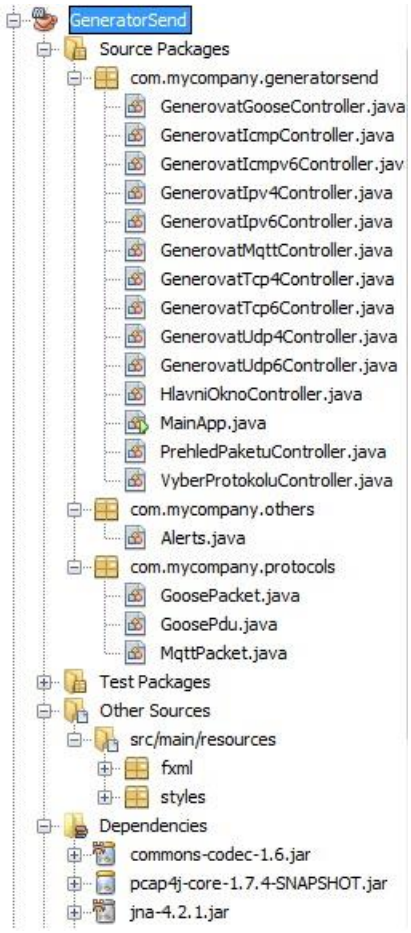
Okno je rozděleno na tři části. Nahoře je seznam zachycených paketů, přičemž jednotlivé pakety jsou rozlišeny pořadovým číslem a časem zachycení paketu. V tomto seznamu lze jednotlivé pakety označovat a pohybovat se mezi nimi pomocí počítačové myši nebo klávesnice (šipky nahoru a dolů). Druhé dvě části okna se pak mění právě v závislosti na označeném paketu. Prostřední část okna vypisuje podrobné informace o zvoleném paketu a v dolní části je poté vypsána hexadecimální podoba označeného paketu.

4.5 Technická dokumentace

V této podkapitole je technický popis aplikace. Je zde znázorněna struktura projektu a také několik důležitých tříd. Popis všech tříd obou aplikací je přiložen na CD v souboru *popis_trid.pdf*.

4.5.1 Struktura projektu

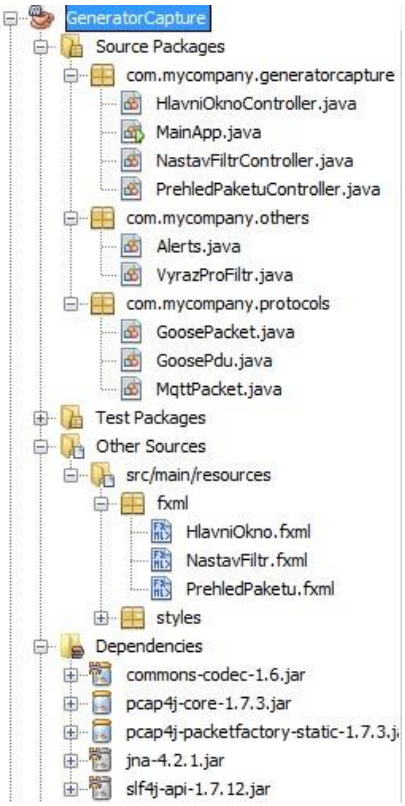
Aplikace klienta je rozdělena do několika balíčků. Balíček *com.mycompany.generatorsend* obsahuje hlavní spouštěcí soubor a kontroléry pro FXML soubory. Kontroléry jsou speciální třídy, které zajišťují spojení grafického uživatelského rozhraní s jádrem programu. Samotné FXML soubory lze nalézt ve složce *src/main/resources/fxml*. Balíček *com.mycompany.others* obsahuje třídu vytvářející informační okna pro uživatele a v balíčku *com.mycompany.protocols* se nachází třídy, které definují IoT protokoly. Na obrázku (Obrázek 23) je rovněž v části *Dependencies* vidět, že je k projektu připojena knihovna Pcap4J.



Obrázek 23 – Struktura klienta

Zdroj: vlastní

Struktura projektu serveru je podobná již popsané struktuře klientské aplikace, jak je znázorněno na následujícím obrázku (Obrázek 24).

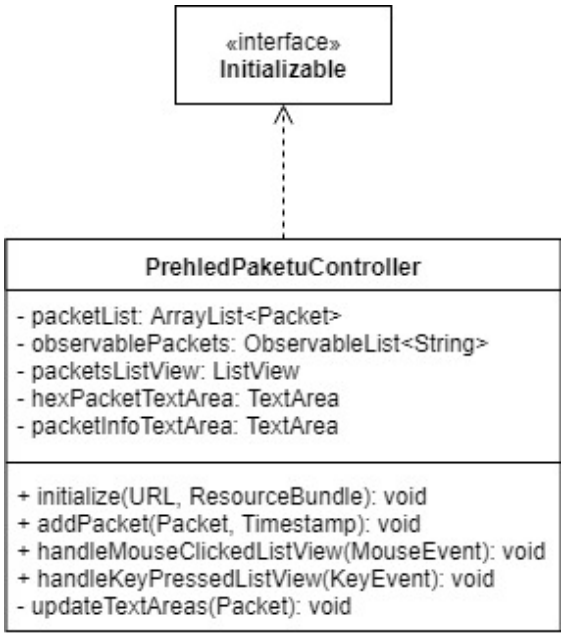


Obrázek 24 – Struktura serveru

Zdroj: vlastní

4.5.2 Třídy

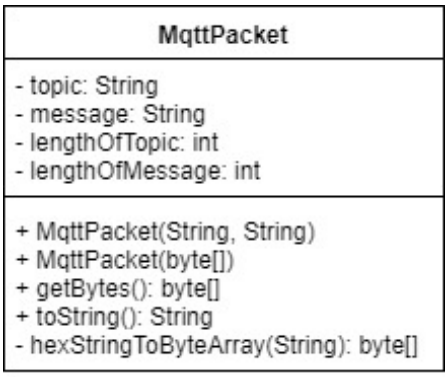
Třída PrehledPaketuController je třída obsluhující okno se zachycenými pakety. Třída implementuje rozhraní *Initializable* a mezi důležité metody třídy patří například metoda *addPacket*, která přidá zachycený paket do seznamu paketů a následně ho zobrazí nebo metoda *handleMouseClickedListView*, která zpracovává událost kliknutí na některý z paketů v seznamu paketů.



Obrázek 25 – UML diagram třídy PrehledPaketuController

Zdroj: vlastní

Třída MqttPacket je stěžejní třídou pro práci s MQTT protokolem, protože vytváří pole bajtů z člověkem čitelných hodnot a naopak. Konkrétně k tomu slouží metody *getBytes* a *toString* s pomocí privátní metody *hexStringToByteArray*.

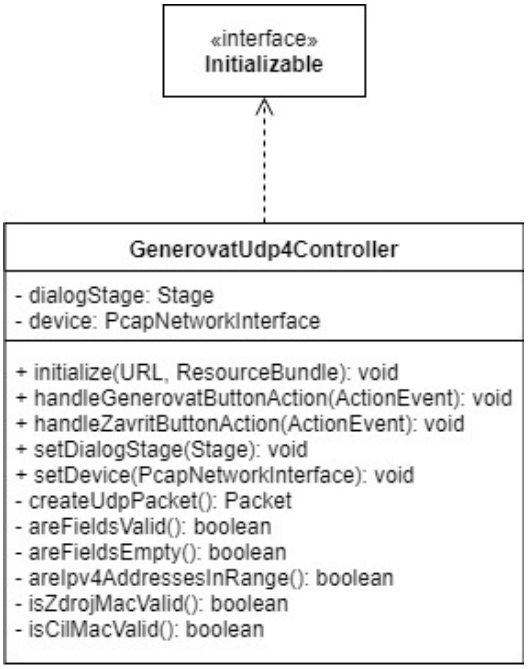


Obrázek 26 – UML diagram třídy MqttPacket

Zdroj: vlastní

Třída GenerovatUdp4Controller je další třídou obsluhující grafické uživatelské rozhraní, proto implementuje rozhraní *Initializable*. V této třídě probíhá vytváření a odesílání uživatelem definovaného UDP paketu. Jsou zde tedy kontrolovány všechny uživatelem zadané hodnoty,

k čemuž slouží metody *areFieldsValid*, *areFieldsEmpty*, *areIpv4AddressesInRange*, *isZdrojMacValid* a *isCilMacValid*. Samotné odesílání paketů pak probíhá v metodě *createUdpPacket*.



Obrázek 27 – UML diagram třídy GenerovatUdp4Controller

Zdroj: vlastní

Podobnou strukturu jako třída GenerovatUdp4Controller pak mají i třídy pro generování paketů ostatních protokolů.

ZÁVĚR

Základem této práce bylo analyzovat již existující nástroje pro generování síťového provozu. Byly vybrány tři takové nástroje, přičemž u každého z nich byl sepsán stručný úvod obsahující základní informace o daném nástroji. Byla popsána i instalace nástroje na vybraný operační systém. Nechybí ani popis stěžejních funkcí popisovaného nástroje a tato práce také názorně předvádí ukázky použití. Srovnání analyzovaných nástrojů je provedeno formou přehledné tabulky.

Cílem práce bylo také vytvořit vlastní implementaci takového nástroje. Nástroj funguje na základě architektury typu klient-server, proto je rozdělen na dvě samostatné aplikace. Byly vytvořeny multiplatformní aplikace s grafickým uživatelským rozhraním. Aplikace jsou napsány v jazyce Java, přičemž pro grafické uživatelské rozhraní je využita platforma JavaFX. První aplikací je generátor (klient), který vytváří a odesílá síťový provoz a druhou je server, který síťovou komunikaci zachytává a dovede ji navíc filtrovat podle různých kritérií. Pro aplikace byly vybrány velmi rozšířené a základní protokoly Ethernet, ICMP, ICMPv6, IPv4, IPv6, TCP a UDP. Aplikace dále podporují dva IoT protokoly, a sice protokoly GOOSE a MQTT. Pole jednotlivých protokolů lze dynamicky nastavovat díky předpřipraveným šablonám. Všechny použité protokoly včetně jejich polí jsou rovněž popsány v kapitole věnující se samotné aplikaci.

Při vytváření obou aplikací byl kladen důraz zejména na ověřování správných hodnot v jednotlivých vstupních polích a s tím spojenou dostatečnou komunikaci s uživatelem v případě nějaké chyby (například v případě špatného vstupu). Při rešerši a analýze existujících generátoru se totiž právě toto zdálo být nedostatkem zkoumaných generátorů.

Teoretická část pak představila internet věcí a některé protokoly s ním související. Jedna kapitola také informovala o rostoucí důležitosti tzv. chytrých sítí. Z vypsaných výhod jsou patrné důvody, kvůli kterým se usiluje o jejich vývoj a zdokonalování.

POUŽITÁ LITERATURA

Architecture · Ostinato User Guide. *Ostinato Network Traffic Generator* [online]. Srivats, c2017 [cit. 2017-11-09]. Dostupné z: https://userguide.ostinato.org/Architecture.html.

Colasoft Packet Builder. *Colasoft* – A professional *provider of network performance analysis, monitoring and diagnostics solution* [online]. Colasoft, c2001-2017 [cit. 2017-11-10]. Dostupné z: http://www.colasoft.com/packet_builder/.

Core IEC Standards. *Welcome to the IEC – International Electrotechnical Commission* [online]. Geneva: IEC, c2017 [cit. 2017-11-24]. Dostupné z: http://www.iec.ch/smartgrid/standards/.

How to generate network packets – Ostinato Packet/Traffic Generator. *How Does Internet Work – We know what is networking* [online]. Popeskic, c2017 [cit. 2017-11-09]. Dostupné z: https://howdoesinternetwork.com/2015/how-to-generate-network-packets.

Internet Constrol Message Protocol version 6. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-23]. Dostupné z: https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_version_6.

Internet Constrol Message Protocol. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-23]. Dostupné z: https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

IoT Standards and Protocols. *Postscapes | Navigate your Connected World* [online]. [cit. 2017-11-24]. Dostupné z: https://www.postscapes.com/internet-of-things-protocols/

IPv4. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-23]. Dostupné z: https://en.wikipedia.org/wiki/IPv4.

IPv6. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-23]. Dostupné z: https://en.wikipedia.org/wiki/IPv6.

IPv6 packet. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-27]. Dostupné z: https://en.wikipedia.org/wiki/IPv6_packet.

Jak funguje smart grid? *Novinky – elektrotechnika elektronika energetika průmyslová automatizace* [online]. MBA – Consulting Services, c2012-2017 [cit. 2017-11-22]. Dostupné z: http://www.proelektrotechniky.cz/vzdelavani/22.php.

Java (programovací jazyk). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-09]. Dostupné z: https://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk).

JavaFX. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-09]. Dostupné z: https://cs.wikipedia.org/wiki/JavaFX.

KRIGER, C., S. BEHARDIEN a J. RETONDA-MODIYA. A Detailed Analysis of the GOOSE Message Structure in an IEC 61850 Standard-Based Substation Automation System. In: *International Journal of Computers Communications & Control* [online]. Vol 8, No 5. Agora University, 2013 [cit. 2018-03-01]. ISSN 1841-9836. Dostupné z: http://univagora.ro/jour/index.php/ijccc/article/view/329/pdf_66.

MITCHELL, Bradley. TCP Headers and UDP Headers Explained. In: *Lifewire* [online]. 2017 [cit. 2017-11-22]. Dostupné z: https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970.

MQTT Version 3.1.1. *OASIS* | Adv*ancing open standards for the information society* [online]. OASIS, 2014 [cit. 2017-11-14]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

Ostinato Network Traffic Generator. *Ostinato Network Traffic Generator* [online]. Srivats, c2017 [cit. 2017-11-09]. Dostupné z: http://ostinato.org/.

ROUSE, Margaret. Ethernet. In: *Networking information, news and tips - SearchNetworking* [online]. TechTarget, 2015 [cit. 2017-11-23]. Dostupné z: http://searchnetworking.techtarget.com/definition/Ethernet.

Scapy portability page. *SecDev.org* [online]. [cit. 2017-11-09]. Dostupné z: http://www.secdev.org/projects/scapy/portability.html.

TCP/IP Reference | Nmap Network Scanning. *Nmap: the Network Mapper - Free Security Scanner* [online]. [cit. 2017-11-23]. Dostupné z: https://nmap.org/book/tcpip-ref.html.

Technology at Ostinato Network Traffic Generator. *RepIQ - The Sales Intelligence Engine* [online]. Chicago: RepIQ, c2017 [cit. 2017-11-09]. Dostupné z: https://repiq.com/company/ostinato.org/technology.

The Ethernet Frame Structure and The Ethernet Frame Fields. *Fschub.com* [online]. Rawalpindi [cit. 2017-11-23]. Dostupné z: http://fschub.com/ethernet-frame-fields-structure/.

Transmission Control Protocol. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-22]. Dostupné z: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

Ubuntu Manpage: tcpdump – dump traffic on a network. *Ubuntu Manpage: Welcome* [online]. London: Canonical, c2010 [cit. 2017-11-09]. Dostupné z: http://manpages.ubuntu.com/manpages/precise/man8/tcpdump.8.html.

Understanding the MQTT Protocol Packet Structure. *Networking, The Internet and MQTT* [online]. Steve's internet Guide, 2018 [cit. 2018-03-27]. Dostupné z: http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/.

User Datagram Protocol. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-11-23]. Dostupné z: https://en.wikipedia.org/wiki/User_Datagram_Protocol.

VOJÁČEK, Antonín. IoT MQTT prakticky v automatizaci – 1.díl – úvod. In: *Automatizace.HW.cz | Elektronika v automatizaci* [online]. Praha: HW server, 2017 [cit. 2017-11-14]. Dostupné z: https://automatizace.hw.cz/iot-mqtt-prakticky-v-automatizaci-1dil-uvod.html.

WinPcap. *WinPcap* [online]. Riverbed Technology, c2013 [cit. 2017-11-09]. Dostupné z: https://www.winpcap.org/.

What is the Smart Grid? *Home | SmartGrid.gov* [online]. U.S. Department of Energy [cit. 2017-11-22]. Dostupné z: https://www.smartgrid.gov/the_smart_grid/smart_grid.html.

ZHANG, Jianqing a Carl A. GUNTER. *IEC 61850 - Communication Networks and Systems in Substations: An Overview of Computer Science* [online]. Illinois: Illinois Security Lab [cit. 2018-03-01]. Dostupné z: http://seclab.illinois.edu/wp-content/uploads/2011/03/iec61850-intro.pdf.

PŘÍLOHY

Příloha A – Ukázky zdrojového kódu

Získání všech síťových rozhraní na daném systému a zobrazení dialogu pro výběr jednoho z těchto rozhraní uživatelem:

```
List<PcapNetworkInterface> allDevs = null;
try {
   allDevs = Pcaps.findAllDevs();
} catch (PcapNativeException ex) {
   Alerts.dialogChyba("Nelze přečíst seznam rozhraní", "Program pravděpodobně nebyl spuštěn s právy správce");
   return;
}

List<String> choices = new ArrayList<>();
int i = 0;
for (PcapNetworkInterface device : allDevs) {
   choices.add("#" + (i++) + device.getAddresses() + "\n");
}

if (choices.isEmpty()) {
   Alerts.dialogChyba("Nelze přečíst seznam rozhraní", "Program pravděpodobně nebyl spuštěn s právy správce");
   return;
}
ChoiceDialog<String> dialog = new ChoiceDialog<>(choices.get(0), choices);
dialog.setTitle("Byla nalezena síťová rozhraní");
dialog.setHeaderText("Vyberte rozhraní, které chcete použít pro přijímání paketů");

Optional<String> result = dialog.showAndWait();
if (!result.isPresent())
   return;

int volba = result.get().charAt(1) - '0';
PcapNetworkInterface device = allDevs.get(volba);
```

Metoda pro kontrolu formátu IPv6 adresy:

```
private boolean isZdrojIPv6Valid() {
   for (Node node : zdrojIPv6FlowPane.getChildren()) {
      if (node instanceof TextField) {
         String str = ((TextField) node).getText();
         if (str.length() != 4 || !str.matches("-?[0-9a-fA-F]+")) {
            return false;
         }
      }
   }
   return true;
}
```

Generování paketu:

```
try {
   PcapHandle sendHandle = device.openLive(65536, PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, 10);
```

```
  if (!areFieldsValid())
    return;
  Packet p = createGoosePacket();
  int i = 0;
  for (; i < Integer.parseInt(pocetPaketuTextField.getText()); i++) {
    try {
      sendHandle.sendPacket(p);
    } catch (NotOpenException ex) {
      Alerts.dialogChyba("Rozhraní není otevřeno", "");
    }
    try {
      Thread.sleep(Integer.parseInt(intervalTextField.getText()));
    } catch (InterruptedException e) {
      break;
    }
  }
  Alerts.dialogInfo("Info", "Bylo vygenerováno " + i + " paketů");
} catch (PcapNativeException ex) {
  Alerts.dialogChyba("Chyba rozhraní", "Program pravděpodobně nebyl spuštěn s právy správce");
}
```

Zachytávání paketů pomocí metody loop:

```
final PcapHandle handle;
try {
  handle = device.openLive(65536, PcapNetworkInterfa-ce.PromiscuousMode.PROMISCUOUS, 10);
} catch (PcapNativeException ex) {
  Alerts.dialogChyba("PcapNativeException", "");
  return;
}

PacketListener listener
          = new PacketListener() {
  int capturedPackets = 0;
  @Override
    public void gotPacket(final Packet packet) {
      controller.addPacket(packet, handle.getTimestamp());
      System.out.println(handle.getTimestamp() + ": Zachyceno "
        + (++capturedPackets) + " z "
        + Integer.parseInt(pocetPaketuTextField.getText()) + " paketů");
    }
};

handle.loop(Integer.parseInt(pocetPaketuTextField.getText()), listener);
```