VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Vojtěch Kaláb

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

TESTOVÁNÍ VÝKONNOSTI SÍTÍ S PRVKY MIKROTIK
PERFORMANCE TESTING OF NETWORKS WITH MIKROTIK COMPONENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Vojtěch Kaláb**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Pavel Mašek**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Vojtěch Kaláb ***ID:*** 146855
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Testování výkonnosti sítí s prvky Mikrotik

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a realizujte aplikaci pro zátěžové testování ethernet sítí se síťovými prvky Mikrotik. Navržená aplikace bude přenášet náhodně generovaná data z jednoho síťového koncového zařízení na druhé a bude zjišťovat vytížení jednotlivých prvků. Aplikace bude využívat Mikrotik API pro obsluhu těchto prvků sítě a pro náhodné změny dostupných rozhraní.

DOPORUČENÁ LITERATURA:

[1] BURGESS, Dennis. Learn RouterOS. Lexington, 2009, 391 s. : il. ISBN 978-0-557-09271-0.

[2] TOY, Mehmet. Networks and Services: Carrier Ethernet, PBT, MPLS-TP, and VPLS. Somerset: Wiley, 2012. ISBN 9780470391198.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 25.5.2016

Vedoucí práce: Ing. Pavel Mašek
Konzultant diplomové práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Práce je zaměřena na popsání a vytvoření aplikace pro zátěžové testování sítí skládajících se ze směrovačů od firmy Mikrotik. Aplikace je tvořena v programovacím skriptovacím jazyce Python a umožňuje zobrazení a nakonfigurování zapojených směrovacích prvků a následné zátěžové testování nakonfigurované sítě. Teoretická část se zabývá firmou Mikrotik, jejím vývojem a výrobky. Dále se zaměříme na metodiky zátěžového testování a teorii grafů. Nakonec si popíšeme samotný vývoj naší aplikace, použitý programovací jazyk Python a jeho nástroje a knihovny.

KLÍČOVÁ SLOVA

Mikrotik, MikrotikAPI, Python, zátěžové testování, IP sítě

ABSTRACT

The goal of the thesis is to develop and describe application for Mikrotik Network stress testing. The aplication is developed in high-level scripting programming language Python. The aplication allowed us to configure and display in diagram our connected routers and network stress testing. Theoretical part is focused on Mikrotik company, their development and products. Next part is about theory of network stress testing and graph theory. The last part is about aplications development, programming language Python and used tools and libraries.

KEYWORDS

Mikrotik, MikrotikAPI, Python, network stress testing, IP networks

KALÁB, V. *Testování výkonnosti sítí s prvky Mikrotik.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 82 s. Vedoucí diplomové práce Ing. Pavel Mašek.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma ,,Testování výkonnosti sítí s prvky Mikrotik" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne ....

(podpis autora)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

ÚVOD

Cílem práce je prostudovat možnosti konfigurace směrovačů od firmy Mikrotik, jejich operační systém RouterOS, metodiky zátěžového testování a zpracování výsledků měření. Na základě těchto znalostí máme vytvořit aplikaci pomocí programovacího jazyka Python, která bude schopná konfigurovat směrovače a získávat jejich dosavadní nastavení pomocí MikrotikAPI. Dále bude aplikace generovat data a zasílat je z jednoho koncového zařízení na druhé, při čemž bude měřit vytížení sítě. Aplikace bude generovat data na straně klienta a na straně serveru je zachytávat, posílat potvrzení zpět klientovi, který vyhodnotí výsledky měření. Výsledky bude aplikace zobrazovat v grafech a vygeneruje protokol o měření s naměřenými hodnotami a informacemi o stavu sítě v době testování.

V úvodu teoretické části se práce věnuje historii a vzniku firmy Mikrotik, jejich vývoji síťových prvků, jak po stránce hardwarovém, tak i softwarové. Hlavně se zaměříme na možnosti konfigurace RouterOS využívané v naší aplikaci.

Dále se zaměříme na metodiky zátěžového testování počítačových sítí. Podíváme se na organizaci IETF, její funkce a podrobněji prostudujeme RFC doporučení týkající se právě metodik zátěžového testování, kde se definují veličiny jako propustnost, zpoždění, zotavení, ztrátovost a jiné. Také se podíváme na doporučené metody měření a prezentaci výsledků, které poté aplikujeme v naší aplikaci.

Třetí část je zaměřena na teorii grafů, kdy si předvedeme co je to graf, z čeho se graf skládá, základní typy grafů a všeobecnou terminologii vztaženou k danému tématu. Zaměříme se na algoritmy pro procházení grafů a hledání nejkratší cesty v grafu.

Nakonec si popíšeme vývoj samotné aplikace a jaké jsme použily nástroje. Popíšeme si funkci jednotlivých prvků aplikace, tvorbu uživatelského rozhraní, spouštění a vyhodnocování jednotlivých testů a podobně. Nakonec si popíšeme, jak uživatel může naši aplikaci používat a co je třeba pro její spuštění a instalaci.

1. MIKROTIK

Mikrotik je jedou z mnoha společností zabývající se vývojem a výrobou prvků pro počítačové sítě, zejména směrovače a bezdrátové ISP systémy. Dnes má Mikrotik kolem 160 zaměstnanců a těší se z rostoucí celosvětové popularity, kterou získává hlavně díky nabízenému poměru cena/výkon a možnosti úpravy softwaru pro potřeby zákazníka. Cílová skupina zákazníků je od malých firem, až po středně velké poskytovatele internetu. [1,2]

Samotná firma má sídlo v lotyšské Rize a byla založena roku 1996 Johnem Trullym a Anisem Riekstinsem. Z počátku se firma zabývala vývojem softwaru na linuxové bázi pro flexibilní směrování všeho druhu v datových sítích. Výsledkem se stal velmi známý operační systém RouterOS. Kvůli zdokonalení využití vlastního operačního systému se firma rozhodla pro vytvoření vlastního hardwarového řešení a v roce 2002 představila produkt zvaný RouterBOARD. [1,2]

1.1. ROUTEROS

Jak již bylo zmíněno, RouterOS je operační systém vyvinutý firmou Mikrotik pro RouterBOARD a nebo může být nainstalován na jakémkoliv počítači s dostatečným výpočetním výkonem a poskytne mu plnou funkcionalitu směrovače se všemi nezbytnými částmi, ať už se jedná o směrování, firewall, správu šířky pásma, bezdrátový přístupový bod (pokud je podporovaný síťovou kartou), ,,hotspot" brána, VPN server a mnoho dalšího. [3]

RouterOS je samostatný operační systém založený na bázi Linuxu, kde cílem vývojářů bylo vytvořit přehledný, snadno konfigurovatelný software, který s co největší efektivitou využívá hardwarové prostředky. [3,4]

Rozšíření nebo naopak omezení jednotlivých funkcí je dána úrovní zakoupené licence. Mikrotik poskytuje jak plně funkční trial verzi na 24 hodin, tak pro registrované zájemce je dána možnost Demo verze, jejíž funkcionalita je limitována. Placené licence mají 4 úrovně. Všechny placené verze mají nelimitovaný počet rozhraní a nejsou omezené dobou trvání, ale jsou jen pro dané zařízení, tedy jsou nepřenosná. [3]

1.1.1. KONFIGURACE ROUTEROS

RouterOS umožňuje různé metody konfigurování, a to přístup přes síť pomocí protokolu telnet nebo zabezpečený a doporučovaný SSH protokol, který je bezesporu lepší volbou, přístup přes terminál pomocí sériové konzole, GUI aplikace Winbox a nebo velice podobné webové rozhraní Webfig a nakonec pomocí MikrotikAPI, což je programovatelné rozhraní pro vytvoření vlastní aplikace pro kontrolu a konfiguraci RouterOS. [4]

Nově na zařízeních s RouterOS V4 přibyl skriptovací jazyk Lua, který otevírá spoustu možností v oblasti automatizovaní konfigurace a programování. [4]

1.1.2. MIKROTIKAPI

Jedním z možných přístupů pro konfiguraci mikrotiků, které využíváme i v naší práci je MikrotikAPI. To umožňuje uživatelům vyvíjet software pro komunikaci s RouterOS a zjišťovat stav daného nastavení, popřípadě nastavení měnit. Syntaxe je velmi podobná jako u příkazové řádky. Je nutné podotknout, že MikrotikAPI lze používat pouze u zařízení s verzí RouterOS 3.x a novější. [5]

Komunikace je tvořena zasíláním a přijímáním zpráv ve formě vět, což je sekvence slov ukončená slovem s nulovou délkou. Každá věta se skládá ze slov, kde první slovo by mělo být příkazové slovo (command word).

Výpis kódu 1.1: Příklady příkazových slov pro MikrotikAPI [5]

```
/login
/user/active/listen
/interface/vlan/remove
/system/reboot
```

Každé příkazové slovo má svůj seznam atributů (attribute word), který se skládá z pěti částí

1. kódovaná délka
2. prefix se znakem "="
3. Název atributu
4. oddělující znak "="
5. hodnota atributu (může být nulová)

Výpis kódu 1.2: Příklady atributů

```
=address=10.0.0.1
=name=iu=c3Eeg
=disable-running-check=yes
```

Pokud nepotřebujeme konfigurovat, ale pouze zjistit hodnotu atributu, využijeme dotazu (Query word), který se přidává jako další parametr věty.

Výpis kódu 1.3: Příklady použití dotazů [5]

```
/interface/print
?type=ether
?type=vlan
?#|!
```

Klient na každou větu zaslanou na směrovač dostane odpověď (Replay word). Každá odpověď začíná znakem "`!`" a poslední odpověď na větu je označená "`!done`". Pokud se vyskytla chyba, tak směrovač odpovídá chybou uvedenou jako "`!trap`".

Výpis kódu 1.4: Příklad komunikace při přihlášení [5]

```
/log
 !done
=ret=ebddd18303a54111e2dea05a92ab46b4
/login
=name=admin
response=001ea726ed53ae38520c8334f82d44c9f2
 !done
```

1.1.3. WEBFIG

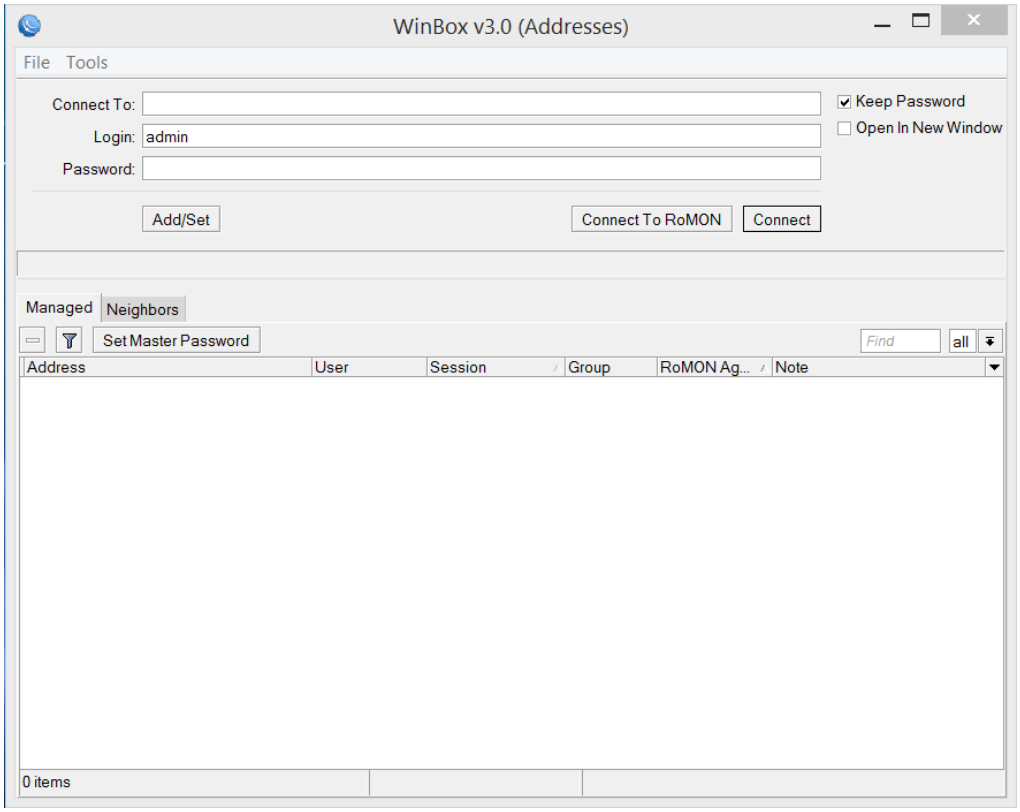
Další možností, kterou nám mikrotik nabízí pro konfiguraci RouterOS je Webfig. Již název napovídá, že se jedná o konfiguraci pomocí webového rozhraní, tudíž je k němu zapotřebí pouze konektivita k příslušnému zařízení a webový prohlížeč. Komunikace probíhá po protokolu HTTP. Jako adresu lze použít jak IPv4 tak i IPv6, kterou mikrotik podporuje. Výhoda Webfigu je, že je nezávislá na platformě. [5]

Webfig lze použít jak na prohlížení konfigurace a jejím editování, tak na monitorování pomocí logů nebo využití nástroje Graphing, který slouží právě k monitorování a kompletování různých údajů a jejich grafického zobrazení.

Webfig lze použít při hledání chyb pomocí nástrojů ping, tracerout, traffic generator (nástroj pro generování náhodného provozu) a podobně, které lze ovládat právě z Webfigu. [5]

1.1.4. WINBOX

Winbox je jednoduché GUI pro snadný přístup do RouterOS a je velmi podobným nástrojem jako Webfig, který uživateli umožňuje přístup k veškeré funkcionalitě. Winbox lze používat jak na operačním systému Windows, tak i Linuxu či MacOS.



Obr 1.1: Ukázka přihlašovacího okna Winboxu

Při zapnutí Winboxu se zobrazí úvodní okno pro přihlášení do konkrétního směrovače, který určíme buď IP adresou anebo MAC adresou. Použití IP adresy je doporučováno ve většině případů díky větší spolehlivosti. Uživatel by se měl uchýlit k MAC připojení pouze v krajních případech. Pro takové spojení je tu nástroj „neighbor discovery", který nám zobrazí seznam dostupných směrovačů. Výhodou Winboxu oproti podobným konkurenčním softwarům je, že nám nezobrazí směrovače nekompatibilní s Winboxem. Po určení směrovače pro naše spojení zadáme uživatelské jméno a heslo, klikneme na tlačítko „Connect" a již můžeme směrovač konfigurovat. [14]

Po přihlášení vidíme hlavní okno Winboxu. Na horní liště vidíme hlavní panel nástrojů s informativními poli obsahující informace jako například využití CPU a paměti.

Na levé straně je svislá lišta s nabídkou (Menu bar) s výčtem dostupných konfiguračních nástrojů. Obsah lišty závisí na nainstalovaném balíčku.

Výběrem z nabídky a následné podnabídky se nám otevře okno na pracovní ploše (Work area), jakožto centrální části okna Winboxu.[14]

Konfigurační rozhraní Winbox je navrhované pro co nejjednodušší pochopení a intuitivní ovládání, na kterém si Mikrotik velice zakládá, avšak pro nezkušeného uživatele je rozřazení nabídek zpočátku matoucí, hlavně co se týče zanořování několikaúrovňových složitějších konfigurací, kdy umístění se často liší od konkurenčních konfiguračních rozhraní. Nástroj Winbox lze zdarma stáhnout ze stránek Mikrotiku, aktuální verze je 3.0. Od verze v5RC6 Winbox podporuje konektivitu pomocí IPv6. [14]

1.2. ROUTERBOARD

Jak již bylo řečeno, Mikrotik se po vyvinutí vlastního softwaru pustil i do vývoje hardwaru a svoje produkty nazval RouterBOARD. RouterBOARD jsou tedy základní desky Mikrotik směrovačů se sloty pro doplňující moduly podle potřeb zákazníka. Díky vlastnímu specializovanému hardwaru vytvořenému na míru pro RouterOS, se staly výrobky firmy Mikrotik silnou konkurencí ve světě s vynikajícím poměrem ceny a výkonu.

Nyní se podíváme na konkrétní desky a jejich naměřený výkon v režimu mostu (Bridging) a směrování (Routing). Testy byly prováděny s třemi různými velikostmi paketů (1518 B, 512 B, 64 B) a drželi se zásad uvedených v RFC 2544. Jako maximální propustnost byla vzatá hodnota změřená při 30 sekundovém měření propustnosti s tolerancí ztrátovosti 0,1% paketu.

Model	CPU [MHz]	mode	1518 byte		512 byte		64 byte	
			kpps	Mbps	kpps	Mbps	kpps	Mbps
RB411	300	Bridging	8,1	98,6	23,5	96,3	97,3	49,8
		Routing	8,1	98,6	23,5	96,3	79,4	40,7
RB911G-2HPnD	600	Bridging	81	983,7	232	950,3	269,6	138
		Routing	81	983,7	210	860,2	226,9	116,2
RB411GL	680	Bridging	58	704,4	171,9	704,1	187,3	95,9
		Routing	59,7	725	136,1	557,5	147	75,3
RB911G-5HPacD	720	Bridging	162,4	1972,2	170,8	699,6	370,2	189,5
		Routing	162,4	1972,2	317,4	1300,1	370,2	189,5
RB450G	680	Bridging	58	704,4	171,9	704,1	187,3	95,9
		Routing	59,7	725	136,1	557,5	147	75,3
RB850Gx2	Dual Core 533	Bridging	162,4	1972,2	424,1	1737,1	446,7	228,7
		Routing	162,4	1972,2	341,1	1397,1	336,9	172,5
RB953GS-5HnT-RP	720	Bridging	162,4	1972,2	170,8	699,6	370,2	189,5
		Routing	162,4	1972,2	317,4	1300,1	370,2	189,5
RB800	800	Bridging	243,5	2957,1	330,4	1353,3	883,8	452,5
		Routing	243,5	2957,1	330	1351,7	755,4	386,8

Tabulka 1.1: Naměřené hodnoty výkonu základních desek RouterBOARD[21].

Dále se podíváme na Cloud Core sérii směrovačů, která je osazena vícejádrovými procesory Tilera. Díky vícejádrovému uskupení směrovače dosahují imponujících výkonů, jsou pro mikrotik vlajkovou lodí na trhu. Uvedené hodnoty byly měřeny za stejných podmínek jako předchozí měření pro RouterBOARD.

Model	CPU [MHz]	mode	1518 byte		512 byte		64 byte	
			kpps	Mbps	kpps	Mbps	kpps	Mbps
CCR1009-8G-1S	9 cores 1,2	Bridging	975,1	11841,6	2819,3	11547,9	14512,4	7430,3
		Routing	975,1	11841,6	2819,3	11547,9	11105	5685,8
CCR1016-12G	16 cores 1,2	Bridging	974	11828,3	2817	11538,4	15244,5	7805,2
		Routing	974	11828,3	2817	11538,4	13186,3	6751,4
CCR1036-12G-4S	36 core 1,2	Bridging	1299,1	15776,3	3759	15396,9	23808	12189,7
		Routing	1299,1	15776,3	3759	15396,9	23808	12189,7
CCR1036-8G-2S+EM	36 core 1,2	Bridging	27615,5	27615,5	6579	26947,6	39764,6	20359,5
		Routing	2274	27615,5	6579	26947,6	34086,1	17452,1
CCR1072-1G18S	72 core 1	Bridging	6505	78960,3	18790	76963,8	98873,5	50623,2
		Routing	6502	78960,3	18790	76963,8	76201	39014,9

Tabulka 1.2: Naměřené hodnoty výkonu směrovačů z řady Cloud Core Routers[21].

2. METODIKY ZÁTĚŽOVÉHO TESTOVÁNÍ SÍTÍ

Při návrhu počítačových sítí je potřeba vzít v potaz obrovské množství faktorů a při rostoucí velikosti sítě také rostou požadavky na důmyslnost návrhu. Je potřeba zajistit dostatečnou pružnost sítě, odolnost vůči zátěži a to i při částečných výpadcích, tudíž musí návrhář zajistit dostatečnou redundantnost cest a správné dimenzování prvků, ale tím narůstá cena komponentů. Možností jak takovou síť testovat je spoustu, ale jsou dané doporučení v podobě RFC dokumentů.

RFC vzniklo ještě v době, kdy se internet jmenoval ARPANET , roku 1969. Jedná se o soubor dokumentů ohledně internetu a jeho protokolů. Takovýto návrh může podat kdokoliv společnosti IETF, a ta ho prozkoumá. Pokud jej schválí, tak vytvoří RFC dokument se svým unikátním číslem, v opačném případě je návrh zamítnut. Pokud je o daný dokument zájem, pak se stává standardem internetu. Takovýto dokument se již nemění a pokud je potřeba vykonat změnu, vydá se nový RFC dokument s novým unikátním označením. [9]

2.1. ORGANIZACE IETF

IETF je mezinárodní komunita síťových návrhářů, prodejců, operátorů a vývojářů, kteří vyvíjejí a podporují internetové standardy, převážně týkající se TCP/IP sítí. Členové IETF jsou dobrovolníci sponzorováni různými organizacemi a scházejí se třikrát do roka. Jsou organizování do skupin a každá skupina dostane přidělené téma. IETF spolupracuje s dalšími organizacemi jako IANA či ISO. Skupinou IETF která se zajímá metodologií zátěžového testování prvků je skupina BMWG. [10]

2.2. RFC 1242

Jejich prvním vydaným RFC dokumentem bylo RFC 1242 s názvem *Benchmarking Terminology for Network Interconnection Devices*, které ustanovuje terminologii v testování a způsob prezentace výsledků. Návrh prezentace a terminologii se dále objevuje v následujících RFC vydané touto skupinou. Dokument byl vydaný již v roce 1991 a vzhledem k novým trendům a technologiím bylo nutné udělat změny a úpravy, tak byl dokument zmodernizován a znovu vydán jako RFC 6201 v roce 2007. [11]

2.3. RFC 2544

Další dokument vydaný touto skupinou bylo RFC 1944 nazvané *Benchmarking Methodology for Network Interconnect Devices*, které bylo pro zastaralost kompletně nahrazeno novější verzí RFC 2544. Toto RFC popisuje a definuje řadu testů, které mohou být použity k popisu výkonových charakteristik IP sítí. Dále dokument obsahuje popisy formátů pro prezentaci naměřených výsledků.

Vlastnosti a formy výsledků testů definované v tomto dokumentu by měli používat prodejci při propagaci svého výrobku, aby nedošlo k dezinformovanosti, či k šíření neúplných informací s cílem nalákat co nejvíce zákazníků. RFC 2544 používá terminologie stanovené v předchozím dokumentu RFC 1245.[12]

2.3.1. BACK-TO-BACK FRAMES

Back-to-Back je první termín, který RFC 1242 definuje a následně v RFC 2544 rozebírá doporučený postup měření a prezentaci výsledků. Jedná se o odesílání rámců o stanovené délce při maximální rychlosti s minimální možnou časovou mezerou mezi jednotlivými rámci. Důvodem k testování této vlastnosti je růst internetu a používaných rychlostí. Hlavně na

technologiích s malou MTU rozdělují tok dat do mnoho fragmentů a ztráta i jediného fragmentu může vést k chybě při defragmentaci a nutností opakování přenosu. Jednotkou je počet N-oktetových rámců ve shluku (burst). [11]

Měření by mělo být provedeno podle doporučení tak, že pošleme shluk rámců s minimálními rozestupy mezi jednotlivými rámci do DUT a počítáme rámce, které jsou přeposlány. Pokud je počet přeposlaných rámců stejný jako počet odeslaných rámců, zvětšíme délku shlukové sekvence (burst) a test opakujeme. Naopak, pokud je počet přeposlaných rámců menší než původně odeslaných, musíme shluk zmenšit a měření opět opakovat.

Výsledná hodnota back-to-back je počet rámců v nejdelším shluku, se kterým se DUT vypořádá bez ztráty rámce. Každý pokus musí trvat minimálně 2 sekundy a měl by být padesátkrát opakován.[12]

2.3.2. ZTRÁTOVOST

Dalším termínem je ztrátovost (Frame Loss Rate), která stanovuje procento rámců nepředaných síťovým prvkem při ustáleném stavu z důvodu nedostatku zdrojů. Tímto měřením můžeme ukázat výkon síťového prvku v přetíženém stavu. Jednotkou je procento rámců, které je zahozeno a výsledekem měření by měl být graf deklarovaného zatížení ku ztrátám rámců. [11]

Procedura testování by měla probíhat tak, že pošleme určitý počet rámců danou rámcovou rychlostí přes DUT a počítáme rámce, které jsou DUT doručeny. Ztrátovost se při každém opakování měření počítá podle vzorce 2.1.

$$FLR = ((input\_count\ -\ output\_count\ ) * 100)/input\_count \qquad (2.1)$$

Ve vzorci nám „FLR" představuje ztrátovost, „input_count" je počet poslaných rámců do DUT a naopak „output_count" značí počet odeslaných rámců z DUT.

První pokus by se měl přibližovat 100% maximální rámcové rychlosti při dané velikosti rámce na daném médiu. Pokus opakujeme pro 90% rámcové rychlost a poté pro 80%. Takto pokračujeme ve skocích rovných 10% až se dostaneme na dva po sobě jdoucí pokusy, kde máme úspěšně nulová ztrátovost.

Pro prezentování výsledků měření by měl být sestaven graf, kde na ose x musí být použitá rámcová rychlost uvedená v procentech z teoretické propustnosti a na ose y zase ztrátovost opět v procentech. Osa y by měla zahrnovat hodnoty od 0% do 100%. [12]

2.3.3. PROPUSTNOST

Jednou z pro nás nejdůležitějších definic je propustnost (throughput). Definice propustnosti je podle RFC 1242 nejvyšší rychlost, při které se počet testovacích rámců přenášených pomocí DUT rovná počtu rámců přijatých v DUT. Jinými slovy propustnost udává maximální počet rámců za sekundu [12].

Při zjišťování hodnoty propustnosti odešleme určitý počet rámců danou rychlostí přes DUT a následně počítáme počet rámců odeslaných z DUT. Pokud počet rámců odeslaných DUT je nižší než počet přijatých rámců, snížíme rychlost a měření opakujeme, dokud nedosáhneme stejné hodnoty.

Prezentace měření propustnosti by měla obsahovat graf s dvěma osami souřadnic x a y, kde na ose x budeme mít velikost rámců a souřadnice y obsahuje počet rámců, tudíž je to

závislost odeslaných rámců na velikosti rámců. V grafu by také měla být znázorněná křivka s maximálním možnou propustností pro dané médium [12].

Pro reklamní účely se uvádí hodnota rychlosti s minimální velikostí rámce pro dané přenosové médium. Tato hodnota musí být vyjádřena v jednotce fps (frames per second) ale měla by být i v bitech za sekundu. Při prezentování výrobku musí být zveřejněna naměřené maximální rámcové rychlosti, velikost použitých rámců i teoretický limit umožňující použité médium [12].

Význam propustnosti je zcela zjevný, vyšší propustnost nám umožňuje používat vyšší přenosovou rychlost a zrychlí tak naši komunikaci. Pro rychlé měření propustnosti můžeme použít například nástroj IPerf.

2.3.4. ZPOŽDĚNÍ

Další důležitou vlastností sítí, o kterém se zmiňuje RFC 2544 a je definované v RFC 1242, je zpoždění (latency).

Při měření zpoždění nejdříve musíme znát hodnotu propustnosti pro všechny uvedené velikosti rámců. Pošleme tok dat s určitou velikostí rámců přes DUT stanovenou rychlostí. Pro zachování přesnosti měření je nutné, aby test trval minimálně 120 sekund.

Každých 60 sekund by měl rámec obsahovat časovou značku označovanou jako „timestamp". Zaznamená se čas odeslání jako „timestamp A". Při přijmutí tohoto rámce od DUT je zaznamenána další časová značka „timestamp B". Definice zpoždění vychází z těchto časových značek podle RFC 1242, a to jako jejich rozdíl.

Formát výsledku měření musí obsahovat informaci, které zpoždění bylo měřena. Zpoždění totiž rozdělujeme na jednosměrné „one-way", kde je myšleno zpoždění od odeslání po přijetí a obousměrné „round-trip delay time" zahrnující jednosměrné zpoždění spolu se zpožděním z přijímacího zařízení zpět do odesílajícího zařízení [12].

Obousměrné zpoždění je používané častěji, neboť nám poskytne lepší přehled o skutečném prodlení komunikace v síti, kdy je zahrnut i čas potřebný ke zpracování rámců.

Pro rychlé zjištění zpoždění existuje služba Ping, která je zahrnuta do různých systémových platforem, ať už se jedná o Windows, Linux či MacOS. Takovéto měření je velmi hrubé, neboť využívá protokol ICMP, který není skutečný komunikační protokol jako TCP či UDP a nezahrnuje dobu zpracovávání toku dat jako by to bylo u normálního toku dat. Měření zpoždění je však u služby Ping pouze sekundární vlastnost, neboť slouží především jako rychlý nástroj pro vyhledávání chyb, konkrétně dostupnost cílových IP adres [13].

2.3.5. SYSTÉMOVÉ ZOTAVENÍ

Nakonec se budeme věnovat vlastnosti zotavení systému, což charakterizuje na testovaném zařízení dobu, za jakou je schopné se zotavit z přetížení.

Při měření nejprve určíme propustnost DUT pro každou délku rámce. Poté pošleme datový tok s rámcovou rychlostí okolo 110% zaznamenané propustnosti anebo teoretickou propustnost pro použité přenosové médium, kdy vybereme tu hodnotu, které je nižší. Datový tok posíláme minimálně po dobu 60 sekund. [12]

Poté zaznamenáme časové razítko značené „Timestamb A" a v tento okamžik snížíme rámcovou rychlost na 50%. Měříme časový úsek do doby, kdy je zahozen poslední rámec a tento okamžik označíme druhým časovým razítkem „Timestamb B". Čas pro zotavení systému je definován jako rozdíl časových razítek „Timestamb B" a „Timestamb A" [12].

Prezentace výsledků zotavení systému by měla být ve formě tabulky s řádky pro každou velikost testovaných rámců a sloupci pro použitou rámcovou rychlost pro testovací datový tok [12].

2.3.6. RESET

Tento termín charakterizuje rychlost s jakou se DUT zotaví po resetu. Jako reset lze použít softwarový reset, anebo hardwarový reset, kdy DUT odpojíme od napájení po dobu 10 sekund [12].

Při měření hodnoty zotavení po resetu se určí propustnost DUT s minimální délkou rámce na daném médiu použitém při testování. Dále pošleme plynulý datový tok s výše uvedenou propustností a vyvoláme reset na DUT. Tento okamžik si zaznamenáme jako „Timestamb A". Následně monitorujeme výstup, dokut DUT opět nezačne posílat rámce našeho datového toku. Tento okamžik označíme časovou značkou „Timestamb B". Výsledná velikost hodnoty resetu je opět rozdíl časových značek „Timestamb B" a „Timestamb A" [12].

3. TEORIE GRAFŮ

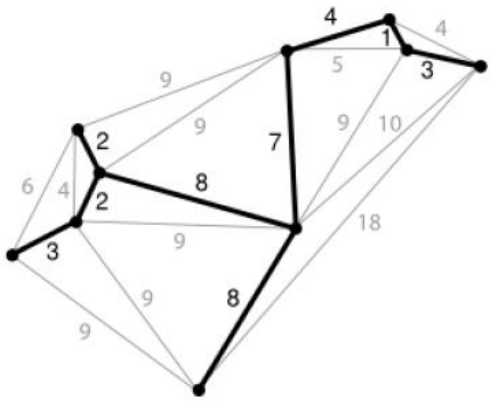
Teorie grafů je rozvinutá matematická disciplína popisující studii a využití grafů, což jsou matematické struktury pro modelování vztahů mezi objekty. Grafy jsou tvořeny vrcholy neboli uzly a hranami, které se ve vrcholech (uzlech) vzájemně spojují. Definice grafu G je uspořádaná dvojice $G = (V, E)$, kde V je konečná množina vrcholů a E je konečná množina hran [15,16].

Základní příklady využití grafů v praxi jsou komunikační modely, kdy jsou topologie sítě a její prvky znázorněny v grafu. V grafu se dají zaznamenat i elektrické obvody, modely počítačové architektury, umělá inteligence a mnoho dalšího [17].

3.1. ZÁKLADNÍ POJMY

Pro práci s grafy je nutné definovat základní pojmy pro jejich popis.

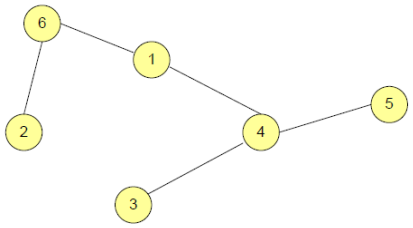
- **Stupeň vrcholu** je počet hran, které vycházejí z daného vrcholu. U orientovaných grafů je stupeň definován pomocí uspořádané dvojice, kde první číslo značí počet vstupujících do vrcholu a druhé číslo počet hran vystupujících [15].
- **Sled** je zase posloupnost vrcholů a hran začínající a končící v daném vrcholu a tyto hrany a vrcholy se mohou v rámci sledu opakovat. Délka sledu je počet hran v rámci daného sledu, a pokud jsou hrany ohodnoceny, tak je roven součtu ohodnocení hran ve sledu [15].
- **Tah** je speciální typ sledu, kde se žádná hrana neopakuje.
- **Cesta** je sled, v němž se zase neopakují žádné vrcholy, tedy každý vrchol se může objevit v cestě nejvýše jednou [15].
- **Podgraf** je graf tvořený částmi původního grafu. Může obsahovat některé, nebo všechny jeho vrcholy a některé, nebo všechny jeho hrany. Podgraf vzniká odebráním některých hran či vrcholů. Na sled, tah nebo cestu lze tudíž nahlížet také jako na podgrafy [15].
- **Průchod** je sekvence vrcholů $(v_1,v_2,...,v_L)$ [17].
- **Cyklus** je případ průchodu, kde $v_1=v_L$,tedy že první a poslední vrchol je identický [17].
- **Kostra grafu** u souvislého grafu tvoří podmnožinu hran, která tvoří strom obsahující všechny uzly a celkový počet hran nebo u váženého grafu celková váha grafu je minimální. U nespojitého grafu hledáme les minimálních koster, tedy u každé komponenty její kostru zvlášť [17].



Obr 3.1: Příklad kostry grafu [17].

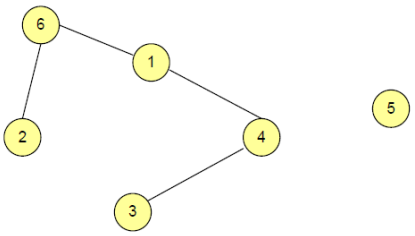
3.2. ROZDĚLENÍ GRAFŮ

Grafy lze dělit podle jejich vlastností, například podle souvislosti:
Souvislý graf je takový graf, kde mezi dvěma různými vrcholy grafu existuje cesta.



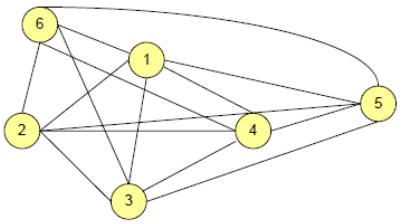
Obr 3.2: Ukázka souvislého grafu [15].

Nesouvislý graf pak znamená, že v něm existují dva různé vrcholy, mezi nimiž neexistuje žádná cesta.



Obr 3.3: Ukázka nesouvislého grafu [15].

Úplný graf je extrémní případ souvislého grafu, jehož každý vrchol je přímo spojený se všemi ostatními vrcholy.

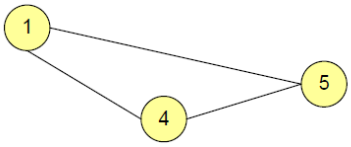


Obr 3.4: Ukázka úplného grafu [15].

Orientovaný graf má hrany začínající ve vrcholech *v* a *u*, taková hrana je značená (*u*,*v*) a je různá od (*v*,*u*). Příkladem takového grafu může být diagram znázorňující průchod počítačové sítě od klientu k serveru [15].

Neorientovaný graf je opak orientovaného grafu a je reprezentován neuspořádanou dvojící (*u*,*v*) = (*v*,*u*) [17].

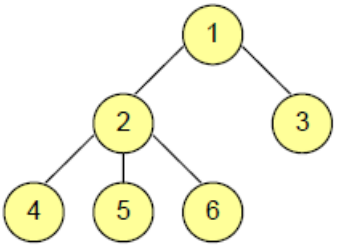
Cyklický graf je orientovaný graf obsahující alespoň jeden cyklus [15].



Obr 3.5: Ukázka cyklického grafu [15].

Acyklický graf je orientovaný graf, který neobsahuje ani jeden cyklus

Strom je souvislý acyklický graf. Mezi každými dvěma vrcholy je právě jedna existující cesta. Jeden z vrcholů je označen jako kořen (root) a ostatní jsou rozděleny do podmnožin, které tvoří také stromy a mají svůj kořen. Více stromů tvořící nesouvislý graf se nazývá les [15,17].



Obr 3.6: Přiklad stromu [15].

Vážený graf je graf, který má ke každé hraně přidělenou váhu, ta je obvykle přidělena váhovou funkcí. Velikost váhy může reprezentovat například u grafu sítě vytížení hrany, která představuje konkrétní linku [15].

3.3. Cesta grafem

Velmi častou úlohou u teorie grafů je nalezení nejkratší nebo nejrychlejší cesty z jednoho vrcholu do druhého. V praxi to může představovat například plánování trasy v dopravě nebo hledání nejkratší cesty paketu v IP sítích. Pokud analyzujeme složitější graf s větším počtem

vrcholů, je nevyhnutelné použít algoritmy pro hledání cesty v grafu. Tyto algoritmy řadíme do dvou kategorií [15].

3.3.1. SLEPÉ PROHLEDÁVÁNÍ

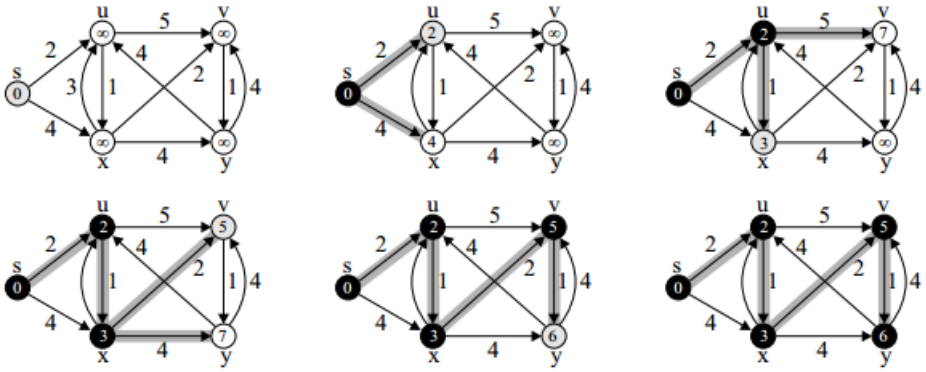
Algoritmy patřící do kategorie slepého prohledávání vyhledávají náhodně, takzvaně bez „přemýšlení", bez predikce výsledku a tudíž bez využití heuristiky. Slepé prohledávání nevyužívá priorit pro procházení stavů [15,18]. Mezi tyto algoritmy patří:

Prohledávání do hloubky označovaný jako DLS (z anglického názvu „Deep-limited search") postupuje od nejvzdálenějších stavů a postupuje směrem k počátečnímu stavu, dokud nedospěje k výsledku. Směr vyhledávání je zvolen náhodně. Algoritmus vždy dospěje k výsledku, je tudíž úplný. DLS není optimální protože upřednostňuje některé větve, a tak může dospět k výsledku, který není nejkratší [15,18].

Prohledávání do šířky, BFS (Breadth-first search) postupuje od stavů blízkých počátečnímu stavu a postupně prochází vzdálenější a vzdálenější stavy, dokud nedospěje k výsledku. BFS využívá kontrolu opakování průchodu stavů, aby nenastaly cykly, což znamená značné zpomalení algoritmu při hledání cesty, ale zase garantuje konvergenci k výsledku. Lze použít i variantu bez kontroly, ale jen v ojedinělých případech, kde jsme si jisti, že nenastane žádný cyklus. BFS je úplným algoritmem a tudíž nalezne vždy řešení, pokud existuje. Algoritmus je optimální a vždy nalezne cestu s nejmenším počtem kroků [15,18].

Dijkstrův algoritmus je nejrychlejší algoritmus pro nalezen nejkratší cesty grafem a byl popsán Edsgerem Dijkstrou v roce 1959. Algoritmus lze chápat jako zobecnění postupu prohledávání grafu do šířky, neboť také začíná u stavů nejblíže k počátku, ale na rozdíl od BSF se nešíří podle počtu hran od zdrojového uzlu, ale podle „vzdálenosti" od zdroje danou váhou. Algoritmus zpracovává tedy stavy, k nimž je nalezena nejkratší cesta [15,19].

Dijkstrův algoritmus si uchovává uzly v prioritní frontě a řadí je dle vzdálenosti ke zdroji. První iterace je zdroj samotný, tudíž je vzdálenost nula a ostatní nekonečno (neznámá vzdálenost). Další krok vybere uzel s nejvyšší prioritou, to jest nejnižší hodnota vzdálenosti od již zpracovaných stavů. Tento uzel je zařazen mezi zpracované, jeho potomci jsou přidáni do prioritní fronty (pokud již nejsou zpracováni) a algoritmus ověřuje, zda nejsou blíže ke zdroji, než byli před zařazením právě zpracovávaného uzlu mezi zpracované, tudíž že od nich není ke zdroji kratší cesta než přes právě prohledávaný uzel. Pokud je tato podmínka splněna, celou iteraci opakujeme, dokud nedospějeme k cílovému uzlu [19,20].



Obr 3.7: Ukázka průběhu Dijkstrova algoritmu [19].

Dijkstrův algoritmus lze použít pouze pro grafy s kladným ohodnocením hran, jinak nelze zaručit správnost výsledku [20].

3.3.2. INFORMOVANÉ METODY

Informované metody upřednostňují některé stavy aby nalezly dříve řešení, což potřebuje „fitness funkci", která ohodnotí každý stav. Právě tato funkce zapříčiní zvýšení časové náročnosti algoritmu [15]. Mezi základní informované metody patří:

Best FS, což je optimalizace neinformované metody BFS. Nejdříve expanduje podobně jako BFS, ale pro pokračování si vybere „nejslibnější" uzel pomocí Fitness funkce:

$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k) \qquad (3.1)$$

Kde $g(S_k)$ je funkce zobrazení, která ohodnocuje váhu projitých hran. $h(S_k)$ je heuristická funkce pokoušející se odhadnout jak je blízko k hledanému stavu S_G.

Implementace Best FS je stejná jako BFS s tím rozdílem, že místo fronty použije frontu prioritní, kde se stavy Sk řadí podle hodnoty $F(S_k)$. Jelikož Best FS využívá heuristiky, tak je algoritmus neoptimální [15,18].

A* algoritmus je velice podobný Best FS, ale navíc bere v potaz kompletní cestu, kterou algoritmus prošel, díky tomu je algoritmus kompletní i optimální [15].

Hill Climbing optimalizuje řešení z pohledu času. Vychází z BFS a je mu velmi podobný, s tím rozdílem, že maže starší řešení a ponechává pouze poslední expandované stavy. Díky této optimalizaci má daleko nižší nároky na paměť, ale algoritmus už není úplný [15].

4. APLIKACE A POUŽITÉ NÁSTROJE

Cílem projektu byl vývoj aplikace pro testování počítačových sítí sestavených pomocí směrovačů od firmy Mikrotik. Aplikace je psána v programovacím jazyce Python a je určena pro operační systém Linux. Aplikace simuluje provoz na síti a vyhodnocuje výsledky.

4.1. PYTHON

Pro vývoj aplikace byl zvolen Python, konkrétně verze 2.7.6. Python vznikl již v roce 1991 a jeho návrhářem je programátor Guido van Rossum a jedná se vysokoúrovňový skriptovací interpretovaný programovací jazyk podporující paradigma objektově orientované, ale i procedurální a funkční programování. [5]

4.1.1. PROČ PYTHON

Díky své poměrně jednoduché regulární syntaxi roste na popularitě a umožňuje velice rychlý vývoj. Příkladem může být syntaxe ukončení příkazu koncem řádku, kde odpadá velmi častá začátečnická chyba u práce s C++ nebo Javou, kdy programátor opomene středník na konci řádku. Podobně je tomu u bloků příkazů, které je označeno odsazením a nestane se chyba při kompilaci, kdy chybí uzavření bloku závorkou (na druhou stranu je nutné dbát na odsazení pomocí znaku tabulátoru nebo mezer a nelze je kombinovat). [5,6]

Typování proměnných v Pythonu je dynamické a není potřeba žádné deklarace, čímž se opět zkrátí délka kódu. Co se týče řídících struktur, tak Python má jak rozhodovací konstrukce `if`, `elif` a `else`, tak `for` cyklus a smyčku `while`. Podobně jako C++ či Java má i Python podporu pro zpracovávání výjimek, používání tříd objektově orientovaného programování, ale zároveň je nevyžaduje a nejsou nutné. Některé aspekty objektově orientovaného programování, jako například dědičnost, jsou do značné míry, zjednodušeno oproti klasickým jazykům. [6]

Další velikou výhodou je modulárnost Pythonu. Díky rostoucí uživatelské základně je k dispozici velká řada modulů, které si můžeme importovat a opět si usnadnit práci a ušetřit čas při vývoji. [6]

4.1.2. NEVÝHODY PYTHONU

Po zmínění výhod pythonu musíme zmínit i zápory, které jeho používání s sebou přivádí, proč není ještě více rozšířený a je mnohými programátory zavrhovaný.

Mezi jednu z nejzávažnějších nedostatků, které si nesou všechny interpretované jazyky, je jejich rychlost. Ta není ve všech případech dostačující a je nutné sáhnout po překládaných jazycích. S rostoucí popularitou skriptovacích jazyků je vyvíjen tlak na jejich vývojáře, aby byli efektivnější a rychlejší. Problémy s rychlostí pomalu odpadají také díky neustálému vývoji hardwaru. [6]

Z výše zmíněné nevýhody plyne i absence Pythonu u aplikací pro mobily, tablety a také není ve webových prohlížečích. Pro některé je i nevýhodou dynamické typování, které může zapříčinit velmi těžko detekovatelné chyby, objevitelné až za chodu a vyžadující detailnější testování. [7]

Python je rostoucí a stále se rozšiřující programovací jazyk s velkou uživatelskou základnou. Jakožto interpretovaný skriptovací jazyk má své uplatnění při tvorbě aplikací, které nekladou velké požadavky na čas a paměť. Zároveň je ale poslední dobou čím dál více oblíbených skriptovacích jazyků (Perl,Visual Basic, Tcl) nejvíce podobných tradičním

systémovým jazykům. Další fakt, který podporuje rozvoj Pythonu je trend IOT (internet of things), kde Python hojně využíván, například na platformě jako Raspherry Pi [6,7].

4.2. SQL

SQL (Structured Query Language), dříve známý též jako SEQUEL (Structured English Query Language) je standardizovaný jazyk pro přístup a manipulaci s relačními databázemi. První prototyp SQL byl představen IBM již v roce 1979 a stál na modelu Dr. Edgara F. Codda popsaném ve známém článku *A Relational Model of Data for Large Shared Data Banks*. Jeho problémem však bylo, že tento model předběhl dobu a nebyl vhodný pro tehdejší hardware. Proto se tento model docenil až v pozdějších letech, hlavně společností Oracle Corporation [22].

4.2.1. SQLite

V naší práci pro správu a vytváření databází používáme SQLite, což je také relační databázový systém, který je ale zjednodušen, avšak zůstává stále samostatným. SQLite je zdarma jak pro soukromé, tak i komerční účely a je to jeden z nejrozšířenějších databázových jazyků na světě. Na rozdíl od ostatních SQL databází nepotřebuje SQLite žádný serverový proces a samo zapisuje přímo do souborů, kde tvoří kompletní SQL databáze, které mohou obsahovat více tabulek, oindexování položek a podobně [23].

Všechny funkce SQLite jsou schované v malé knihovně o velikosti kolem 500 kB v závislost na nastavení cílové platformy a optimalizace kompilátoru. Dále je SQLite velmi šetrné k náročnosti na paměť, díky tomu může být využívaná i na zařízeních jako jsou mobilní telefony, PDA či MP3 přehrávače, neboť se velikost potřebné paměti dá nastavit a zmenšit na úkor rychlosti [23].

Avšak to, na čem si SQLite zakládá nejvíce, je snadné použití a přívětivá syntaxe, kterou si uživatel rychle osvojí a i díky tomu je SQLite tolik populární.

4.3. NetworkX

Aplikace vykresluje diagramy zadaných sítí a pro tuto činnost využíváme nástroj NetworkX, který je balíčkem pro jazyk Python a umí vytvářet a manipulovat s různorodými grafy v nejrůznějších podobách. S NetworkX můžeme načítat sítě a ukládat v různých datových formátech, vytvářet náhodné sítě, analyzovat strukturu, tvořit síťové modely, síťové algoritmy a mnoho dalšího [24].

NetworkX byl vyvinut softwarovými designéry Aricem Hagbergem, Danem Schultem a Pieterem Swartem v letech 2002 a 2003. Jejich cílová skupina byli matematici, fyzici, biologové, počítačový odborníci a podobně [24].

4.4. PyQt

Pro tvorbu našeho GUI používáme PyQt, což je modul zajišťující vazbu Pythonu, ať už verze v2 nebo v3, s populárním GUI nástrojem Qt, který je multiplatformní a je možné ho použít na Windows, MacOS i Linuxu. Nejnovější verze je PyQt v5.5.1, ale jsou stále podporovány i starší verze [25].

Qt třídy využívají signály a sloty jako komunikaci mezi objekty. Signály objektu Qt vyšlou zprávu a když dorazí, tak se spustí určitá událost. Signály mohou předávat libovolný počet argumentů slotu, což je právě metoda, která se spustí v reakci na signál [26].

4.5. QT DESIGNER

Pro zrychlení tvorby oken a jejich komponentů používáme nástroj Qt Designer, který nám umožní interaktivně vytvářet grafické návrhy a posléze pouhým tahem přidávat oknu jednotlivé komponenty a upravovat jejich velikost a podobně. Také nám umožní propojovat signály a sloty v grafickém módu. Nakonec vše přeloží do xml kódu, který si můžeme přeložit do Pythonu [27].

Struktura vygenerovaného kódu odpovídá kódu psanému klasicky v Qt a tvoří jednu třídu s názvem podle kořenového objektu stanoveného programátorem s příponou „Ui_". Abychom takto vytvořený kód mohli v průběhu upravovat dle potřeb, je vhodné ho použít nepřímo jako modul a následně ho importovat do hlavní části programu [27].

4.6. NETADDR

Dalším nástrojem, který v naši aplikaci používáme, je modul Netaddr. Je to knihovna pro Python pracující na druhé a třetí vrstvě referenčního ISO/OSI modelu a obsahuje podporu IPv6. Netaddr obsahuje metody pro klasifikaci adres a rozdělení na IPv4 a IPv6, rozdělení do sítí, výpočet masky, prefixu, sumarizace a zařazování do tříd. Modul má licenci BSD a jeho autorem je David P.D.Moss [30].

4.7. MATPLOTLIB

Pro tvorbu grafů výsledků měření jsme vybrali knihovnu Matplotlib. Matplotlib je Python modul pro generování 2D grafů. Možnosti využití jsou široké, ať už chce uživatel vykreslit jednoduchý bodový graf závislosti, sloupcový graf, histogram či výkonové spektrum. Použití je jednoduché, intuitivní a s pomocí dokumentace a vzorových příkladů si je uživatel rychle osvojí [31].

Výsledky modulu Matplotlib jsou podobné těm, které známe z programu MATLAB. Uživatel má kontrolu nad výsledným vzhledem, stylem, barvou čar, popisky os a podobně. S vygenerovaným grafem může uživatel manipulovat, zvětšit si důležité části, ukládat grafy jako obrázky a podobně. Modul je svobodný software pod licencí PSF a autorem je John Hunter [31].

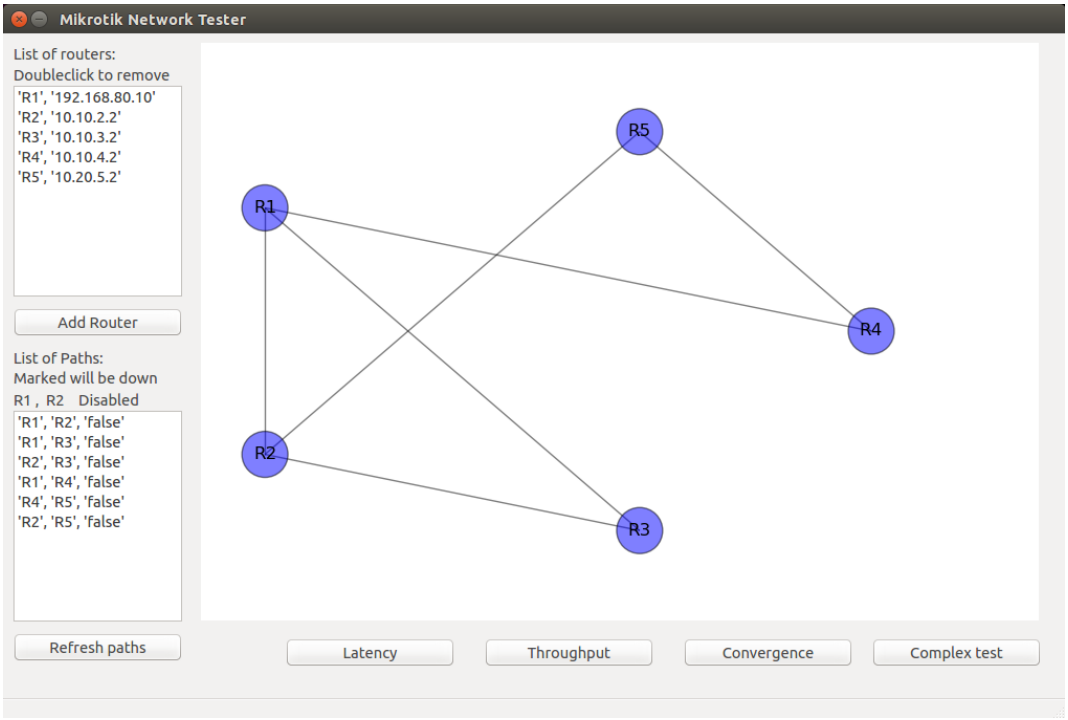
5. Aplikace

5.1. Hlavní okno Aplikace

Při spuštění uživatel vidí hlavní okno programu, které jsme vytvořili pomocí PyQt a jehož zdrojový kód jsme generovali pomocí grafického rozhraní na tvorbu oken a ostatních komponentů, programem Qt Designer.

Okno samotné je objekt s názvem `MainWindow` pro třídu `QMainWindow` a je v něm umístěno několik komponentů. Objekty typu `QMainWindow` slouží pro tvorbu uživatelských rozhraní aplikace a mají rozsáhlejší možnosti než zjednodušená verze `QWidged`. Prvním z komponentů v hlavním okně je seznam směrovačů ovládaný tlačítkem „Add Router". Podobně jako seznam směrovačů je tu i seznam jejich cest, které je spojují a tyto cesty jsou přidávány automaticky po přidání směrovače. Pod seznamem cest je tlačítko pro jednoduchou konfiguraci sítě. Uživatel může vybrat v seznamu cesty, které chce vypnout a po stistknutí „Refresh Path" tlačítka dojde k zakázání vybraných cest, pokud to redundantnost sítě dovolí a všem směrovačům zůstane cesta k ostatním prvkům sítě.

Dále okno obsahuje oblast pro vyobrazení diagramu sítě. Diagram se vytváří při inicializaci hlavního okna a dále se aktualizuje při přidání či odebrání směrovače a při manipulaci s rozhraními pro zapínání nebo vypínání cest.



Obr 5.1: Ukázka náhledu na hlavní okno aplikace

Při vytváření okna pomocí Qt Designer jsme si nejprve po rozložení a upravení velikostí všech komponentů vygenerovali „xml" kód. Jelikož pro vývoj používáme skriptovacího jazyku Python, tak jsme si museli překonvertovat zdrojový kód do našeho programovacího jazyka, abychom k němu měli přístup, a to pomocí nástroje „pyuic4", což je nástroj používaný v příkazové řádce pro tvorbu Python modulu z „.ui" souboru.

Nyní máme připravený Python modul, který můžeme importovat do našeho programu a díky tomu ho můžeme dále editovat, aniž bychom museli přepisovat nějaký kód. Stačí provést editaci v Qt Designeru a pomocí „pyuic4" převést nový „xml" soubor do Python kódu a tím aktualizovat modul s hlavním oknem importovaný do naší aplikace.

Výpis kódu 5.1: Ukázka Python kódu pro hlavní okno

```
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s
try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(1011, 655)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.verticalLayoutWidget = QtGui.QWidget(self.centralwidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(10, 10, 161, 201))
...
```

Když už máme připravený a importovaná modul s hlavním oknem, tak pro něho v aplikaci vytvoříme třídu a inicializační funkci.

Výpis kódu 5.2: Vytvoření a inicializace hlavního okna

```
 def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.setupUi(self)

        # Insert Plot and fill Lists
        QtCore.QTimer.singleShot(500,self.plotFunc)
        QtCore.QTimer.singleShot(500,self.getRouters)
        QtCore.QTimer.singleShot(500,self.getPaths)

        #Creating RoutersList database if not present
        con = lite.connect('RoutersList.db')
        with con:
            cur = con.cursor()
            cur.execute("CREATE TABLE IF NOT EXISTS Routers(Name, IP, Username,
Password TEXT)")
            cur.execute("CREATE TABLE IF NOT EXISTS Paths(R1 , IP1 , R2 , IP2,
disabled TEXT)")
```

Jak je vidět, tak při inicializaci voláme metody `plotFunc` a `getRouters` přes metodu `singleShot`, která umožňuje jednorázové a nebo i opakované spuštění časovače, které nám po vypršení zavolá výše zmíněné metody zadané jako parametr.

Dále zkontrolujeme, jestli existuje databáze obsahující tabulku směrovačů a cest. Tato databáze je uložená jako soubor „RouterList.db", což značí databázi vytvořenou pomocí SQLite. Nejprve se připojíme do databáze, popřípadě se vytvoří, pokud neexistuje pomocí příkazu „`lite.connect('RoutersList.db')`" a uložíme si ji do proměnné `con`. Dále k databázi přistupujeme pomocí „`with con:`". Poté používáme `cursor` , což je objekt, či kontrolní struktura SQLite databáze umožňující nám procházet, přidávat a mazat záznamy v databázi. „`CREATE TABLE IF NOT EXISTS [table]`" příkazem vytvoříme tabulku, pokud neexistuje. Nejprve zkusíme vytvořit tabulku „Routers", která bude mít sloupce „Name", „IP", „Username" a „Password". Obdobně zkusíme vytvořit tabulku s cestami mezi směrovači „Paths" se sloupci pro první směrovač a jeho IP adresu a druhý směrovač s IP adresou. Nakonec přidáme poslední sloupec „disabled", do kterého podle vzoru Mikrotik směrovačů zapisujeme, jestli je cesta povolena, to znamená, že proměnná `disabled` je nastavena na `false` nebo zakázána, kdy `disabled` je nastaveno na `true`.

Dále máme připraveny události spouštěné po zachycení signálu jako stisknutí tlačítka, kliknutí do seznamu a podobně.

Výpis kódu 5.3: Zachytávání signálů a spouštění událostí

```
# button for opening windows SendingWindow
# note: Lambda is anonymous function allowed us pass extra arguments
self.btnOpenSendingWindow.clicked.connect(lambda:
       self.showWindow('SendingWindow'))
self.btnOpenSendingWindowThroughput.clicked.connect(lambda:
       self.showWindow('SendingWindowThroughput'))
self.btnOpenSendingWindowConvergence.clicked.connect(lambda:
       self.showWindow('SendingWindowConvergence'))

# button add Router
self.btnAddRouter.clicked.connect(lambda: self.showWindow('RouterWindow'))

# Event after doubleclick on item in Qlistwidget
self.listWidget.itemDoubleClicked.connect(self.deleteItem)

# Event after click on item in listWidgetPath
# Set multiselection mode on listWidgetPath to be able select more paths
self.listWidgetPath.setSelectionMode(QtGui.QAbstractItemView.MultiSelection)
self.listWidgetPath.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)

# Click on the button Refresh Paths
self.btnTest.clicked.connect(self.disablePaths)
```

Nejprve máme signál na stisknutí tlačítek pro samotné testy. Po stisknutí dojde k události, která zavolá metodu `showWindow` za pomoci anonymní funkce „lambda", která nám umožní předat další argument jako název okna, které chceme otevřít. Stejným způsobem otevíráme okno pro přidávání směrovačů, pro nastavení testů a podobně.

Dvojklikem na směrovač v seznamu zachytíme signál, který nám spustí metodu `deleteItem`. Seznam s cestami si pro naše účely musíme nejprve trochu upravit, a to tak, abychom umožnili označení více elementů v seznamu než jeden. To uděláme změnou vlastnosti `setSelectionMode` do `MultiSelection` módu.

Nakonec je tu zachytávání signálu stisknutí tlačítka „Refresh Paths", kterým zavoláme metodu `disablePaths` a aktualizujeme stav cest.

5.2. PŘIDÁVÁNÍ SMĚROVAČŮ

Směrovače jsou zobrazeny v seznamu v horní levé části hlavního okna, kde je uvedeno jejich jméno a rozhraní, přes které jsme se k němu přihlásili. Tento seznam je tvořen při inicializaci hlavního okna a získává informace z tabulky v databázi „RouterList.db".

Pro přidání dalšího směrovače je zde tlačítko označené jako „Add Router". Po jeho stisknutí se spustí událost volající metodu `showWindow`, které jako parametr zadáme jméno okna pro přidávání směrovačů „RouterWindow".

5.3. OKNO PRO PŘIDÁNÍ SMĚROVAČE

Funkce `showWindow` nám zobrazí okno definované jako „RouterWindow", které jsme si vytvořili jako ostatní okna pomocí Qt Designeru, přeložili do Pythonu a importovali jako modul do naší aplikace. Okno obsahuje čtyři textová pole popsaná popisky „label". První pole je pro zadání IP adresy rozhraní, přes které chceme k směrovači přistupovat a druhé pole je pro zadání názvu směrovače. Dále tu máme pole pro vyplnění přihlašovacího jména a hesla. Nakonec tu máme tlačítko „Add", jehož stisknutím potvrdíme přidání směrovače.

Obr 5.2: Okno pro přidávání směrovačů

`RouterWindow` tvoří vlastní třídu stejným způsobem, jako hlavní okno programu. Při spuštění okna dojde k zavolání inicializační funkce `__init__` , která mimo jiné obsahuje zachycení signálu o stisknutí tlačítka „Add" a spuštění události, která zavolá metodu `addRouterFunc`.

Nejprve získáme informace z textových polí. K tomu použijeme metodu `text`. Dále zkontrolujeme obsah načtených dat pomocí série `If` konstrukcí.

Výpis kódu 5.4: Třída pro RouterWindow

```
routername=self.txtRouterName.text()
result = self.CheckIfExist(str(routername))
        if result == False:
            self.ErorrMsg("Error: Router with this name already exist")
            return
# Call Function to open socket
        result = self.openSocket(ipaddr, 8728)
        if result == False:
            self.ErorrMsg("Error: Check your IP address or connectivity with
target")
            return
        s = result

# Call Function to Login
        result = self.loginToRouter( s, username, password )
        if result == False:
            self.ErorrMsg("Error: Check your Username or password")
            return
        apiros = result
```

Kontrolujeme, jestli zde již není směrovač s tímto jménem, pokud takový směrovač existuje, tak se otevře okno s varováním. To zajišťuje metoda `CheckIfExist`, kterou voláme s parametry `self` a `routername`, což je jméno směrovače. Metoda se připojí do databáze „RoutersList" a příkazem `SELECT` vybere z vygenerované tabulky „sqlite_master" všechna jména tabulek, které si pomocí funkce `fetchall` uložíme do proměnné `tablerow` a `for` cyklem projedeme celé `tablerow`, které je typu tuple (struktura podobná seznamu) a `if` konstrukcí hledáme shodu jména tabulky s argumentem `routername`. Metoda má návratovou hodnotu `False`, pokud najde schodu. Dále kontrolujeme, jestli můžeme navázat BSD schránku (v anglicky psané literatuře „socket") s cílovou IP adresou. Pro otevření schránky slouží metoda `openSocket`, které jako parametry zadáme pro nás dostupnou IP adresu a port, na kterém má být schránka otevřena. Pokud je schránka navázána úspěšně, uložíme si schránku do proměnné `s`. Naopak pokud spojení selhalo, tak nám vyskočí chybová hláška, abychom si zkontrolovali připojení s cílovým směrovačem.

5.4. ZÍSKÁNÍ IP ADRES

Nakonec se zkusíme ke směrovači přihlásit pomocí vlastní MikrotikAPI, kdy zavoláme funkci `loginToRouter`, které zadáme jako parametry vytvořenou schránku a přihlašovací údaje. Při neúspěšném přihlášení je uživatel informován chybovou hláškou.

Po úspěšném přihlášení si vytvoříme seznam `inputsentense`, který rozšíříme pomocí metody `append` o příkaz `/ip/address/print`. Seznam zadáme jako parametr metodě `writeSentence`. Dále ve `for` cyklu přijímáme odpovědi směrovače po řádku pomocí metody MikrotikAPI `readSentence` a ukládáme si je do proměnné `x`. Poté kontroluje konstrukcí `if`, jestli v odpovědi není obsažen řetězec `!done`, který značí konec odpovědi. Nyní si metodou `findall` z modulu `re` (regular expression operations) najdeme všechny IP adresy, kdy adresu nahradíme regulárním výrazem `r'[0-9]+(?:\.[0-9]+){3}/[0-9]{1,2}'`. Dále si z odpovědi uložíme jméno rozhraní a hodnotu proměnné `disabled`, která je `true`, je-li rozhraní vypnuté a `false` značí rozhraní zapnuté.

Poté zavoláme metodu `CheckIPAddr`, která kontroluje, jestli není v databázi adresa ze stejné sítě. Nakonec uložíme získaná data do tabulky, kterou pojmenujeme, stejně jako jsme pojmenovali směrovač. Zadáme do tabulky IP adresy, názvy rozhraní s IP adresami, na kterých je stav `disabled`.

Výpis kódu 5.5: Přidání směrovače

```
inputsentence = []
inputsentence.append("/ip/address/print")
apiros.writeSentence(inputsentence)

for n in range(1,20):
    r = select.select([s], [], [], None)
    x = apiros.readSentence()

    if "!done" in x:
         break

    ip = re.findall( r'[0-9]+(?:\.[0-9]+){3}/[0-9]{1,2}', (str(x)) )

    intrf = str(x[4])[11:]
    disabled = str(x[8])[10:]

    self.CheckIpAddr(ip, routername, disabled)

    with con:
        cur = con.cursor()

    cur.execute("INSERT INTO %s VALUES (?,?,?)" % str(routername),
(str(ip),str(intrf),str(disabled)) )

message= []
message.append("/quit")
self.sendMessage(message, apiros )

self.addItemToList()
```

Nakonec ukončíme komunikaci se směrovačem posláním příkazu `/quit` odeslaným metodou `sendMessage` a zavoláme metodu `addItemToList` pro přidání směrovače do seznamu směrovačů umístěném v hlavním okně, ale také přidáme záznam o směrovači do tabulky „Routers", která obsahuje seznam směrovačů a jejich IP adresy, přes které byly přidány.

Metoda získá data z textových polí pomocí funkce `text` a uloží je do proměnných, připojí se k databázi „RoutersList" a pomocí příkazu `INSERT` s položkou `INTO` určující do které tabulky, položkou `VALUES`, do níž vložíme hodnoty jako proměnné, tedy jméno směrovače, IP adresu, uživatelské jméno a heslo. Následně načteme všechny řádky z tabulky „Routers" příkazem `SELECT` a opět pomocí metody `fetchall` si uložíme záznamy do proměnné `rows`. Každý řádek tvoří element `rows`, který je typu `tuple`. Před samotným vkládáním do seznamu smažeme předchozí obsah seznamu funkcí `clear` a pomocí `for` cyklu vkládáme jednotlivé elementy z `rows` do seznamu, který je tvořen jako objekt typu `QlistWidget`. Samotné vkládání je metodou `addItem`, kde jako parametr zadáme první a druhé slova z elementu v `rows`, který upravíme ořezáním uvozovek pro vzhlednější a

přehlednější záznam. Na konci této metody se volá další metoda `plotFunc`, které slouží k tvorbě grafu.

Výpis kódu 5.6: Metoda pro přidání položek do seznamu

```
def addItemToList(self):
        # Get text from texboxes to variables
        IPaddr=self.txtIPaddr.text()
        RouterName=self.txtRouterName.text()

        username = self.txtUserName.text()
        password = self.txtPassword.text()
        # Insert item to database and refresh list with routers
        con = lite.connect('RoutersList.db')
        with con:
            cur = con.cursor()
            cur.execute("INSERT INTO Routers VALUES (?, ?, ?, ?)", (str
(RouterName), str(IPaddr), str(username), str(password)))
            cur.execute("SELECT * FROM Routers")
            rows = cur.fetchall()
            self.parent().listWidget.clear()

            for row in rows:
                #self.parent().listWidget.addItem('Name: %s, %s' % row)
                additem=str(row[0]),str(row[1])
                self.parent().listWidget.addItem(str(additem)[1:-1])

        # Calling plotFunc to refresh plot in MainWindow
        self.parent().plotFunc()
```

5.5. GENEROVÁNÍ CEST

Při přidávání směrovače byla volána metoda `CheckIpAddr`, která kontroluje, jestli již nebyl přidán směrovač, který by měl IP adresu ze stejného rozsahu. Tuto metodu voláme při obdržení řádku odpovědi na příkaz `/ip/address/print`, který obsahuje IP adresu a voláme ji s parametry `self`, IP adresa `ip`, jména směrovače `routername` a stav IP adresy `disabled`.

Jelikož odpovědi směrovače jsou ve formátu `tuple`, tak si upravíme řetězec s IP adresou tak, že odřízneme závorky a uvozovky. Dále se připojíme do databáze „RoutersList" a projedeme automaticky tvořenou tabulku „sqlite_master", která obsahuje názvy všech tabulek obsažených v databázi, čímž získáme jména tabulek již přidaných směrovačů za použití `for` cyklu, kde voláme funkci `fetchall`, jejíž návratová hodnota je jméno tabulky. Pro to slouží příkaz `SELECT name FROM sqlite_master WHERE type='table'`. `SELECT` je obvyklý příkaz SQLite pro dotaz do databáze, `FROM` položka určuje tabulku a `WHERE` položka slouží jako filtr s vyhledávací podmínkou.

Takto získané jméno tabulky použijeme jako proměnnou pro položku `FROM` a necháme si funkci `fetchall` uložit všechny řádky této tabulky do proměnné typu `tuple rows`, kde každý řádek tvoří jeden element. Tuto proměnnou projedeme `for` cyklem a její první slovo z každého elementu, neboli první sloupec, si uložíme do proměnné `ipR2`. Zkontrolujeme konstrukcí `if`, jestli je adresa povolená a nakonec použijeme opět konstrukci `if` pro porovnání `ipR2` a `ip`, což je proměnná obsahující adresu, kterou jsme zadali jako parametr metodě `CheckIpAddr`. Porovnání je provedeno díky modulu „netaddr", který nám

umožňuje porovnávat IP adresy, jestli jsou ze stejného rozsahu sítě (`IPNetwork(ip) == IPNetwork(ipR2)`). Pokud je podmínka splněna a adresy jsou ze stejného rozsahu, tak zavoláme metodu `AddPath`, která nám přidá takto nalezenou cestu do tabulky „Paths".

Výpis kódu 5.7: Hledání adresy ze stejného rozsahu

```
def CheckIpAddr(self, ip, routername, disabled ):
        ip=str(ip)[2:-2]

        con = lite.connect('RoutersList.db')
        with con:
            cur = con.cursor()
            cur.execute("SELECT name FROM sqlite_master WHERE type='table'")
            for tablerow in cur.fetchall():
                routernameR2 = tablerow[0]

                if routernameR2 not in ('Routers','Paths', "%s" % routername) :
                    cur.execute("SELECT * FROM %s" % routernameR2)
                    rows = cur.fetchall()

                    for row in rows:
                        ipR2 = row[0]
                        ipR2 = ipR2[2:-2]

                        if "false" in disabled and "false" in row[2] :
                            disabledR1R2 = "false"
                        else :
                            disabledR1R2= "true"

                        if IPNetwork(ip) == IPNetwork(ipR2):
                            self.AddPath(ip, ipR2, routername, routernameR2,
disabledR1R2 )
```

Metoda `AddPath` je volána s parametry `self`, IP adresou `ip`, IP adresou druhého směrovače `ipR2`, jménem směrovače `routername` a jménem druhého směrovače `routernameR2`. Poslední parametr `disabled` označuje, jestli je cesta aktivní. Dojde k připojení do databáze a vložení příkazem `INSERT` s položkou `INTO` směřující do tabulky „Paths" a hodnoty `VALUE` jsou proměnné, které byly zadány jako parametry samotné metodě.

Nakonec přidáme cesty do seznamu cest na hlavním okně, který je tvořen stejně jako seznam směrovačů pomocí objektu `QListWidget`, jemuž přidáme jednotlivé položky (zvané „items") metodou `addItems`. Každá položka obsahuje názvy směrovačů, které spojuje a také svůj stav `disabled`.

Výpis kódu 5.8: Přidání cesty do databáze a seznamu cest

```
    def AddPath(self, ip, ipR2, routername, routernameR2, disabled):
        con = lite.connect('RoutersList.db')
        with con:
            cur = con.cursor()
            cur.execute("INSERT INTO Paths VALUES (?, ?, ?, ?, ?)", (
str(routername), str(ip), str(routernameR2), str(ipR2), str(disabled) ) )

        path = "'"+str(routernameR2)+"'" + ", " + "'"+str(routername)+"'"+ ", "
+ "'"+str(disabled)+"'"
        self.parent().listWidgetPath.addItem(path)
```

5.6. MAZÁNÍ SMĚROVAČŮ

Jak bylo popsáno u inicializace hlavního okna, tak po dvojkliku na položku ze seznamu směrovačů dojde k vymazání směrovače. V aplikaci to znamená zavolání metody `deleteItem`, která si uloží obsah právě aktivované položky metodou `currentItem`. Z takto získaného řetězce znaků pomocí metody `split` získáme první slovo řetězce, kterým je název směrovače a uložíme si jméno do proměnné `name`.

Nakonec je zavolána metoda `takeItem`, která slouží k odebrání položky ze seznamu a jako parametr ji zadáme návratovou hodnotu funkce `currentRow`, která navrátí index používaného řádku.

Poté se připojíme do databáze „RoutersList" a příkazem `DELETE` vymažeme z tabulky směrovačů „Routers" ty záznamy, které mají ve sloupci „name" řetězec shodný s obsahem proměnné `name`. Dále smažeme tabulku se záznamy o právě smazaném směrovači. Tato tabulka má stejný název jako samotný směrovač. K mazání tabulek slouží u SQLite příkaz `DROP`, který ještě podmíníme podmínkou `IF EXISTS` a jméno tabulky zadáme naši proměnnou `name`.

Nakonec řešíme vymazání cest spojených s tímto směrovačem. Nejprve začneme mazat v tabulce s cestami „Paths" pomocí SQLite příkazu `DELETE` a jako podmínku zadáme obsah sloupce „R1" a poté „R2". Aby SQLite nevypsalo chybu, kdyby nebyl přítomný žádný shodující záznam, je mazání umístěno v `try` bloku.

Posledním krokem je volání metody `deletePath`, kterou voláme ve smyčce `while`, dokud nám `deletePath` vrací hodnotu `True` a zadáváme ji jako parametr jméno mazaného směrovače `name`. Metoda slouží k odstraněné položek obsahujících jméno směrovače. Cyklem `for` prohledáme jednotlivé položky seznamu cest tak, že mu zadáme jako rozsah počet položek v seznamu, který získáme metodou `count`. Poté pomocí konstrukce `if` zkontrolujeme, zdali je název směrovače obsažen v položce. Je-li podmínka splněna, tak položku smažeme metodou `takeItem`.

Výpis kódu 5.9: Vymazání položek ze seznamu cest

```
def deletePath(self, name):
        for item in xrange(self.listWidgetPath.count()):
            if name in self.listWidgetPath.item(item).text():
                self.listWidgetPath.takeItem(item)
                return True
        return False
```

Nyní jsou záznamy o smazaném směrovači vymazány a můžeme aktualizovat diagram sítě.

5.7. GENEROVÁNÍ DIAGRAMU SÍTĚ

Pro vizuální zobrazení testované sítě máme v aplikaci vykreslování diagramu, které provedeme už při inicializaci hlavního okna, také při přidání a mazání směrovače a nakonec při zapínání a vypínání cest mezi směrovači. Pro samotné vykreslení nám slouží metoda `drawGraphFunc`, kterou můžeme volat s parametrem `*disabledPath`.

Jak již bylo řečeno, pro tyto účely používáme modul NetworkX importovaný jako `nx`. Nejprve si vytvoříme prázdnou strukturu grafu, kterou pojmenujeme `G`. Dále se připojíme do databáze „RoutersList" a příkazem `SELECT` vybereme sloupec se jmény „Name" z tabulky

„Routers" a funkcí `fetchall` si je uložíme do proměnné „RouterList", která je typu `tuple` a bude nám sloužit jako seznam vrcholů grafu. Ty jsou v NetworkX nazývány „Nodes".

Poté z tabulky „Paths" podobným způsobem získáme obsahy sloupců „R1", „R2" a „disabled" a uložíme si je do proměnné `rows`.

Jednotlivé cesty ještě zkontrolujeme, jestli jsou povoleny. To provedeme prohledáním proměnné `rows`, kde u jednotlivých elementů hledáme konstrukcí `if`, jestli obsahují řetězec `'false'`, pokud je podmínka splněna, vložíme funkcí `append` tento element do `tuple PathList`, který tvoří seznam hran grafu zvané „Edges".

Pokud je metoda `drawGraphFunc` volána s parametrem, tak z `PathList` odebereme prvky shodné s elementy parametru `disabledPath`. To slouží pro kontrolu spojitosti grafu před vypnutím některého z rozhraní. Je-li metoda volaná bez parametru, tato část se přeskočí.

Nyní už jen přidáme do grafu `G` vrcholy funkcí `add_nodes_from`, které jako parametry zadáme `tuple RouterList` a stejným způsobem za použití metody `add_edges_from` volané s parametrem `PathList` přidáme hrany grafu.

Ještě si upravíme umístění vrcholů grafu pro přehlednost do kruhu funkcí `shell_layout` s parametrem `G`.

Výpis kódu 5.10: Tvorba grafu pomocí modulu NetworkX

```
G.add_nodes_from(RouterList)
        G.add_edges_from(PathList)
        pos=nx.shell_layout(G)

        # Check if all routers are reachable
        result = MainWindow.pathExist(G, RouterList)

        if result == False:
            return False

        nx.draw(G,pos,alpha=0.5,node_color='b', node_size=1000,
with_labels=True)
        # Save our graph to file graph.png
        plt.savefig("graph.png")
        # Clear our graph for next use of frawGraphFunc
        plt.clf()
```

Doposud jsme volali metodu pro tvorbu grafu, aniž bychom zkontrolovali, jestli nově vygenerovaný graf je souvislý, což v praxi znamená, že všechny směrovače mají alespoň jednu cestu ke všem ostatním směrovačům. To zajistíme metodou `pathExist`, které předáme parametry graf `G` a seznam vrcholů grafu `RouterList`. Tato metoda používá `for` cyklus ve `for` cyklu. První `for` vybírá popořadě vrcholy a uloží je do proměnné `router1`. Dále druhý cyklus udělá to samé a uloží vrchol do proměnné `router2` a porovná, jestli existuje cesta v grafu `G` z vrcholu `router1` do `router2`. Jednoduše řečeno se porovná každý vrchol s každým (včetně sebe sama), jestli mezi nimi existuje alespoň jedna cesta.

Samotné porovnávání zajišťuje metoda z NetworkX `has_path`. Tato metoda má návratovou hodnotu `True`, když cesta existuje, `False`, pokud mezi vrcholy zadanými jako parametry cesta není.

Výpis kódu 5.11: Kontrola spojitosti grafu

```
    def pathExist(G, RouterList):
        var = nx.nodes(G)

        for router1 in RouterList:
            for router2 in RouterList:
                pathexist = nx.has_path(G,router1,router2)
                if pathexist == False :
                    return False
        return True
```

Zpět ale do metody `drawGraphFunc`, ve které si uložíme návratovou hodnotu metody `pathExist` do proměnné `result` a `if` konstrukcí zkontrolujeme, jestli není rovna `False`. Pokud by byla podmínka splněna, tak ukončíme metodu `drawGraphFunc` s návratovou hodnotou `False`, jinak přejdeme k nastavování písma, barvy a velikosti vrcholů a podobně. Tyto vlastnosti nastavujeme až při samotném vykreslování metodou `draw`, které jako parametry zadáme náš graf `G`, kruhové zobrazení vrcholů uložené v proměnné `pos`, tloušťku linek `alpha`, barvu vrcholů `node_color` a jejich velikost `node_size` a nakonec povolíme zobrazení popisů vrcholů nastavením `with_labels` na hodnotu `True`. Posledním krokem je uložení vygenerovaného grafu ve formátu png.

5.8. VYPNUTÍ CESTY

Po zadání směrovačů, vygenerování a zobrazení grafu, si může uživatel konfigurovat síť pomocí MikrotikAPI. K tomu slouží `QlistWidget`, který tvoří seznam cest a tlačítko „Refresh Paths". Ovládání je jednoduché. Uživatel pouze označí kliknutím v seznamu cesty, které chce vypnout. Pokud na cestu klikne podruhé, označení se zruší. Neoznačené cesty se nezávisle na současném stavu aktivují. Po potvrzení tlačítkem „Refresh Paths" se odešle příkaz do směrovače (který byl přidán dříve) o nastavení hodnoty IP adresy `disabled` na `True`. Samotné vypnutí spočívá v zakázání IP adresy na daném směrovači, tudíž se nejedná o vypnutí rozhraní celého. To má své výhody hlavně při vývoji nebo pro bezpečnější používání aplikace, aby při nenadále chybě nedošlo ke špatné konfiguraci a následné odstřihnutí směrovače pro uživatele. Pokud by nastala chybná konfigurace a uživatel by zakázal všechny přístupové adresy, stále se může na směrovač přihlásit pomocí MAC adresy, například pomocí Telnetu.

Po stisknutí tlačítka „Refresh Paths" se zavolá metoda `disablePaths`. `For` cyklem projdeme všechny vybrané položky. Pro zjištění vybraných položek slouží funkce `selectedItems`, která vrací označené položky. Každou položku si uložíme do proměnné `path` a metodou `split` vezmeme její třetí slovo, kterým je stav „disabled" a uložíme si ho do proměnné `disabled`.

Dále konstrukcí `if` kontrolujeme, jestli je cesta aktivní (`disabled==false`) a je-li podmínka splněna, tak si do proměnných `router1` a `router2` uložíme jména směrovačů, které získáme opět funkcí `split` jako první a druhé slovo z proměnné `path`, obsahující celý text položky.

Připojíme se do databáze „RoutersList" a vybereme záznamy příkazem `SELECT` z tabulky `Paths`, které obsahují ve sloupcích „R1" a „R2" jména směrovačů uložených v proměnných `router1` a `router2`. Použijeme metodu `fetchall` a záznamy se shodou si uložíme do proměnné `row`. V proměnné `row` tak máme jak názvy směrovačů, tak i IP adresy,

kterými jsou připojeny. Jména směrovačů si uložíme do seznamu `disablePath` a adresy vložíme metodou `append` do seznamu `ipaddr`.

Výpis kódu 5.12: Výběr cest k vypnutí

```
# Get Names of routers
        for item in self.listWidgetPath.selectedItems():
            path = str(item.text())
            disabled = path.split()[2][1:-1]

            if disabled == 'false' :

                router1 = path.split()[0][1:-2]
                router2 = path.split()[1][1:-2]

                con = lite.connect('RoutersList.db')
                with con:
                    cur = con.cursor()
                    # Get ipaddress from path table - row which contain router1
and router2
                    cur.execute("SELECT * FROM Paths WHERE (R1 IS ? AND R2 IS
?) OR (R2 IS ? AND R1 IS ?)", (router1,router2,router1,router2))
                    row = cur.fetchone()
                    # Put R1 and R2 as an element in list of disabledPath, pass
this list as argument to draw graph
                    disabledPath.append([row[0],row[2]])

                    # Fill list with router1 name and his IP address which
should be disabled, pass this list to function sendDisableRouter
                    ipaddr.append(router1)
                    ipaddr.append(str(row[3]))
```

V další části si naopak uložíme ty cesty, které zůstanou nezměněny. `For` cyklem projdeme všechny položky z `QlistWidget` s cestami. Z každé položky si vezmeme třetí slovo a dáme si ho do proměnné `enabled`, které tak obsahuje „true" či „false" podle stavu cesty. Dále metodou `isSelected`, která vrací hodnotu `True`, pokud je položka vybrána nebo v opačném případě `False` a přidáme ještě druhou podmínku pomocí `and`, kterou kontrolujeme, jestli v proměnná `enabled` neobsahuje `false`.

Tudíž pokud položka nebyla vybrána a je aktivní (obsahuje „true"), tak si u této položky uložíme jména směrovačů do `router1` a `router2`. Opět se připojíme do databáze a najdeme záznamy v tabulce „Paths", které se týkají cesty spojující tyto dva směrovače (záznam obsahuje `router1` a `router2` ve sloupcích „R1" a „R2"). Metodou `fetchone` si uložíme záznam do `row` a z tohoto záznamu si uložíme jména směrovačů do seznamu `enabledPath` a jejich IP adresy do seznamu `ipaddrEnable`.

Pro shrnutí máme seznamy, kde `disabledPath` obsahuje elementy tvořené dvojicí jmen směrovačů jejichž cesta, která je spojuje má být vypnuta. Dále seznam `ipaddr` obsahující IP adresy na rozhraních spojující tyto směrovače. Seznam `enabledPath` obsahuje dvojice směrovačů, jejichž cesta je zapnutá a nemá být vypnuta a `ipaddrEnable` jsou IP adresy rozhraní spojující tyto směrovače.

Nyní zavoláme metodu pro vytvoření grafu a jako parametr vložíme celou proměnnou `disabledPath`. Pro vložení seznamu je nutné připsat parametru hvězdičku, která metodě

říká, že má brát parametr celý se všemi elementy a že je to struktura seznam. V opačném případě by byl jako parametr předán pouze první element. Pokud je možné graf sestavit aby byl spojitý, tak je návratová hodnota `True`.

Výpis kódu 5.13: Aktualizace záznamů v tabulce cest

```
result = MainWindow.drawGraphFunc(*disabledPath)
        if result == True :

            self.sendDisableToRouter("enable", *ipaddrEnable )
            self.sendDisableToRouter("disable", *ipaddr )

            # Set disabled in Paths table to True/False
            for index in xrange(self.listWidgetPath.count()):

                router = self.listWidgetPath.item(index).text()
                router1 = str(router).split()[0][1:-2]
                router2 = str(router).split()[1][1:-2]

                if self.listWidgetPath.item(index).isSelected() == False :
                    con = lite.connect('RoutersList.db')
                    with con:
                        cur = con.cursor()
                        cur.execute("UPDATE Paths set disabled=? WHERE (R1=?
AND R2=?)", ("false",str(router2),str(router1)))
                        con.commit()

                else :
                    con = lite.connect('RoutersList.db')
                    with con:
                        cur = con.cursor()
                    cur.execute("UPDATE Paths set disabled=? WHERE (R1=? AND
R2=?)", ("true",str(router2),str(router1)))
                    con.commit()
```

Pokud není návratová hodnota `True`, tak vyskočí okno s varováním, že některý ze směrovačů by byl nedostupný. Konečně můžeme přistoupit k samotnému zapínání a vypínání rozhraní.

Nyní zavoláme metodu `sendDisableToRouter`, která nastavuje hodnoty `Disabled` na `True` či `False` podle hodnoty parametru. Tuto metodu si popíšeme dále.

Pokračujeme nastavením sloupce „disabled" v tabulce „Paths". `For` cyklem projedeme opět všechny řádky seznamu cest. Pokud položka není vybraná, což zjistíme opět metodou `isSelected`, tak po připojení k databázi si v tabulce „Paths" najdeme cestu a nastavíme u ní sloupec „disabled" na `false`. Pokud položka vybrána je, tak ji v databázi nastavíme hodnotu na `true`, neboť veškeré nevybrané cesty se aktivují nezávisle na předchozím stavu.

Změny provádíme SQLite příkazem `UPDATE`, kde vybereme tabulku „Paths" a `set disabled` nastavíme požadovanou hodnotu u řádků splňující podmínku `WHERE`, která prochází obsah sloupců „R1" a „R2"a hledá názvy směrovačů. Nakonec změny potvrdíme a nahrajeme pomocí metody `commit`.

Posledním krokem je aktualizace seznamu cest na hlavním okně podle aktuálního stavu cest. K tomuto účelu použijeme `for` cyklus procházející jednotlivé elementy seznamu. Uložíme si obsah elementu jako text do proměnné `textBefore` a `if` konstrukcí zjistíme,

jestli byl prvek vybrán. Pokud ano, tak mu přepíšeme poslední slovo na `true`, v opačném případě na `false`.

Nyní si konečně zavoláme metodu pro vygenerování grafu `drawFunc`, která nám vytvoří obrázek v adresáři s aplikací. Odtud si obrázek načteme pomocí metody `QPixmap`, která se používá pro nahrání do PyQt GUI objektů a vložíme do našeho hlavního okna jako obsah `QlabelWidget`.

5.9. CHYBOVÉ HLÁŠKY

Již jsme se párkrát zmínili o zobrazení chybové hlášky. Nyní si popíšeme, čím jsou tvořeny a jak se spouští. V PyQt jsme si vytvořili jednoduché `QWidget` okno s tlačítkem „OK" a štítek `Qlabel`, do kterého vkládáme varovnou větu. Přeložili jsme si „xml" kód do python kódu a importovali ho do aplikace jako modul, stejným způsobem jako předchozí okna.

Výpis kódu 5.14: Třída a inicializační funkce pro chybové hlášky

```
class ErrorWindow(QtGui.QDialog,Ui_ErrorWindow):
    def __init__(self, parent=None):
        super(ErrorWindow, self).__init__(parent)
        self.setupUi(self)
        self.errorButton.clicked.connect(self.PushErrorButton)
```

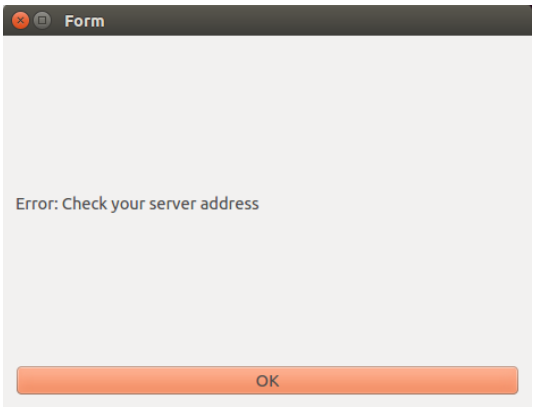
Rozdíl tohoto okna oproti ostatním je, že není tvořeno jako hlavní okno, nýbrž jako zjednodušené dialogové okno.

Chybové hlášky obsahují pouze jediný ovládací prvek, a tím je tlačítko „OK", jehož stisknutím se zavolá metoda `PushErrorButton`. Tato metoda pouze okno zavírá.

Výpis kódu 5.15: Otevření chybové hlášky

```
    def ErorrMsg(self, errorText):
        window = ErrorWindow(self)
        window.ErrorLabel.setText(errorText)
        window.show()
```

Použití je jednoduché. Každá třída má svoji `ErrorMsg` metodu, která inicializuje `ErrorWindow`, nastaví text štítku `Qlabel` pomocí metody `setText` a nakonec zavolá metodu `show`, která okno zobrazí.



Obr 5.3: Chybová hláška

5.10. MIKROTIKAPI

Pro konfigurování směrovačů a zjišťování stavu sítě, jsme si vytvořili API na základě příkladu klienta a doporučení uvedených na webových stránkách Mikrotiku [5]. Celé API je tvořené třídou `ApiRos` , která obsahuje metodu `login`, která slouží pro přihlášení do směrovače. Ta se spouští se třemi parametry, a to proměnnou `self`, přihlašovacím jménem `username` a heslem `pwd` .

5.10.1. PŘIHLÁŠENÍ

Nejprve ve `for` cyklu zavoláme metodu `talk`, které jako parametr zadáme `/login`. Tímto pošleme směrovači takzvanou „výzvu", neboli žádost o přihlášení. Pokud se vše odeslalo v pořádku, tak směrovač odpoví `!done` a `=ret=` spolu s MD5 hashem.

Proto v metodě `login` využíváme kryptovacího modulu md5. Vytvoříme si instanci `md`, kterou dále používáme a vkládáme do ní řetězce pomocí metody `update`.

Dále v metodě `login` voláme opět metodu `talk` a zadáme jako parametr `worlds` obsahující řetězec `/login`, dále `=name=` s hodnotou `username` a `=response=`, do kterého je vložen „hash" (kontrolní součet) složený z hesla a přijatého hashe od směrovače. Hash ale nejdříve překonvertujeme do hexadecimálního tvaru pomocí metody `digest`, abychom ho mohli vložit do metody `hexlify`, která navrátí binární podobu hexadecimálního řetězce. V případě navrácení hodnoty `False` metodou `talk`, metodu `login` ukončíme.

Výpis kódu 5.16: Přihlášení do směrovače pomocí MikrotikAPI

```
def login(self, username, pwd):
        for repl, attrs in self.talk(["/login"]):
            chal = binascii.unhexlify(attrs['=ret'])
        md = md5.new()
        md.update('\x00')
        md.update(pwd)
        md.update(chal)
        self.talk(["/login", "=name=" + username,
                   "=response=00" + binascii.hexlify(md.digest())])
```

Metoda `talk` slouží k celistvému náhledu na komunikaci a pracuje již z celými zprávami. Metodou `talk` voláme metodu `writeSentence` sloužící k posílání vět a `if` konstrukcí opět kontrolujeme návratovou hodnotu, která je rovna počtu odeslaných slov ve větě. Pokud je rovna nule, ukončíme metodu `talk`. Ve `while` smyčce voláme metodu `readSentece`, která slouží k načítání odpovědí a ukládáme si návratovou hodnotu tvořenou odpovědí do proměnné `i`. Pokud `i` obsahuje řetězec znaků `=message=cannot log in`, tak metodu ukončíme s návratovou hodnotou `False`. První znak `i` je uložen do proměnné `reply` a ve `for` cyklu procházíme znaky `i` od druhého znaku a hledáme pozici rovnítka, kterou si uložíme do proměnné `j`. Dále si uložíme část odpovědi za rovnítkem do `attrs`. Pomocí `append` přidáme `reply` i `attrs` do seznamu `r`.

Nakonec kontrolujeme, jestli v `reply` není řetězec `!done`, pokud ano, značí to konec přihlašování a můžeme ukončit metodu `talk` a jako návratová hodnota slouží seznam `r`, která obsahuje odpověď směrovače.

Výpis kódu 5.17: Metoda pro odesílání a přijímaní zpráv

```
def talk(self, words):
        if self.writeSentence(words) == 0: return
        r = []
        while 1:
            i = self.readSentence();
            if "=message=cannot log in" in i: return False
            if len(i) == 0: continue
            reply = i[0]
            attrs = {}
            for w in i[1:]:
                j = w.find('=', 1)
                if (j == -1):
                    attrs[w] = ''
                else:
                    attrs[w[:j]] = w[j+1:]
            r.append((reply, attrs))
            if reply == '!done': return r
```

5.10.2. ODESÍLÁNÍ ZPRÁV

Po přihlášení přejdeme k odesílání příkazů. Zavoláme metodu `writeSentence`, která slouží k rozčlenění vět představujících celé příkazy na jednotlivá slova. Metoda je volaná s parametrem `words`, což je datová struktura seznam obsahující slova tvořící příkaz neboli větu. `For` cyklem procházíme všechny elementy, tedy jednotlivá slova v seznamu `words` a voláme metodu `writeWord`, které jako parametr `w` zadáme právě jedno slovo.

Metoda `writeWord` pracuje s parametrem `w`, ve kterém je obsažené slovo předané metodou `writeSentence`. Nejprve je zavolána metoda `writeLen` s parametrem délka slova `w` a poté je zavolána metoda `writeStr` pro odeslání slova.

Metoda `writeLen` slouží k odeslání délky slova a volá metodu `writeStr`. Metoda využívá metody `chr`, která vrací řetězec o délce jednoho znaku podle ASCII hodnoty parametru zadané jako integer. `If` konstrukcí jsou zprávy rozděleny do čtyř typů podle velikosti. Do prvního typu spadají zprávy o délce do 128 b. Další skupina jsou zprávy o délce do 16 kB, a tak dále. Je to proto, že delší zprávy jsou rozděleny na více částí. [29]

Samotná metoda `writeStr` slouží k odesílání řetězců pomocí metody `send` modulu `socket`. Nakonec má `writeStr` ještě kontrolu, jestli se podařilo řetězec odeslat pomocí `if` konstrukce, která ověřuje návratový kód metody `send`.

5.10.3. PŘIJÍMÁNÍ ZPRÁV

Pro získání odpovědi od směrovače zavoláme metodu `readSentence` a její návratovou hodnotu si uložíme do proměnné. Metoda deklaruje seznam, do kterého bude vkládat jednotlivá slova získaná jako výstupní hodnoty metody `readWord`, kterou volá v smyčce `while`. Pokud je přijaté slovo prázdný řetězec, metoda se ukončí. Pokud ale není přijaté slovo prázdné, přidá se metodou `append` do seznamu `r`.

Metoda `writeWord` volá metodu `readStr` s parametrem délky přijatého slova, kterou získá metodou `readLen`. Výsledný řetězec uloží do proměnné `ret` a použije jej jako návratovou hodnotu pro metodu `readSentence`.

Další metoda `readLen` je pro zjištění délky přijímaného řetězce a funguje na opačném principu než metoda `writeLen`. Metoda `readLen` používá metodu `ord`, která vrací `integer` reprezentující „Unicode" hodnotu znaku. Zavoláme metodu `readStr` s

parametrem „1", což znamená, že přijmeme jeden byte. Z binární hodnoty získáme metodou `ord integer` s velikostí zprávy. Dále `if` konstrukcemi rozdělujeme zprávy do několika skupin podle velikosti. Každá skupina opakovaně volá metodu `readStr` s parametrem 1 B podle velikosti zprávy, která obsahuje informace o délce následující zprávy.

Výpis kódu 5.18: Přijímání zpráv o velikosti dat

```
def readLen(self):
        c = ord(self.readStr(1))
        if (c & 0x80) == 0x00:
            pass
        elif (c & 0xC0) == 0x80:
            c &= ~0xC0
            c <<= 8
            c += ord(self.readStr(1))
        elif (c & 0xE0) == 0xC0:
            c &= ~0xE0
            c <<= 8
            c += ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
        elif (c & 0xF0) == 0xE0:
            c &= ~0xF0
            c <<= 8
            c += ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
        elif (c & 0xF8) == 0xF0:
            c = ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
            c <<= 8
            c += ord(self.readStr(1))
        return c
```

Poslední metodou je metoda `readStr`, která je volána s parametrem `length` a ve smyčce `while` přijímá pomocí metody `recv` modulu `socket` data, dokud je jejich délka menší než zadaný parametr `length`. Metoda navrací řetězec `ret,` do kterého se vkládají přijatá data.

5.11. ECHO SERVER

Doposud jsme se bavili o aplikaci s aktivním chováním, která při testování odesílá data. Nyní si probereme druhou část, která je tvořena pasivním programem, který slouží jako „echo server", to znamená, že data která obdrží pošle zpět. Nedochází zde k žádným výpočtům či vykreslování grafů, neboť jsou zajištěny druhou aktivní částí.

5.11.1. OKNO SERVERU

Okno serverové části má tři základní oblasti. První a největší z nich je informační okno slouží podobně jako konzole, ve kterém se zobrazují výpisy událostí, ať už přijímání a odesílání dat,

navázání schránky, chybové hlášky a podobně. Toto okno je tvořeno objektem `QTextEdit`, což je textové pole s možností úprav obsahu.

V další části uživatel zadává data pro síťovou schránku, a to IP adresu, na které bude server poslouchat, port na kterém naslouchá a nakonec port očekávaného spojení, kterým se k němu klient připojí.

V poslední části jsou jednoduchá dvě tlačítka. První z nich je tlačítko „Start", kterým se spustí schránka v módu „listen" a tlačítko „Stop", jehož funkcí je ukončit spojení a smazat síťovou schránku.

Obr 5.4: Okno echo serveru

Okno bylo vytvořeno opět pomocí PyQt a přeloženo do Pythonu. Takto získaný kód jsme použili jako modul pro aplikaci.

5.11.2. POSÍLÁNÍ DAT

Po stisknutí tlačítka „Start" se zavolá metoda `StartListen`, které předáme informace o spojení z textových polí `lineEdit` na hlavním okně aplikace. Vytvoří se instance `WorkThread` a zavoláme její metodu `start`.

`WorkThread` je třída, pomocí které vytváříme nové vlákno programu. To je nezbytné, abychom mohli mít spuštěnou schránku v módu naslouchat (listen) a zároveň měli k dispozici ovládání našeho GUI. Například abychom mohli stisknout tlačítko „Stop" v době, kdy náš server přijímá nebo očekává data. To nám umožňuje PyQt, kterým vytváříme vlákna jako instance třídy `QThread`.

Výpis kódu 5.19: Spuštění odesílacího vlákna a zachytávání signálů

```
    def startListen(self):
        # Get HOST address and port from lineEdit
        HOST = self.lineEdit.text()
        PORT = self.lineEdit_2.text()
        ClientPORT = self.lineEdit_3.text()
        # Create WorkThread
        self.WorkThread = WorkThread(HOST, PORT, ClientPORT)
        # Start WorkThread thread
        self.WorkThread.start()
        # SIGNAL sending text for TextEdit creating out log window
        self.connect(self.WorkThread, QtCore.SIGNAL('Text_Line_Update'),
self.updateTextEdit)
        self.connect(self.WorkThread,
QtCore.SIGNAL('Text_Line_Sending_Update'), self.updateTextEdit)
```

Pro komunikaci mezi vlákny na hlavním vlákně GUI ještě musíme předpřipravit zachytávání signálů a vytvořit metodu pro definování chování při zachycení signálu. Z vlákna `WorkThread` očekáváme dva druhy signálů, prvním z nich je `Text_line_Update` a druhým `Text_line_Sending_Update`, po jejichž zachycení zavoláme metodu u`pdateTextEdit`.

Tato metoda slouží k vypisování textu, který metoda příjme jako argument do textového pole na hlavním okně programu `textEdit`. K vypisování používáme metodu `append` , která přidává argument `text` za již vypsaný text a textové pole tak má chování a funkci konzole, kde uživatel vidí informace o daném stavu schránky, o přijímaných a odesílaných datech, popřípadě o chybách, které nastaly.

5.11.3. VLÁKNO PRO ODESÍLÁNÍ

Vlákno sloužící ke spuštění schránky posílající data zpět nazýváme `WorkThread`. Při vytváření jeho instance je volána inicializační metoda, která příjme parametry o portech a zavolá se metoda `run`. V metodě `run` vytvoříme instanci vlákna pro přijímání dat a vlákno spustíme. Dále si nadefinujeme zachytávání signálů z vlákna pro zasílání dat do naslouchacího vlákna.

Zachytáváme tři různé signály. Prvním z nich je `Text_line_Update` signál, po jehož zachycení voláme metodu `writeToLog`. Tato metoda pouze přeposílá text, který má být zobrazen v textovém poli hlavnímu vláknu.

Druhým signálem je `IP_address`, kterým zachytáváme IP adresu. Tu nám pošle naslouchací vlákno při přijmutí dat. Po zachycení IP adresy zavoláme metodu `getIP`. Tato metoda zapíše přijatý argument do proměnné `ip`.

Posledním signálem, který zachytáváme je signál `Text_line_Send`. Po zachycení tohoto signálu zavoláme metodu `sendBack`. Tato metoda slouží k vytváření schránek a posílání dat klientovi. Nejprve zkontrolujeme, jestli proměnná `ip` není prázdná a poté pomocí `try` bloku zkontrolujeme, jestli už máme vytvořenou síťovou schránku `sock`. Pokud schránka není ještě vytvořena, tak ji v `except` bloku vytvoříme.

Schránku `sock` vytvoříme tak, aby očekávala IPv4 adresu zadáním parametru `AF_INET`, a také specifikujeme typ schránky na `SOCK_STREAM`, což je nejběžněji používaný typ schránek umožňující obousměrné spojení tvořené tokem bytů. Poté schránku připojíme k zadané IP adrese uložené v proměnné `ip`, kterou jsme zachytili signálem pro zachytávání IP

adres a specifikujeme port, na který se má připojit. Číslo portu jsme získali už v hlavním okně a předali jako argument při vytváření vlákna.

Nyní máme přichystanou schránku pro posílání dat, tudíž se můžeme věnovat datům. Nejprve zjistíme jejich délku metodou `len`. Tu zabalíme pomocí metody `pack` z modulu `struct`, kterým data o délce naformátujeme a metodou `sendall` pošleme klientovi.

Déle pošleme samotná `data` opět za použití metody `sendall`. Posílání délky i samotných dat je umístěno v `try` bloku a při chybě upozorníme uživatele chybovou hláškou, že nemohla být data poslána a zobrazíme je v textovém poli hlavního okna serverové aplikace.

Nakonec vypíšeme odeslaná data do textového pole. Tím může uživatel sledovat, co přesně přijímá a odesílá.

Výpis kódu 5.20: Vytváření schránky a odesílání dat

```
try:
                WorkThread.sock
            except:
                WorkThread.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                server_address = (WorkThread.ip, int(self.ClientPORT))
                WorkThread.sock.connect(server_address)

            if WorkThread.sock==None:
                WorkThread.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                server_address = (WorkThread.ip, int(self.ClientPORT))
                WorkThread.sock.connect(server_address)

            # Get length of message
            length = len(text)
            # Pack length of message and send
            try:
                WorkThread.sock.sendall(struct.pack('!I', length))
                WorkThread.sock.sendall(text)
            except:
                self.writeToLog('Can not send message back')

            self.writeToLog('<<< %s' % text)
```

5.11.4. VLÁKNO PRO NASLOUCHÁNÍ

V metodě `run` u vlákna pro posílání dat zpět klientovy jsme si vytvořili instanci třídy `SendingWorkingThread` jako třetího vlákno, které slouží pro přijímání dat od klienta a vysílání signálu posílajícího vlákna. Naslouchajícímu vláknu byly přidány parametry portu a IP adresy, na kterých má naslouchat příchozí komunikaci.

Po spuštění vlákna je zavolána metoda `run`. V té upozorníme uživatele o vytvoření schránky. Zavoláme metodu `writeToLog`, které pošle signál o vypsání textu do textového pole. Dále vytvoříme samotnou síťovou schránku stejným způsobem jako u předchozího vlákna metodou `socket` s parametry `socket.AF_INET` pro používání IPv4 adresy a `socket.SOCK_STREAM` pro vytvoření schránky pro TCP spojení.

Dále metodou `writeToLog` upozorníme uživatele o vytvoření schránky připojenou na danou IP adresu a port. Zavoláme metodu `setsockopt` s parametrem `socket.SO_REUSEADDR`, pro možnost použít schránku znovu. To slouží hlavně když

uživatel použije tlačítko „STOP" a poté chce naslouchat na stejné IP adrese a portu. BSD schránku spojíme metodou `bind` k portu a IP adrese, na kterých bude naslouchat. Číslo portu i IP adresa byly předány jako parametry při vytváření vlákna. To provedeme v `try` bloku a v případě chyby upozorníme uživatele chybovou hláškou.

Nyní přepneme schránku do naslouchacího režimu metodou `listen`. Dále ve `while` smyčce čekáme na připojené od klienta. Když dorazí data, tak připojení metodou `accept` přijmeme a uložíme si ho do proměnné `connection`. Zkontrolujeme, jestli se nepřipojuje serverová aplikace, neboť to by znamenalo, že se potřebuje ukončit schránka. Pokud IP adresa není serverové aplikace, tak informujeme uživatele o přijetí spojení od klienta a vypíšeme jeho adresu.

Dále přistoupíme k přijímání zprávy o délce dat, a to tak, že ve `while` smyčce voláme metodu `recv` s parametrem `length`. Metoda `recv` slouží k přijímání dat pomocí schránky a je volána s parametrem `bufsize`, což je velikost očekávaných dat a vrací velikost přijatých dat. V proměnné `length` máme uloženou velikost dat, která je nastavená na 4 byty. To je pro nás víc než dostatečně velká délka zprávy o velikosti nadcházejících dat.

Po přijetí délky nadcházejících dat zavoláme metodu `sendIP`, kterou pošleme signál vláknu pro posílání dat s IP adresou klienta a rozbalíme si zprávu o délce dat pomocí metody `unpack` z modulu `struct` a naformátujeme si délku do Python řetězce. Takto získanou délku si uložíme do proměnné `lengthOfMessage`.

Zanoříme se do další smyčky `while`, kterou voláme metodu `recv`, dokud nepřijmeme data o velikosti l`engthOfMessage` a nastavíme ji parametr `bufsize` na délku `lengthOfMessage`. Takto získaná data uložíme do proměnné `newdata` a pokud není proměnná `newdata` prázdná, pak přidáme `newdata` do `data` a zmenšíme `lengthOfMessage` o délku již přijaté části.

Zkontrolujeme, jestli proměnná data nejsou prázdná a zavoláme metodu `sendBack` právě s parametrem `data`. Metoda `sendBack` předá signál vláknu pro odesílání, že má poslat data zpět, ale nejprve je ořízneme na prvních 26 znaků, což jako potvrzení klientovy stačí a zároveň tato délka umožňuje přijímat celé časové značky, tudíž zbytek zpráv, který je oříznut obsahuje pouze vyplňující mezery pro nastavení délky dat.

Výpis kódu 5.21: Přijímání dat

```
            try:
               # Receive the data and retransmit it back
               while True:
                   # Get first part of message with length of message
                   getlength = connection.recv(length)
                   # From client_address get pure IP address
                   self.sendIP(client_address[0])

                   # Unpack and get length
                   lengthOfMessage, = struct.unpack('!I', getlength)
                   # Receive rest of the message
                   data= b''
                   while lengthOfMessage:
                       newdata = connection.recv(lengthOfMessage)
                       if not newdata: break
                       data += newdata
                       lengthOfMessage -=len(newdata)
```

```
                    #If data aren't empty
                    if data and not data == "" :
                        # Send message back
                        self.sendBack(data[0:26])
                        # Write info about sending to log
                        self.writeToLog('>>> "%s"' % data)

            except:
                self.writeToLog('no more data from %s ' % str(client_address))
```

Pokud již nejsou přijímána žádná data, upozorníme uživatele zprávou, že již nepřijímáme data od klienta zavoláním metody `writeToLog`. Nyní síťovou schránku pro posílání dat ukončíme metodou `close` a schránku smažeme.

To nás vede k původní `while` smyčce, kde opět čekáme na přijetí dat. Tuto smyčku lze ukončit nastavením proměnné `keepRunning` na `False`, to zajišťuje tlačítko „Stop".

5.11.5. UKONČENÍ NASLOUCHÁNÍ

Pokud uživatel již nechce naslouchat na daném portu a adrese, stačí mu pouze stisknout tlačítko „Stop". To způsobí ukončení síťové schránky ve vlákně pro naslouchání.

Po stisknutí tlačítka „Stop" se zavolá metoda `stopListen`. Zavoláme metodu vlákna pro odesílání `stop`, která zavolá metodu `stop` ve vlákně pro přijímání. Ta se připojí na síťovou schránku čekající na data nebo přijímající data, ve které je kontrola, jestli se nepřipojila samotná aplikace. Jakmile rozezná vlastní IP adresu, ukončí se `while` smyčka a metodou `close` se ukončí síťová schránka.

Nakonec zavoláme metodu `writeToLog` a upozorníme uživatele hláškou, že síťová schránka byla uzavřena.

5.12. KONZOLOVÝ ECHO SERVER

Pro testování sítě můžete mít spuštěno více serverů, aby každý nepotřeboval další počítač, stačí nějaké zařízení s operačním systému Linux s nainstalovaným Pythonem a použitými moduly (například Qt, které používáme pro tvorbu vláken). Takovým zařízením může být dnes populární Raspberry Pi.

5.12.1. PŘEPOSÍLÁNÍ DAT

Konzolová verze je oproti verzi s uživatelským rozhraní zjednodušena, neboť nepotřebuje vlákno pro GUI, moduly s kódem pro tvorbu oken a podobně.

Při spuštění si vytvoříme instanci `WorkThread` třídy, která převzala své jméno po vláknu pro přeposílání dat. Při její inicializaci spustíme `run` funkci, která spustí vlákno pro naslouchání a připravíme si odchytávání signálů `IP_address`, kde očekáváme IP adresu klienta a `Text_Line_Send`, kde zachytáváme signál s přijatými daty, které předáme jako parametr metodě `sendBack`.

Metoda `sendBack` zůstává stejná, jako metoda `sendBack` u klasické GUI verze serveru. Zkontroluje, jestli má IP adresu a jestli je vytvořená síťová schránka, pokud není, tak ji vytvoří metodou `socket` z modulu `socket` s parametry `AF_INET` pro práci s IPv4 adresami a `SOCK_STREAM`, kde schránka tvoří tok bytů pro TCP komunikaci.

Nakonec připojíme schránku k dané adrese a portu, které jsme získali z parametrů, se kterými jsme server spustili.

Nyní si zjistíme délku dat, které chceme poslat zpět metodou `len` a metodou `pack`, délku zabalíme a překonvertujeme pomocí modulu `struct`. Takto připravená binární data pošleme metodou `sendall` klientovi.

Po odeslání délky nadcházející zprávy pošleme zprávu samotnou, opět metodou `sendall`. Pokud se vyskytla chyba při posílání, vypíšeme chybovou zprávu. V opačném případě vypíšeme informaci o poslání dat. Výpis informací je tu za použití příkazu `print`.

5.12.2. PŘIJÍMÁNÍ DAT

Pro přijímání dat jsme si vytvořili vlastní vlákno, kterému jako parametry předáme adresu a port kde má naslouchat. Při spuštění zavoláme metodu `run`. V ní vytvoříme síťovou schránku `sock` metodou `socket` a vypíšeme informace o jejím spuštění. Dále ji přiřadíme adresu a port metodou `bind` a přepneme schránku do naslouchacího módu metodou `listen`.

Dále ve smyčce `while` spouštíme metodu `accept`, které čeká na příchozí spojení a uloží číslo portu a adresu klienta do proměnné `connection` a `client_address`, ze které získáme IP adresu a pošleme ji metodou `sendIP` hlavnímu vláknu metodou `sendIP`.

Ve smyčce `while` spouštíme metodu `recv`, která přijímá informace o délce následné zprávy a ukládá ji do proměnné `getlength` a tu předáme metodě `unpack` z modulu `struct` pro rozbalení. Rozbalenou délku si uložíme do `lengthOfMessage`.

V další smyčce voláme metodu `recv` s parametrem `lengthOfMessage` a přijatá data ukládáme do `newdata`. Pokud `newdata` nejsou prázdná, tak rozšíříme proměnnou `data` o `newdata` a z proměnné `lengthOfMessage` odečteme délku přijaté zprávy.

Nakonec data předáme metodě `sendBack`, které pošle signál hlavnímu vláknu. To přepošle data zpět, opět oříznuta na prvních 26 bytů a vypíšeme příkazem `print` informace o přijetí dat.

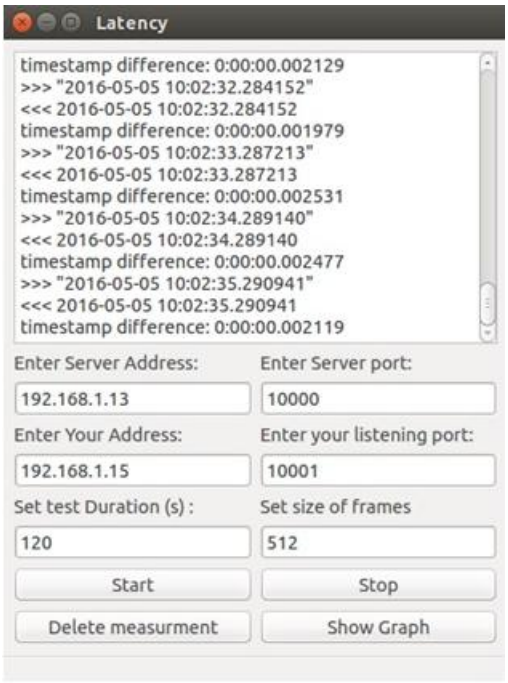
5.13. TEST ZPOŽDĚNÍ

Prvním testem, který jsme se rozhodli zařadit do naší testovací aplikace, je test zpoždění. Základním princip spočívá v uložení časové značky jako řetězce znaků a odeslání klientem na server. Server po přijetí časové značky pošle přijatá data zpět na stranu klienta, kde porovnáme časovou značku obsaženou v přijatých datech se systémovým časem. Rozdíl časů zahrnuje odeslání i přijetí dat, čímž získáme hodnotu obousměrného zpoždění.

5.13.1. OKNO TESTU ZPOŽDĚNÍ

Okno testu spustíme kliknutím na tlačítko „Latency" umístěné na dolní části hlavního okna programu. Okno testu obsahuje textové pole sloužící jako informační okno a textové pole pro zadání adresy serveru, vlastní adresy klienta, přes kterou chceme data odesílat, čísla portu serveru a číslo portu, přes který budeme data posílat. Porty mají předvyplněné hodnoty pro příjemnější ovládání. Dále je nastavení doby trvání testu a nastavení délky odesílaných dat.

Nakonec tu máme tlačítka. První dvě „Start" a „Stop" slouží ke spuštění testu s aktuálním nastavením parametrů. Tlačítko „Delete measurment" vymazává údaje o předchozích měření a tlačítko „Show Graph" zobrazí graf všech měření zpoždění v databázi jako graf závislosti velikosti odesílaných dat na délce zpoždění. Po každém testu je uživateli zobrazen jednoduchý graf časového průběhu zpoždění.



Obr 5.5: Okno testu zpoždění

5.13.2. SPUŠTĚNÍ TESTU

Po stisknutí tlačítka „Start" je odeslán signál, po jehož zachycení je zavolána metoda `pushStart`. V ní nejprve vyčteme informace z textových oken metodou text a uložíme si informace do proměnných, které zkontrolujeme, jestli jejich obsah odpovídá předpokládaným datům.

Nejprve konstrukcí `if` zkontrolujeme IP adresu server za použití metody `findall` hledající v proměnné regulární výraz `r'[0-9]+(?:\.[0-9]+){3}/[0-9]{1,2}'`. Pokud není nalezena IP adresa, upozorníme uživatele chybovou hláškou a metodu `pushStart` ukončíme. Stejným způsobem kontrolujeme i vlastní adresu klienta.

Dále kontrolujeme číslo serverového portu. K tomu použijeme `try` blok, kdy se snažíme k hodnotě portu přičíst jedničku. Pokud by uživatel do textového pole nedal číslo, součet by se nemohl provést a vyskytla by se chyba. Proto v `except` bloku spouštíme chybovou hláškou a ukončujeme metodu `pushStart`. Stejným způsobem kontrolujeme vlastní port, dobu trvání testu i velikost dat.

Dále se přihlásíme do databáze a pokusíme se smazat předchozí měření SQLite příkazem `DROP TABLE IF EXISTS` a poté tabulku znovu vytvoříme příkazem `CREATE TABLE`. Tabulku, do které budeme zapisovat, si nazveme „Latency". Tabulka bude obsahovat čísla řádků ve sloupci „rowid" a jednotlivé hodnoty ve sloupci „Latency".

Konečně si vytvoříme vlákno pro posílání dat, které si nazveme `WorkThread` a předáme mu parametry čísla portů a IP adresy. Vlákno spustíme metodou `start` a připravíme si zachytávání signálů `Text_Line_Update` a `Time_Line_Update`. První zmíněný signál po zachycení zavolá metodu `updateTextEdit`, která slouží k aktualizování informačního textového pole. K tomu používáme metodu `append`, kterou aktualizujeme zobrazovaný text a také kontrolujeme, jestli signál naobsahuje řetězec `drawing graph`. Signál s tímto řetězcem nám zavolá metodu `drawLatencyGraph`, která slouží k

vygenerování grafu časového průběhu zpoždění a zobrazení grafu v okně vytvořeném pomocí modulu Matplotlib.

Po zachycení signálu `Time_Line_Update` zavoláme metodu `updateTime`. Signál obsahuje velikost zpoždění v proměnné `time` ve tvaru časové značky.

Jelikož nás zajímá velikost zpoždění v milisekundách, musíme si vypočítané zpoždění, které je nyní ve formátu `%Y-%m-%d %H:%M:%S.%f` (roky-měsíce-den hodiny:minuty:vteřiny:mikrosekundy) převést na milisekundy. Výsledné zpoždění v milisekundách si vypíšeme v hlavním okně.

Poté se připojíme do databáze „Measurment" s naměřenými daty a vložíme časový údaj o zpoždění ve tvaru milisekund do tabulky „Latency".

Výpis kódu 5.22: Převod času na milisekundy a uložení hodnoty do databáze

```
    def updateTime(self, time):

        difTime2 = str(time)[-14:]
        # Print difference to textEdit
        self.textEdit.append(("timestamp difference: "+str(difTime2)))
        ms = (time.days * 24 * 60 * 60 + time.seconds) * 1000 +
time.microseconds / 1000.0
        con = lite.connect('Measurment.db')

        with con:
            cur = con.cursor()
            cur.execute("INSERT INTO Latency VALUES(?)", (str (ms),))
```

5.13.3. VLÁKNO PRO ODESÍLÁNÍ DAT

Po stisknutí tlačítka „Start" jsme si v metodě `pushStart` vytvořili instanci třídy `WorkThread` a zavolali metodu `start`, kterou jsme vlákno spustili. Při samotné inicializaci jsme mu předali jako parametry čísla portů, IP adresy, délku testu a velikost odesílaných dat.

Při spuštění se zavolá metoda `run`, která vytvoří třetí vlákno pro přijímání potvrzení a připravíme si odchytávání signálů `Text_Line_Update` a `Time_Line_update`. První ze signálů po zachycení zavolá metodu `writeToLog` pro zobrazení informací v textovém poli a druhý signál zavolá metodu `sendTime`. Metoda `sendTime` pouze předá signál hlavnímu vláknu, kde zavolá metodu `updateTime` popsanou výše.

Také si vytvoříme síťovou schránku, kterou použijeme pro posílání dat. Jako data nám poslouží časové značky získané pomocí modulu `datetime` příkazem `datetime.datetime.now()`, jehož výstup je systémový čas s přesností na mikrosekundy, což je nejpřesnější metoda použitelná pro Python na získání systémového času. Značka obsahuje kromě času i datum a celkově tak má délku 26 znaků.

V hlavním okně jsme zadali velikost posílaných dat, jelikož má časové značka 26 bytů, upravujeme délku zprávy metodou `format`, které předáváme parametry `data`, což je naše časová značka a požadovanou délku zprávy. Tato metoda doplní za zprávu mezery.

Poté si zjistíme délku požadované zprávy metodou `len` a zabalíme data o délce metodou `pack` z modulu `struct`. Takto zabalená data o délce nadcházející zprávy pošleme metodou `sendall`. Nyní již můžeme poslat samotná data s tím, že server zná přesnou délku dat, čímž se vyhneme nežádaného spojování a rozdělování zpráv, neboť BSD schránku máme

v módu „stream" a naše zprávy jsou tok bitů a nikoliv zpráv jako takových. Po poslání informujeme výpisem do textového pole a vteřinu počkáme použitím metody `sleep`.

Celé vygenerování časové značky a poslání serveru je ve `for` cyklu, jehož tělo opakujeme podle zadaného časového údaje.

5.13.4. VLÁKNO PRO PŘIJÍMÁNÍ DAT

V `run` funkci vlákna pro odesílání dat jsme si spustili další vlákno pro příjem potvrzení. Po spuštění zavoláme jeho metodu `run`, ve které si vytvoříme BSD schránku. Tu připravíme na používání IPv4 a přepneme do módu `SOCK_STREAM` představující TCP spojení. Metodou `setsockopt` umožníme znovupoužití adresy a metodou `bind` přiřadíme adresu a port zadané v hlavním okně. Nakonec přepneme schránku do naslouchacího režimu metodou `listen`.

Příjem dat je stejný jako příjem dat u serveru. Metodou `accept` vyčkáváme na příchozí spojení. Nejdříve očekáváme zprávu o délce nadcházejících dat metodou `recv`. Zprávu s délkou rozbalíme metodou `unpack` a získanou velikost zadáme jako parametr metodě `recv`.

Po přijetí celé zprávy si nejprve zjistíme systémový čas, poté vypíšeme obsah zprávy v hlavním okně. Nyní si musíme převést přijatá data zpět na časovou značku a přejdeme k výpočtu zpoždění. Samotný výpočet je pouhý rozdíl časové značky přijaté a té, kterou jsme si vygenerovali po přijetí. Výsledek pošleme hlavnímu vláknu na další zpracování metodou `sendTime` a ukončíme spojení.

Výpis kódu 5.23: Příjem dat a výpočet zpoždění

```
            while True:
                   try:
                       getlength = connection.recv(4)
                       # Unpack and get length
                       lengthOfMessage, = struct.unpack('!I', getlength)

                   except:
                       break

                   # Receive rest of the message
                   data= b''
                   while lengthOfMessage:
                       newdata = connection.recv(lengthOfMessage)
                       if not newdata: break
                       data += newdata
                       lengthOfMessage -=len(newdata)

                   if data and not data == "":
                       time2 = datetime.datetime.now()
                       self.writeToLog('<<< %s' % data)

                       time1 = datetime.datetime.strptime(data[0:26], "%Y-%m-
                               %d %H:%M:%S.%f")

                       time = time2 - time1
                       self.sendTime(time)
```

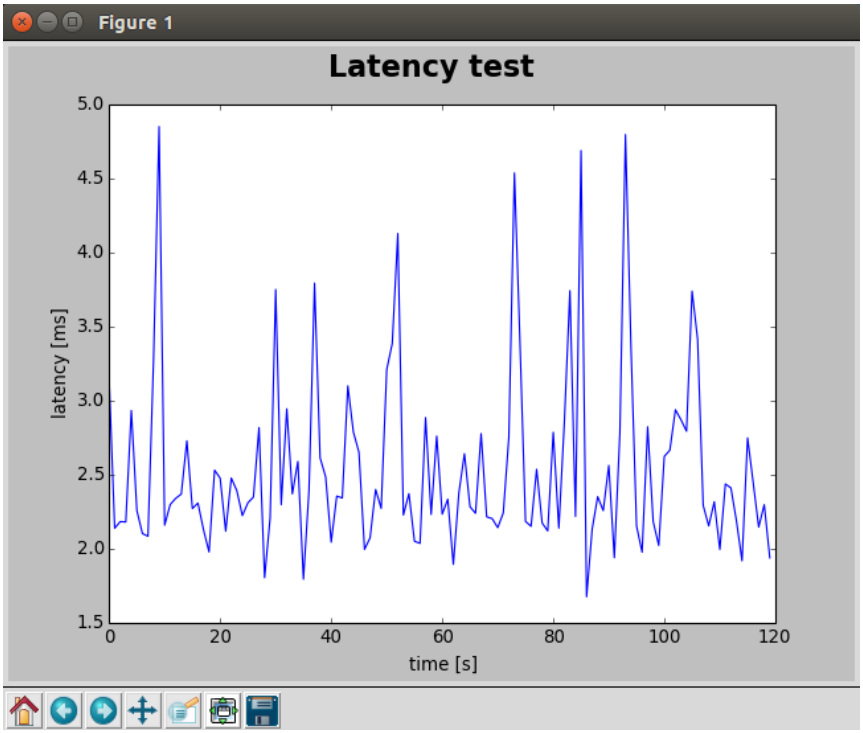
5.13.5. *VYKRESLENÍ GRAFU ZPOŽDĚNÍ*

Po ukončení spojení vlákno pro odesílání dat odešle signál hlavnímu vláknu, že může vykreslit graf zpoždění. Tento graf znázorňuje velikost zpoždění v průběhu času a je generován metodou `DrawGraphLatency`.

Tato metoda se připojí do databáze „Measurment" a metodou `fetchall` si vypíše všechny hodnoty a uloží si je do seznamu `latency` a do seznamu `count` si uloží jejich pořadí. Kromě vykreslení grafu má tato metoda na starost zápis do tabulky „LatencyFrames", která obsahuje průměrnou hodnotu naměřeného zpoždění pro danou velikost dat. Průměrnou hodnotu si spočítáme součtem všech hodnot zpoždění metodou `sum` a tuto hodnotu vydělíme délkou (neboli počtem elementů) seznamu `latency`.

Takto získanou průměrnou hodnotu zapíšeme příkazem `INSERT` do tabulky „LatencyFrames" spolu s použitou velikostí zpráv.

Nyní již vytvoříme graf zpoždění v čase. K tomu potřebujeme pole (array) místo seznamů. Proto převedeme seznamy `latency` a `count` na pole metodou `array` z modulu `numpy`. Graf vytvoříme metodou `plot` z modulu `networkx` a jako parametry zadáme pole `count` a `latency`. Dále grafu nastavíme nadpis metodou `subtitle`, nastavíme velikost písma a nastavíme písmo na tučné. Přidáme popisky os metodami `xlabel`, `ylabel` a graf zobrazíme metodou `show`.

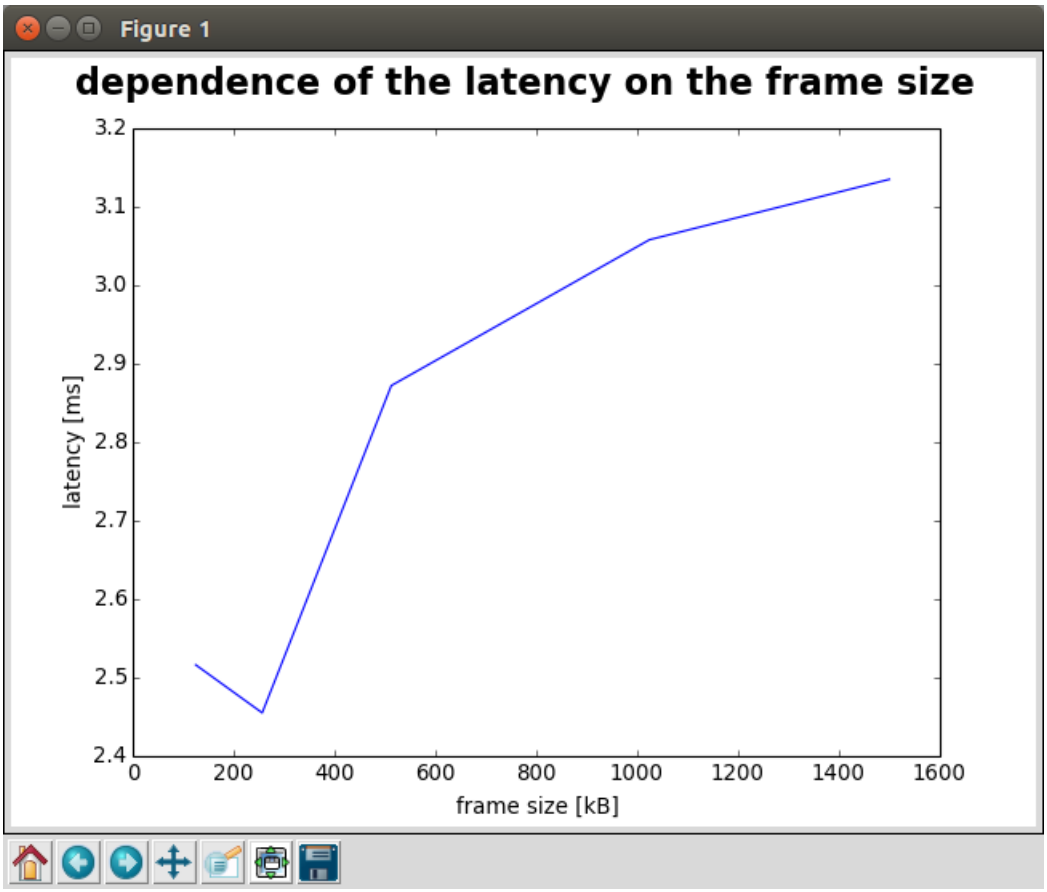


Obr 5.6: Graf zpoždění

5.13.6. *VYKRESLENÍ GRAFU ZÁVISLOSTI ZPOŽDĚNÍ NA VELIKOSTI DAT*

Hlavní tlačítko obsahuje tlačítko „Show graph". Po jeho stisknutí se zobrazí graf závislosti zpoždění na velikosti odesílaných dat. Již u popisu metody `DrawGraphLatency` jsme si popsali, jak zapisujeme průměrné hodnoty zpoždění do tabulky `LatencyFrames`. Nyní pomocí těchto dat vykreslíme graf.

Připojíme se do databáze „Measurment" a metodou `fetchall` si vypíšeme všechny řádky tabulky „LatencyFrames". Obsah prvního sloupce si uložíme do seznamu `framesSize` a druhého sloupce do seznamu `latency`. Převedeme seznamy na pole metodou `array` a metodou `plot` vytvoříme graf. Dále grafu nastavíme nadpis metodou `subtitle`, popíšeme osy metodou `xlabel` a `ylabel` a konečně výsledný graf zobrazíme.



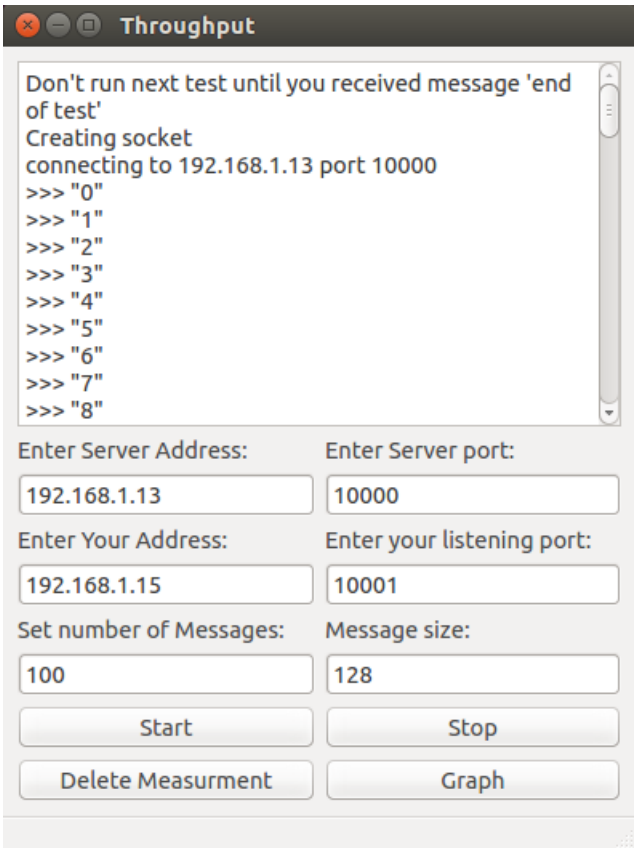
Obr 5.7: Graf závislosti zpoždění na velikosti rámců

5.13.7. VYMAZÁNÍ STARÝCH MĚŘENÍ

Posledním tlačítkem, které jsme si nepopsali v hlavním okně, je tlačítko „Delete measurment". Po stisknutí tlačítka zavoláme metodu `DeletePreviousMeasurment`, kterou se připojíme do databáze „Measurment" a smažeme příkazem `DROP` s podmínkou `IF EXISTS` tabulku „LatencyFrames" obsahující naměřené střední hodnoty zpoždění.

5.14. TEST PROPUSTNOSTI

Test propustnosti spustíme kliknutím na tlačítko „Throughput" umístěné na hlavním okně. Po kliknutí se nejprve zobrazí okno testu. V něm máme opět textové pole, kam vypisujeme informace pro uživatele, pole pro zadání adresy serveru, naši adresu rozhraní, které chceme použít a čísla portu, na kterém server naslouchá a který použije klient. Dále je tu možnost nastavit délku trvání testu a velikost dat, které budeme odesílat.



Obr 5.8: Okno testu propustnosti

Poté tu jsou čtyři tlačítka se stejnou funkcí jako u testu pro zpoždění. Tlačítkem „Start" spouštíme test, tlačítkem „Stop" test předčasně ukončíme, „Delete Measurment" vymaže předchozí měření a tlačítkem „Graph" zobrazíme graf závislosti propustnosti na velikosti zpráv.

Po stisknutí tlačítka „Start" dojde k zavolání metody `pushStart`, která si uloží informace z textových polí metodou `text` a zkontroluje jejich obsah. IP adresy jsou kontrolovány `if` konstrukcí, kde metodou `findall` hledáme regulární výraz `r'[0-9]+(?:\.[0-9]+){3}/[0-9]{1,2}'`. Ostatními políčky kontrolujeme jen jestli jde k obsahu přičíst jednička, čímž si ověříme, že se jedná o číslo.

Nakonec vytvoříme vlákno `WorkThread` pro odesílání dat a připravíme si zachytávání signálů. První ze signálů `je Text_line_Update`, který po zachycení volá metodu `updateTextEdit`. Tato metoda slouží k výpisu informací do textového pole na okně testu. Druhý signál volá metodu `updateThroughput`.

Metodě `updateThroughput` předáváme jako parametr čas, kdy dorazí poslední potvrzení ze strany serveru o odeslání dat. Metodou vypočítáme celkovou propustnost jako velikost odesílaných zpráv vynásobený počtem odeslaných zpráv. Tento součin vydělíme dobou trvání testu. Vypíšeme informaci o propustnosti do testového okna a přihlásíme se do databáze „Measurment", kde vytvoříme tabulku příkazem `CREATE TABLE` s podmínkou `IF NOT EXISTS` a pojmenujeme tabulku „Throughput". Do tabulky zapíšeme velikost propustnosti a použitou velikost zpráv.

Stisknutím tlačítka „Stop" zavoláme metodu `pushStop`, která zavolá metodu vlákna `WorkThread` stop.

5.14.1. *VLÁKNO PRO ODESÍLÁNÍ DAT*

Vlákno pro odesílání dat funguje na stejném principu jako vlákno tvořené v testu zpoždění. Vytvoříme vlákno pro přijímání dat, připravíme si zachytávání signálu a vlákno spustíme. Dále vytvoříme BSD schránku, kterou použijeme pro odesílání dat.

Před odesíláním si uložíme čas získaný příkazem `datetime.datetime.now()`, který značí začátek odesílání zpráv.

Odesílání provádíme ve `for` cyklu. Rozdíl oproti testu zpoždění je obsah zpráv, které posíláme. Zatímco u zpoždění jsme posílali časové značky, nyní nám stačí posílat pouze náhodná data, například číslo s identifikací zprávy.

Po zachycení signálu `Time_line_update` z vlákna pro přijímání dat zavoláme metodu `sendTime`, která přijatý signál obsahující řetězec znaků s časovou značku označující čas přijetí poslední zprávy převede z podoby textového řetězce na časový údaj metodou `strptime`, kde jako výsledný tvar očekáváme "`%Y-%m-%d %H:%M:%S.%f`".

Nyní vypočítáme čas trvání testu odečtením času odeslání první zprávy od přijetí poslední zprávy. Takto získaný časový údaj si převedeme na milisekundy a pošleme signál hlavnímu vláknu testu.

Výpis kódu 5.24: Výpočet doby trvání testu a přeposlání hlavnímu vláknu

```
  def sendTime(self, text):
        time2 = datetime.datetime.strptime(text[0:26], "%Y-%m-%d %H:%M:%S.%f")
        time= time2 - WorkThread.time1
        sec = ((time.days * 24 * 60 * 60 + time.seconds) * 1000 +
             time.microseconds / 1000.0)/1000.0
        self.emit(QtCore.SIGNAL('Time_Line_Update'), str(sec))
```

5.14.2. *VLÁKNO PRO PŘIJÍMÁNÍ DAT*

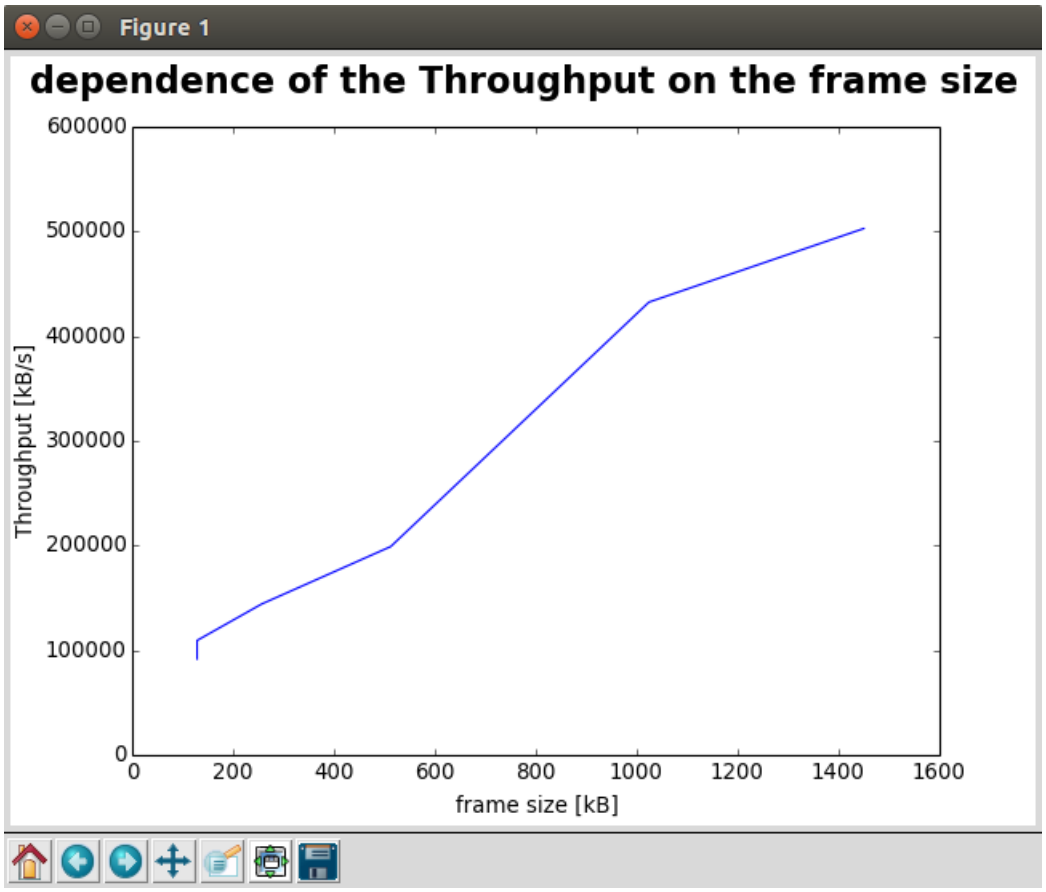
Vlákno pro přijímání dat je stejná jako v testu pro zpoždění. Po jeho spuštění se zavolá metoda `run`. V této metodě vytvoříme BSD schránku, přidělíme jí adresu a port a přepneme ji do režimu naslouchání pro očekávání příchozího spojení. Protokol obsažený ve zprávách je identický jako u zpráv se zpožděním.

Přijatá data vypisujeme do hlavního okna testu a po přijetí poslední zprávy vyšleme signál vláknu pro odesílání dat s časovou značkou a ukončíme spojení.

5.14.3. *VYKRESLENÍ GRAFU PROPUSTNOSTI*

V tomto testu negenerujeme po každém spuštění testu graf, ale pouze graf závislosti propustnosti na velikosti zpráv. Pro tyto účely máme ve třídě testu metodu `drawGraph`. V této metodě se připojíme do databáze s výsledky, které sem zapsala metoda `updateThroughput`.

Pomocí metody `fetchall` si vyčteme všechny řádky tabulky „Throughput", obsah prvního sloupce si uložíme do seznamu `framesSize` a obsah druhého sloupce do seznamu `Throughput`. Seznamy si převedeme na pole metodou `array` z modulu `numpy` a pomocí metody `plot` vytvoříme graf. Dále už jen vytvoříme nadpis, popíšeme osy a graf zobrazíme.

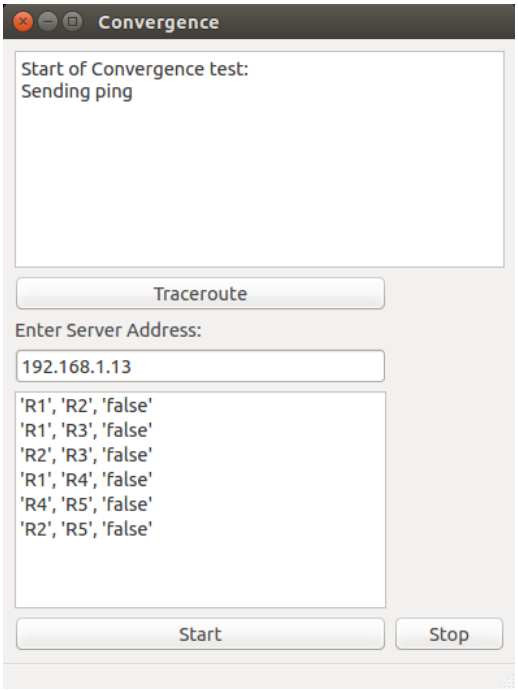


Obr 5.9: Graf závislosti propustnosti na velikosti zpráv

5.15. TEST KONVERGENCE SÍTĚ

Dalším testem aplikace je test konvergence sítě. Základní myšlenkou je vypnout některá rozhraní používaná pro přenos dat a měřit čas, za který síť zkonverguje k řešení a nalezne alternativní cestu.

Kliknutím na tlačítko „Convergence" na hlavním okně aplikace nám otevře okno testu. Toto okno obsahuje textové pole sloužící pro výpis informací uživateli, tlačítko pro spuštění mapování cesty k serveru, textové pole pro zadání adresy serveru, seznam cest sítě, tlačítko „Start" pro spuštění testu a „Stop" tlačítko pro předčasné ukončení testu. Okno bylo vypracování pomocí QT designeru.



Obr 5.10: Okno testu konvergence sítě

5.15.1. *Inicializace okna*

Při inicializaci okna dojde k zavolání metody `getPathsFromMain`, která nám `for` cyklem nahraje všechny položky seznamu cest z hlavního okna a vytvoří je i v seznamu v okně testu. Ještě musíme seznamu umožnit výběr více položek, aby uživatel mohl označit a následně vypnout více cest najednou. To provedeme metodou objektu `QlistWidget` `setSelectionMode` kdy zapneme mód `multiselection`.

5.15.2. *Mapování cest*

Abychom mohli vypnout rozhraní, přes která probíhá komunikace se serverem, máme připraveno mapování cesty. To uživatel spustí stisknutím tlačítka „Traceroute". Po jeho stisknutí dojde k zachycení signálu a zavolání metody `pushTraceroute`.

V této metodě zkontrolujeme obsah textového pole `QEditLine` pro zadání IP adresy serveru, jestli obsahuje adresu. To provedeme metodou `findall` a regulárním výrazem.

Pokud je adresa zadána, tak vytvoříme a spustíme vlákno `WorkThread`, na kterém bude mapování probíhat. Mapování je na jiném vlákně aby v jeho průběhu nedošlo k zamrznutí GUI. Také si nachystáme odchyt signálů z `WorkThread` vlákna. Po zachycení signálu zavoláme metodu `updateTextEdit`, která slouží pro vypsání informací do textového pole okna testu.

Po zapnutí vlákna je zavolána metoda `run`, ve které si nastavíme maximální počet skoků na 30 a proměnnou `ttl` (time to live) na hodnotu 1. Tuto hodnotu budeme každým průchodem smyčky zvětšovat o jedna, dokud není mapování dokončeno, nebo její hodnota nebude rovna maximálnímu počtu skoků.

Ve `while` smyčce vytvoříme BSD schránku jako „raw socket" parametrem `socket.SOCK_RAW`, neboť „raw socket" je typ síťové schránky pro posílání ICMP zpráv bez

použití TCP a UDP zapouzdření. Tato schránka bude zachytávat odpovědi od dotázaných zařízení, které budou odpovídat protokolem ICMP.

Pro vytvoření schránky typu „raw socket" aplikace potřebuje práva „root" (administrátorský účet) uživatele, tudíž pokud chceme využívat mapování cest, je potřeba aplikaci spouštět pomocí sudo příkazu.

Také si vytvoříme schránku typu `SOCK_DGRAM`. Schránka s tímto módem představuje UDP komunikaci.

Nakonec si přiřadíme port pro naslouchací schránku a očekáváme odpovědi metodou `bind` a schránkou pro odesílání pošleme metodou `sendto` prázdný řetězec na adresu serveru.

Dále analyzujeme přijaté ICMP pakety zachycené naslouchající schránkou a metodou `recvfrom`, kterou přijmeme prvních 512 bytů. Aby mapování fungovalo, musí být firewallem povolená ICMP komunikace a každé zařízení v cestě musí mít povoleno odesílání ICMP request zpráv (dotazy ICMP protokolu). Proto je přijímací část s metodou `recvfrom` umístěn v `try` bloku, aby nedošlo k chybě způsobené vypršením času (timeout).

Z přijatých 512 bytů uložených v proměnné `curr_addr` vyjmeme první slovo s adresou a schránky ukončíme metodou `close`.

Nakonec budeme porovnávat získanou adresu s adresami našich směrovačů v testované síti. Přihlásíme se do databáze „RoutersList" a procházíme každou tabulku databáze, jestli nemá ve sloupci IP hledanou IP adresu. Pokud je nalezena shoda, vypíšeme do textového pole okna testu o jaký skok se jedná, jakou adresu jsme získaly a vypíšeme kterému směrovači patří. Pokud jsme nenašli žádnou shodu v databázi, vypíšeme pouze číslo skoku a získanou adresu.

Ještě zvětšíme hodnotu proměnné `ttl` a zkontrolujeme, jestli její hodnota není větší než maximální počet skoků a zkontrolujeme, jestli získaná adresa není cílová adresa. Pokud není ani jedna z podmínek splněna pokračujeme dalším průchodem `while` smyčkou.

5.15.3. SPUŠTĚNÍ TESTU KONVERGENCE SÍTĚ

Před samotným spuštěním testu potřebujeme označit cesty v seznamu, které chceme v průběhu testu vypnout. Když máme výběr hotový, stačí zmáčknout tlačítko „Start".

Po jeho stisknutí dojde k zavolání metody `pushStart`, která opět kontroluje pomocí regulárního výrazu hodnotu IP adresy zadanou do textového pole pro IP adresu serveru a zavolá metodu `disablePaths`, jejíž návratovou hodnotu si uloží do proměnné `ipaddrEnable`, která slouží k navrácení konfigurace.

Metodou `disablePaths` vytvoříme seznam vybraných položek seznamu cest v okně testu `for` cyklem. Dále zkontrolujeme, která z vybraných cest je zapnutá a uložíme si jména směrovačů, které tato cesta spojuje do proměnných `router1` a `router2`.

Dále se připojíme k databázi a hledáme, která cesta v tabulce „Paths" obsahuje tyto dva směrovače `router1` a `router2`. Pokud najdeme shodu, přidáme jejich názvy do seznamu metodou `append` a adresu `router1` směrovače spolu s jeho názvem do seznamu `ipaddr`.

Poté zavoláme metodu `drawGraphFunc`, které předáme parametr seznam `disabledPath`. Metoda zkusí vytvořit graf sítě bez cest v seznamu `disabledPath` a zkontroluje, jestli má každý vrchol grafu cestu k ostatním vrcholům, čímž ověříme, že zůstanou všechny směrovače dostupné.

Pokud je daná konfigurace v pořádku a nedojde k odstřižení nějakého ze směrovačů od zbytku sítě, tak spustíme vlákno `WorkThreadConvergence`, které posílá programem ping ICMP pakety na server. Metodou `sleep` počkáme dvě vteřiny, aby vlákno zahájilo posílání dat, a poté zavoláme metodu `sendDisableToRouter` , kterou jsme již popisovali u funkcionality hlavního okna. Metoda pomocí MikrotikAPI pošle do směrovačů příkaz, kterým vypne zadané adresy. Adresy jsou předány jako parametr.

Po ukončení testu dojde k zaslání signálu s řetězcem znaků "Restoring configuration", čímž zavoláme metodu `restorConfiguration`, která volá metodu `sendDisableToRouter`, ale tentokrát s parametrem `"enable"`, čímž zašleme směrovačům příkaz o zapnutí adres.

5.15.4. VLÁKNO PRO ODESÍLÁNÍ ICMP ZPRÁV

Po zapnutí vlákna `WorkThreadConvergence` zavoláme metodu `run` a ve `while` smyčce si zjistíme systémový čas jako `timeOfPing` a zapneme program ping, kterým odešleme jeden ICMP paket serveru. Ping spouštíme ještě s parametrem `-W` jehož hodnotu nastavíme na 1, což znamená, že bude čekat na odpověď maximálně jednu vteřinu.

Zkontrolujeme návratovou hodnotu, jestli bylo odeslání úspěšné a inkrementujeme proměnnou `succes` nebo `failed`. Pokud inkrementujeme `failed` poprvé, uložíme si systémový čas odeslání paketu do `time1`, tudíž doba výpadku je počítána od ztráty prvního paketu. Při inkrementaci `failed` si také uložíme čas odeslání paketu do `time2`. Smyčku opakujeme, dokud neodešleme sto ICMP paketů.

Nakonec vypočítáme čas výpadku sítě jako rozdíl časových značek `time2` a `time1`, kde `time1` je čas, kdy program ping selhal v odeslání paketu poprvé a v proměnné `time2` je uložen čas, kdy selhalo odeslání naposledy. Čas si převedeme na milisekundy a vypíšeme čas do textového pole v okně testu.

Výpis kódu 5.25: Smyčka spouštějící program ping

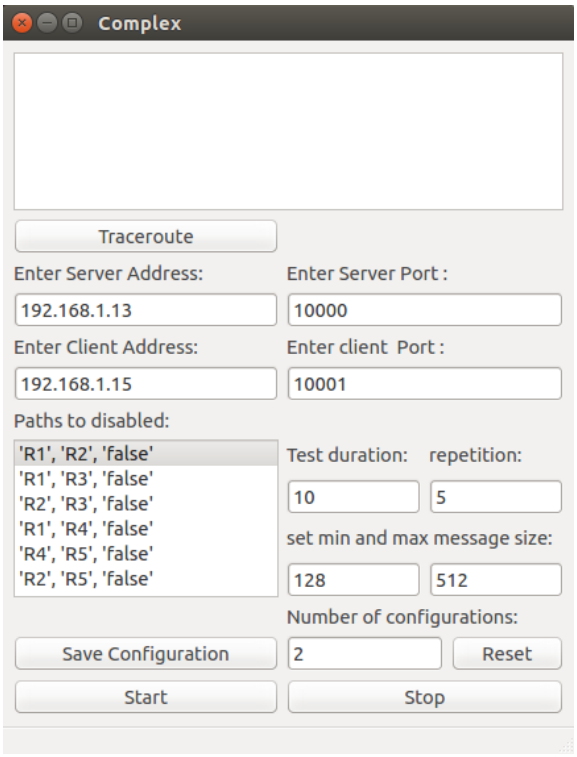
```
while WorkThreadConvergence.keepRunning==True:
            timeOfPing = datetime.datetime.now()
            response = os.system("ping -W 1 -c 1 " + str(self.HOST))
            ping +=1
            if response == 0 :
                succes +=1
            else:
                failed +=1
                if failed==1:
                    time1 = timeOfPing
                time2 = datetime.datetime.now()
            if ping > 99 :
                WorkThreadConvergence.keepRunning=False
            time.sleep(0.1)

        self.writeToLog('succesful ping: %i' % succes )
        self.writeToLog('Failed ping: %i' % failed )
```

5.16. SOUHRNNÝ TEST

Pro jednodušší používání aplikace může uživatel spustit souhrnný test, který provede předchozí testy a vygeneruje protokol obsahující grafy naměřených hodnot, diagramy použitých konfigurací a výpis konfigurací směrovačů.

5.16.1. OKNO SOUHRNNÉHO TESTU

Zobrazení okna souhrnného testu provedeme stisknutím tlačítka „Complex" v hlavním okně programu. Okno testu obsahuje informační textové pole, tlačítko „Traceroute", kterým se spouští mapování cesty, dále jsou tu textové pole pro zadání IP adres a portů, seznam cest, textové pole pro zadání doby trvání jednotlivých testů (neovlivňuje všechny testy), počet opakování testů s měnící se velikostí zpráv, nastavení minimální a maximální délky zpráv, tlačítko „Save Configuration" pro uložení konfigurace sítě, textové pole, kde se zobrazuje počet konfigurací, tlačítko „Reset", kterým jde konfigurace vymazat a konečně tlačítko „Start" pro zapnutí testu a „Stop" pro předčasné ukončení.



Obr 5.11: Okno souhrnného testu

5.16.2. PŘÍPRAVA KONFIGURACÍ SÍTĚ

Před samotným spuštěním musí uživatel uložit konfigurace sítě, které chce testovat. Kliknutím se cesta zvýrazní a po stisknutí tlačítka „Save Configuration" zavoláme metodu `savePath`. Tato metoda volá metodu `disablePaths`, která stejně jako metoda `disablePaths` ve třídě `MainWindow` zkontroluje, jestli danou konfigurací nedojde k odstřižení některého ze směrovačů. Pokud je kontrola metodou `disablePaths` úspěšná, její návratová hodnota obsahuje seznam IP adres spolu se jmény směrovačů, kterým náleží. Dále vypíšeme do textového pole označeného „Number of configurations" počet připravených konfigurací.

Poté přidáme seznam získaný jako návratovou hodnotu metody `disablePaths` do seznamu `listOflists` obsahujícího seznamy IP adres, které se mají zakázat,a to pro všechny konfigurace.

Konfigurace si uložíme to tabulky „ComplexConfiguration", kde každá položka představuje jednu konfiguraci a obsahuje seznam `disablePaths`. Takto uložené konfigurace jsou později použity pro vygenerování síťových diagramů ve výsledném protokolu.

5.16.3. SPUŠTĚNÍ TESTU

Po stisknutí tlačítka „Start" se zavolají dvě metody. Nejprve metoda `deleteOldData`, která příkazem `DROP TABLE` s podmínkou `IF EXIST` smaže tabulky s naměřenými hodnotami, aby se nezamíchali s hodnotami novými.

Druhou metodou je metoda `startTest`. Touto metodou zavoláme metodu `checkValues`, která kontroluje zadané hodnoty v okně testu. Poté se podíváme, jestli je vybraná alespoň jedna konfigurace pro testování, pokud ne, upozorníme uživatele chybovou hláškou.

Dále si uložíme hodnoty zadané v textových polích testovacího okna a přejdeme k části, ve které voláme metody jednotlivých testů.

Pro udržení kontroly jaký test se provádí, kolikrát byl opakován a jaká konfigurace se testuje jsou tu proměnné `testNumber`, což je celkový počet testovaných konfigurací. Dále `testRound`, který představuje číslo právě testované konfigurace. Proměnná `currectTest` zase určuje jaký druh testu pro danou konfiguraci proběhne a `repetitionCount` je číslo kolikrát byla daná konfigurace testována s tím, že pro každý test se mění velikost zpráv. Podle těchto proměnných spouštíme testy.

Podle hodnoty `currentTest` zavoláme metodu buď pro test konvergence `convergenceTest`, test zpoždění `LatencyTest` a nebo test propustnosti `throughputTest`. Dále metodou `sendDisableToRouter` umístěné ve třídě `MainWindow` vypneme rozhraní pro dosážení požadované konfigurace. Které IP adresy mají být zakázány, metodě předáme parametrem `ipaddr`, což je seznam získaný metodou `disablePaths` a je uložený v seznamu `listOflists`.

5.16.4. TEST KONVERGENCE SÍTĚ

Metoda `convergenceTest` spustí vlákno `WorkThreadSending` s parametrem `Convergence`. Po spuštění vlákna `WorkThreadSending` zavoláme metodu `run`, ve které `if` konstrukcí zjistíme hodnotu parametru testu, podle kterého zavoláme metodu `convergence`, `latency` a nebo `throuhgput`.

Při testu konvergence zavoláme metodu `Convergence`, která provede test stejným způsobem jako samotného test konvergence sítě pomocí programu ping, pouze je zde přidáno nastavení velikosti ICMP paketu pomocí parametru `-s`.

Po započnutí odesílání paketu na vlákně `WorkThreadSending` dojde k zavolání metody `sendDisableToRouter` na hlavním vlákně a dojde tak k požadovanému výpadku.

Výsledky zapíšeme do tabulky `ComplexConvergence`, kam zapíšeme délku výpadku, hodnotu `testRound`, určující pro jakou konfiguraci byl test proveden a velikost odesílaných zpráv.

5.16.5. TEST ZPOŽDĚNÍ SÍTĚ

Pokud metodou `run` zavoláme metodu `latenci`, tak vytvoříme BSD schránku pro odesílání dat. Vytvoříme a spustíme třetí vlákno `WaitingWorkThread`, kde v metodě `run` zavoláme metodu `latency` a vytvoříme schránku pro přijímání dat a odesíláme časové značky na server stejným způsobem, jako u samostatného testu zpoždění sítě.

Zapisování výsledků zajišťuje vlákno `WaitinWorkThread`, které v metodě `latency` po dokončení přijímání dat od serveru vypočítá průměrnou hodnotu zpoždění získanou metodou `mean` z modulu `numpy`. Výsledky jsou zapsány do tabulky `ComplexLatency`.

5.16.6. TEST PROPUSTNOSTI SÍTĚ

Pokud parametr test při spuštění vlákna `WorkThreadSending` obsahuje řetězec znaků `"Throughput"`, zavoláme v metodě `run` metodu `throughput`. Tou na základě stejného principu jako u samostatného testu pro propustnost vytvoříme BSD schránku pro odesílání dat, spustíme vlákno `WaitingWorkThread`, kterým přijímáme potvrzení od serveru.
Výslednou propustnost vypočítáme jako součin délky zpráv a počtu přijatých zpráv, který vydělíme dobou trvání přenosu dat.

Takto získanou propustnost zapíšeme do tabulky „ComplexThroughput" spolu s hodnou `testRound` určující pro jakou konfiguraci byl test proveden a velikostí odesílaných zpráv.

5.16.7. UKONČENÍ TESTOVÁNÍ

Každý test po svém ukončení pošle signál hlavnímu vláknu. Po obdržení signálu zkontrolujeme jestli hodnota `testRound` (číslo testované konfigurace) je menší než celkový počet konfigurací `testNumbe`r konstrukcí `if`, ve které je ještě obsažena druhá podmínka, jestli je hodnota `keepRunning` rovna `True`. To je z toho důvodu, kdyby uživatel předčasně ukončil testování tlačítkem „Stop". Pokud jsou obě podmínky splněny, zavoláme opět metodu `startTest`.

Pokud podmínky splněny nejsou, tak k vytvoření instance třídy `GeneratingProtocol` zavoláme její metodu `generate`. Po ukončení metody `generate` zavoláme metodu `pushReset`.

Metoda `pushReset` je volána jak po zmáčknutí tlačítka reset v okně souhrnného testu, tak po ukončení testování. Tato metoda vymaže všechny proměnné `testNumber`, `testRound`, `repetitionCoun` i `currentTest` a také smaže tabulku s předpřipravenými konfiguracemi.

5.17. GENEROVÁNÍ PROTOKOLU

Po ukončení testování vygenerujeme protokol o měření s diagramy konfigurací sítě, grafy s výsledky a konfigurací směrovačů. Protokol je ve formátu pdf a jeho generování provádíme pomocí modulu `matplotlib`.

Pro tvorbu protokolu máme vytvořenou třídu `GeneratingProtocol` v souboru `GenerateProtocol`. V této třídě máme metodu `generate`, které předáme parametr počtu konfigurací. Dále si v metodě vytvoříme obrázek `fig` a nastavíme mu požadovanou velikost, v našem případě 8,27 palců šířku a 11,69 palců výšku, což je velikost formátu A4.

Grafu v obrázku nastavíme nadpis, vypneme osy metodou `axis` s parametrem `'off'` a metodou `figtext` vypíšeme datum. Tento „graf" tvoří úvodní stranu. Následující grafy

přeberou velikost z tohoto prvního. Obrázek uložíme do pdf a graf `plt` vyprázdníme metodou `clf`.

Výpis kódu 5.26: Generování titulní strany

```
            fig = plt.figure(figsize=(8.27, 11.69), dpi=100)
             plt.title('Testing protocol', fontsize=40, fontweight='bold')
             date=datetime.datetime.today()
             plt.figtext(0.05, 0.8, 'date: %s' % str(date)[0:10])
             plt.axis('off')
             #plt.figtext(0.05, 0.5, 'foo2')
             # Save our graph to file graph.png
             pdf.savefig()
             plt.clf()
```

5.17.1. GENEROVÁNÍ DIAGRAMŮ

Další strany obsahují diagramy sítí a jsou rozděleny na čtyři „řádky" a dva „sloupce". Metodou `fetchall` získáme z databáze „Measurment" seznam zakázaných IP adres a stejným způsobem, jako u vytváření diagramu v hlavním okně aplikace vytvoříme diagram sítě.

Pokud je číslo vygenerovaného diagramu sudé (druhý, čtvrtý, šestý) tak ho vložíme na pozici druhého řádku s velikostí dvou řádků a dvou sloupců. Pokud je diagram lichý, tak jej umístíme na začátek stránky, tedy na pozici nultého řádku a nultého sloupce, velikost ponecháváme stejnou.

Dále zkontrolujeme, o jakou se jedná konfiguraci, neboť se na stranu vejdou pouze dva grafy. Tudíž po každém sudém grafu zavoláme metodu `tight_layout` upravující obsah stránky tak, aby se vše vešlo, metodou `savefig` uložíme stranu do našeho pdf souboru.

5.17.2. GENEROVÁNÍ GRAFŮ MĚŘENÍ

Grafy měření jsou v protokolu tři. Graf doby konvergence, zpoždění a propustnost. Proto stranu pro grafy rozdělíme na 6 řádků a dva sloupce.

Na pozici nultého řádku umístíme graf konvergence. `For` cyklem z tabulky „ComplexConvergence" získáme hodnoty naměřené doby konvergence a velikost zpráv, které byly pro dané měření použity. Tyto hodnoty si uložíme do seznamu `framesSize` a `convergence` metodou `append`.

Tyto seznamy převedeme na pole metodou `array` a vytvoříme metodou `plot` graf. Jelikož graf bude zobrazovat měření pro různé konfigurace (automaticky se křivky oddělují barevně) vytvoříme ke grafu ještě legendu metodou `legend` a umístíme ji do horního pravého rohu. Nakonec nastavíme nadpis grafu a popis os. Stejným způsobem vytváříme graf zpoždění i propustnosti.

Nakonec uložíme grafy do pdf souboru a nastavíme mu základní informace metodou `infodict`, jako nadpis, předmět a datum vytvoření. Nyní vytvoříme instanci třídy ftp a zavoláme její metodu `routerInfo`.

5.17.3. FTP KLIENT

Třída `ftp` je umístěna v souboru `ftp` a slouží k vytvoření zálohy konfigurací do textového souboru ve směrovačích, vytvoření ftp klienta, stáhnutí konfigurací a jejich vypsání do protokolu.

Při zavolání metody `routerInfo` se připojíme nejprve do databáze „RouterLists" a metodou `fetchall` získáme její obsah, který ve `for` cyklu analyzujeme. Obsah prvního sloupce si uložíme do proměnné `routername`, obsah druhého sloupce s IP adresou do `ipaddr`, obsah třetího sloupce s uživatelským jménem do `username` a poslední sloupec s heslem si uložíme do proměnné `password`. Tyto proměnné použijeme jako parametry pro metodu `createBackup`, která slouží k připojení pomocí MikrotikAPI do směrovačů a poslání příkazu o exportu konfigurace do textového souboru.

Metodou `createBackup` vytvoříme BSD schránku pomocí metody `openSocket` a dále zavoláme metodu `loginToRouter` pro přihlášení se do směrovače. Metody `createSocket`, `loginToRouter` a `sendMessage` jsou stejné jako ve třídě `MainWindow` popsané výše.

Připravíme si příkaz, který chceme odeslat. Vytvoříme si seznam `message`, do kterého metodou `append` vkládáme nejprve příkaz `'/export'` a `'=file='`, kam přidáme i název souboru, do kterého se má konfigurace exportovat. Název je tvořen slovem „backup" a jménem směrovače.

Metodou `sendMessage` pošleme připravený příkaz do směrovače a vyšleme zprávu o ukončení komunikace. Metoda vrací jako návratovou hodnotu jméno textového souboru s exportovanou konfigurací.

Výpis kódu 5.27: Exportování konfigurace do textového souboru

```
    def createBackup(self,routername ,ipaddr, username, password):
        # Call Function to open socket
        result = self.openSocket(ipaddr, 8728)
        if result == False:
            self.ErorrMsg("Error: Check your IP address or connectivity with
                   target")
            return False
        s = result

        # Call Function to Login
        result = self.loginToRouter( s, username, password )
        if result == False:
            self.ErorrMsg("Error: Check your Username or password")
            return False
        apiros = result

        nameOfFile="backup"+ routername

        message = []
        message.append("/export")
        message.append("=file=%s" % nameOfFile)
        ftp.sendMessage( message, apiros )
        message = []
        message.append("/quit")
        ftp.sendMessage( message, apiros )
        return nameOfFile
```

Při návratu do metody `routerInfo` si uložíme návratovou hodnotu metody `createBackup` do proměnné `createBackup` a zavoláme metodu `downloadBackup` a etodu nakonec `readFile`.

Metodu `downloadBackup` jsme volali s parametry `IPaddress` s IP adresou cílového směrovače, `userName` s přihlašovacím jménem, `password` s heslem a `filename` s názvem souboru s konfigurací.

Nejprve si upravíme jméno souboru, neboť exportovaný soubor má koncovku „.rsc" a vytvoříme si instanci třídy `FTP`. Zavoláme její metodu `login` pro přihlášení ke směrovači, přepneme se do kořenového adresáře, kde jsou exportované soubory umístěny a vytvoříme si soubor v adresáři na straně klienta a soubor stáhneme metodou `retrbinary`.

Nakonec ukončíme ftp spojení metodou `quit` a uložíme právě vytvořený soubor metodou `close`.

Výpis kódu 5.28: Stáhnutí konfigurace pomocí ftp

```
  def downloadBackup(self, IPaddress, userName, password, filename):
        filename = filename + ".rsc"
        ftp = FTP(IPaddress)
        try:
            ftp.login(user=userName, passwd = password)
        except:
            self.ErorrMsg("Error: Check if router has enabled ftp")

        ftp.cwd('/')

        localfile = open(filename, 'wb')
        ftp.retrbinary('RETR ' + filename, localfile.write, 1024)

        ftp.quit()
        localfile.close()
```

Posledním krokem v metodě `routerInfo` jsme zavolali metodu `readFile` a jako parametry jsme ji zadali jméno souboru s konfigurací a pdf soubor tvořící protokol o měření. Metoda `readFile` slouží k načtení souborů s konfiguracemi, jejich rozdělení na strany a pomocí modulu `matplotlib` vloženy do pdf souboru.

Nejprve si zjistíme délku konfigurace metodou `sum`, která počítá řádky v souboru `filename`. Metodou `open` otevřeme konfiguraci a uložíme si obsah jako řetězec znaků do proměnné data a vypočítáme, kolik budeme potřebovat stran na zobrazení celé konfigurace. Počet stran vypočítáme jako celkový počet řádků, které vydělíme počtem řádků, které se vejdou na stranu. Výsledek zaokrouhlím nahoru na nejbližší celé číslo metodou `ceil` z modulu `math`.

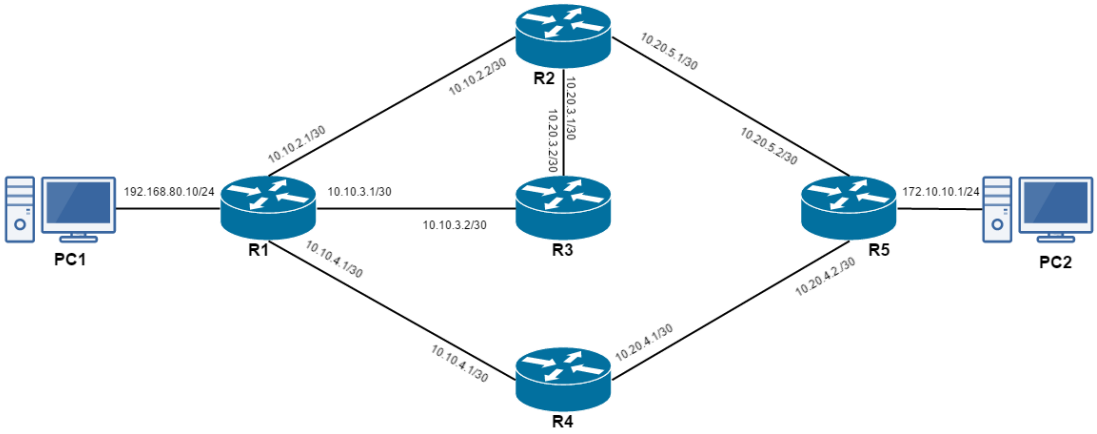
Z proměnné `data` metodou `islice` vybereme počet řádků, které se vejde na A4. Návratová hodnota metody je seznam, ne řetězec znaků, proto ještě překonvertujeme seznam zpět na řetězec znaků metodou `join`. Výsledkem je maximálně 55 řádků konfigurace uložených v proměnné `head`.

Jelikož metoda `figtext` vkládá text od spodu, ještě musíme upravit umístění textu pro každou stranu. To uděláme spočítáním řádků v proměnné `head` a podle počtu řádků nastavíme konstrukcí `if` odsazení od konce stránky.

Nakonec jednotlivé stránky vkládáme do pdf souboru metodou `figtext`, nastavíme nadpis konfigurace a pdf soubor uložíme.

5.18. TESTOVÁNÍ APLIKACE

Aplikace byla vyvíjena a testována na operačním systému Ubuntu verze 14.04 a Python verze 2.7.6. Síť, na které byla aplikace testována, se skládala z pěti směrovačů. Všechny směrovače byly typu RB751G-2HnD měli verzi RouterOS 6.32.2.



Obr 5.12: Schéma konfigurace testovací sítě

6. PRÁCE S APLIKACÍ

Před prvním spuštěním aplikace je třeba doinstalovat některé balíčky do operačního systému. Také je nutné zkontrolovat, že všechny směrovače testované sítě mají povolené odpovědi na ICMP „requests", povolený protokol FTP a jejich RouterOS podporuje připojení přes MikrotikAPI.

6.1. OVLÁDÁNÍ KLIENTA

Na straně klienta před prvním použitím spustíme skript setup.py, který se nachází ve složce *Client*. Tento instalační skript potřebujeme spustit buď jako „root" uživatel, nebo s příkazem `sudo` a aby skript fungoval správně, musí být přítomen program *python-apt*. Spuštění skriptu je `sudo python setup.py`. Skript doinstaluje balíčky `setuptools`, `matplotlib`, `pyqt4`, `netaddr` a `networkx`.

Pokud chceme používat veškerou funkcionalitu aplikace, včetně mapování cest, tak musíme klienta spustit s příkazem `sudo`. Spuštění provedeme `sudo python Tester.py`. Soubor *Tester.py* spustí hlavní okno aplikace a je umístěn v adresáři s klientem.

6.1.1. PŘIDÁNÍ SMĚROVAČŮ

Nejprve musíme přidat směrovače testované sítě tlačítkem „Add Router". Po jeho stisknutí vyplníme adresu, na které je směrovač dostupný, jeho jméno, uživatelské jméno a heslo a směrovač přidáme potvrzením tlačítkem „Add". Po jeho stisknutí by se nám měl objevit směrovač v seznamu směrovačů, a také by se měl zobrazit v diagramu sítě. Pokud již byly přidány směrovače, se kterými je přímo spojený, vypíšou se jejich cesty v seznamu cest a v diagramu se propojí černou linkou.

Dále máme možnost upravit konfiguraci sítě tak, že vypneme nežádané spoje. K tomu stačí kliknutím označit ty cesty v seznamu cest, které mají být zakázány a kliknout na „Refresh paths".

6.1.2. TEST ZPOŽDĚNÍ

Pro testování samotného zpoždění klikneme na tlačítko „Latency". V otevřeném okně testu vyplníme adresu serveru a adresu klienta, kterou chceme používat pro odesílání zpráv. Můžeme nastavit délku trvání testu a nastavení velikosti odesílaných zpráv, ovšem velikost je nutné určit maximálně do velikosti ethernetového rámce.

Test zahájíme stisknutím tlačítka „Start". Po skončení testu se nám zobrazí graf zpoždění v čase. Tlačítkem „Delete measurment" vymažeme přecházející naměřená data a tlačítkem „Show Graph" zobrazíme graf závislosti zpoždění na velikosti zpráv z dat předchozích testů.

6.1.3. TEST PROPUSTNOSTI

Tlačítkem „Throughput" otevřeme okno pro měření propustnosti. Nastavíme adresy, popřípadě můžeme měnit počet odeslaných zpráv a velikost zpráv. Tlačítkem „Delete Measurment" smažeme naměřené hodnoty. Tlačítkem „Graph" zobrazíme závislost propustnosti na velikosti odesílaných zpráv z předchozích testů.

6.1.4. TEST KONVERGENCE SÍTĚ

Pro test konvergence klikneme na tlačítko „Convergence" a otevře se nám okno testu. Nejprve vyplníme adresu serveru. Stisknutím „Traceroute" si můžeme nechat vypsat používanou cestu v síti a podle toho vypnout cestu, čímž navodíme výpadek sítě. Vybereme cesty, které budeme chtít v průběhu testu vypnout a stiskneme tlačítko „Start". K vypnutí cest dojde dvě vteřiny po té, co se spustí odesílání ICMP paketů k serveru. Výsledná doba konvergence výpadku je vypsána do textového pole umístěného na horní straně okna.

6.1.5. SOUHRNNÝ TEST

Souhrnný test provede sérii testů pro zpoždění, propustnost i konvergenci sítě. Zadáme v okně testu adresy, můžeme si pro testy konvergence vypsat cestu tlačítkem „Traceroute" V kolonce „repetition" zadáme počet opakování testu. V kolonkách pro nastavení minimální a maximální velikosti zprávy máme rozsah velikost, který rozdělíme podle počtu opakování testu. Tudíž při zadání 10 opakování s minimální velikostí zpráv 100 bytů a maximální velikostí 1000 bytů, bude první opakování testu probíhat s velikostí 100 bytů, druhé bude používat velikost zpráv 200 bytů, třetí 300 bytů a podobně.

V kolonce „Number of configuration" musíme mít nějaké konfigurace sítě, které se budou testovat. Na základě výsledku mapování cest vybereme cesty, které se mají vypnout a uložíme si konfiguraci tlačítkem „Save Configuration". Tím se nám inkrementuje počet konfigurací v kolonce „Number of configuration". Nyní můžeme vybrat další konfiguraci sítě a opět vybereme cesty, které se mají vypnout a konfiguraci uložíme.

Tudíž pokud si nastavíme počet opakování na pět a vytvoříme si tři konfigurace, tak dojde k patnácti testům konvergence, zpoždění a propustnosti. Každá konfigurace sítě bude testována pro různé délky odesílaných zpráv.

Tlačítkem „Reset" vymažeme všechny uložené konfigurace. Samotné testování spustíme stisknutím tlačítka „Start".

Uživatel je průběžně informován o výsledcích v horním textovém poli. Po skončení všech testů začne aplikace generovat protokol a stahovat přes ftp konfigurace jednotlivých směrovačů, což zabere několik vteřin. Výsledný protokol je uložen v souboru Protocol.pdf umístěném v adresáři s klientem.

6.2. POUŽITÍ ECHO SERVERU

Před prvním použitím serveru je opět nutné doinstalovat některé balíčky instalačním skriptem. Instalační skript spustíme `sudo python setup.py`. Tento soubor najdeme v adresáři *Server* a doinstaluje nám balíček se `setuptools` a `pyqt4`. Spuštění samotné aplikace provedeme příkazem `python ServerTester.py`, kde tento soubor najdeme také v adresáři *Server*. Pro zapnutí serveru stačí nastavit IP adresu, na které bude server naslouchat a zmáčknout tlačítko „Start".

6.2.1. KONZOLOVÁ VERZE SERVERU

Konzolová verze se od klasické nijak neliší. Instalace balíčků je stejným skriptem *setup.py*. Příkaz pro spuštění je `python ServerConsole [IP address] [Server PORT] [Client PORT]`. Příklad spuštění je `python ServerConsole 192.168.1.1 10000 10001`. Ukončení konzolové verze provedeme stisknutím kláves *ctrl+*.

7. ZÁVĚR

V diplomové práci jsme vyvíjeli aplikaci pro testování sítí skládajících se ze směrovacích prvků od firmy Mikrotik. Aplikace vytváří vlastní provoz, zajišťuje přenos z jednoho koncového zařízení na druhé a zpět. Z dat vyhodnocuje výsledky zatížení sítě. Pro vývoj jsme si zvolili programovací jazyk Python pro jeho přívětivou syntaxi a snadné použití. Začali jsme tvorbou GUI rozhraní pomocí nástrojů PyQt a Qt Designeru, kde jsme vytvořili hlavní okno programu s ovládacími prvky a zobrazováním důležitých informací. Pro ukládání dat o naší síti jsme zařadili SQLite databázi s tabulkou směrovačů obsahující jejich názvy a přístupová rozhraní, tabulkou cest mezi síťovými prvky, tabulkou naměřených dat a podobně.

Také jsme pracovali na vizuálním zobrazení diagramu testované sítě pomocí nástroje NetworkX. Uživatel při přidání směrovače zadá pouze adresu přístupného rozhraní směrovače a pomocí MikrotikAPI se uloží seznam rozhraních spolu s jejich adresami a dojde k automatickému přidání cest mezi rozhraními směrovačů ze stejné sítě, čímž se urychlí práce obsluhy pro vytvoření celkového náhledu na zkoumanou síť.

Připravili jsme testování zpoždění s vygenerováním grafu závislosti zpoždění na velikosti rámců, test propustnosti s výpisem hodnot do informačního okna a test konvergence s vlastní mapovací funkcí. Nakonec jsme připravili souhrnný test, který otestuje všechny zmíněné veličiny najednou, pomocí FTP stáhne konfigurace směrovačů a vygeneruje protokol o měření.

Protokol zobrazuje diagramy použitých konfigurací sítě, grafy závislostí zpoždění, propustnosti a konvergence na délce rámců pro jednotlivé zapojení sítě a nakonec jsou vypsány kompletní konfigurace směrovačů obsahující záznamy o všech rozhraních, aktivních bezpečnostních tunelech a podobně.

Druhou část aplikace tvoří server, který zajišťuje připojení koncového bodu na druhé straně sítě a kterému posíláme klientem přes testovanou síť data. Server slouží pro přeposlání dat zpět na stranu klienta, kde jsou data zpracována a vyhodnocena.

Tato aplikace může sloužit pro testování návrhů menších sítí, kde velikost je limitována manuálním nahráváním jednotlivých směrovačů, časovou náročností konfigurace jednotlivých prvků a stahování jejich celkových konfigurací při generování protokolu.

Aplikace je vytvářena na základě teoretických poznatků uvedených v teoretické části práce. Zabývali jsme se obecně firmou Mikrotik, její historií vzniku, výrobou směrovačů a jejich konfigurací, operačním systémem, výkonností a podobně.

Také jsme se zaměřili na organizaci IEFT vydávající dokumenty RFC, které slouží jako doporučení pro standardy týkající se počítačových sítí, ať už jde o různé protokoly nebo stanovování výkonu a jeho měření. Podrobněji jsme si rozepsali RFC dokumenty RFC 1242 a RFC 2544, ve kterých je popis metod zátěžového testování i prezentování výsledků.

Dále jsme se zabývali teorií grafů, kde jsme si uvedli co je to graf a z čeho se skládá. Zmínili jsme si i základní pojmy týkající se grafů a základní rozdělení.

Také jsme řešili použité nástroje pro vývoj, od použitého programovacího jazyka, přes tvorbu uživatelského rozhraní i knihovny pro různé výpočty a podobně. Popsali jsme si jednotlivé prvky aplikace, události a metody, které jsme názorně demonstrovali kódovými ukázkami. Nakonec jsme se věnovali návodu pro instalaci klienta i serveru a obsluhu celé aplikace.

LITERATURA

[1] About us. *Mikrotik.com* [online]. [cit. 2015-12-10]. Dostupné z: http://www.mikrotik.com/aboutus

[2] The history of Mikrotik. *Mikrotiktutorials.blogspot.cz* [online]. [cit. 2015-12-10]. Dostupné z: http://mikrotiktutorials.blogspot.cz/2011/10/history-of-mikrotik.html

[3] MikroTik RouterOS. *Mikrotik.com* [online]. [cit. 2015-12-10]. Dostupné z: http://download2.mikrotik.com/what_is_routeros.pdf

[4] RouterOS. *Mikrotik-routeros.net* [online]. [cit. 2015-12-10]. Dostupné z: http://www.mikrotik-routeros.net/routeros.aspx

[5] Manual:API. *Wiki.mikrotik.com* [online]. [cit. 2015-12-10]. Dostupné z: http://wiki.mikrotik.com/wiki/Manual:API

[6] What is Python? Executive Summary. *Python.org* [online]. [cit. 2015-12-10]. Dostupné z: https://www.python.org/doc/essays/blurb/

[7] ZELLE, John M. *Python as a First Language* [online]. Wartburg College Waverly, IA 50677: Department of Mathematics, Computer Science, and Physics [cit. 2015-12-10]. Dostupné z: http://mcsp.wartburg.edu/zelle/python/python-first.html

[8] KRILL, Paul. *DEVELOPMENT LANGUAGE PROS AND CONS: A developer's guide to the pros and cons of Python* [online]. 2015 [cit. 2015-12-10]. Dostupné z: http://www.infoworld.com/article/2887974/application-development/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html

[9] RFC. *Webopedia.com* [online]. [cit. 2015-12-10]. Dostupné z: http://www.webopedia.com/TERM/R/RFC.html

[10] About the IETF. *Ietf.org* [online]. [cit. 2015-12-10]. Dostupné z: https://www.ietf.org/about/

[11] RFC 1242. *Datatracker.ietf.org* [online]. Benchmarking Terminology for Network Interconnection Devices, 1991 [cit. 2015-12-10]. Dostupné z: https://datatracker.ietf.org/doc/rfc1242/?include_text=1

[12] RFC 2544. *Datatracker.ietf.org* [online]. Benchmarking Terminology for Network Interconnection Devices, 1999 [cit. 2015-12-10]. Dostupné z: https://datatracker.ietf.org/doc/rfc2544/?include_text=1

[13] BEAL, Vangie. *Ping* [online]. [cit. 2015-12-10]. Dostupné z: http://www.webopedia.com/TERM/P/PING.html

[14] Manual:Winbox. *Wiki.mikrotik.com* [online]. [cit. 2015-12-10]. Dostupné z: http://wiki.mikrotik.com/wiki/Manual:Winbox

[15] BURGET, Radim. *Teoretická informatika* [online]. Purkyňova 118, 612 00 Brno: Vysoké učení technické v Brně,Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013 [cit. 2015-12-10]. ISBN 978-80-214-4897-1.

[16] Graph Theory. *Http://scanftree.com/* [online]. [cit. 2015-12-10]. Dostupné z: http://scanftree.com/Graph- Theory/

[17] BURGET, Radim. *Datové struktury: Grafy*. Pracovní text k předmětu MTIN,VUT v Brně, 2014

[18] BURGET, Radim. *Datové struktury: Prohledávání*. Pracovní text k předmětu MTIN,VUT v Brně, 2014

[19] KOLÁŘ, Josef. *Teoretická informatika* [online]. 2. vyd. Praha: Česká informatická společnost, 2000, 204 s. [cit. 2015-12-10]. ISBN 80-900-8538-5.

[20] Dijkstrův algoritmus. *Algoritmy.net* [online]. [cit. 2015-12-10]. Dostupné z: https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus

[21] *Routerboard.com* [online]. [cit. 2015-12-10]. Dostupné z: http://routerboard.com/

[22] SQL Definition and History. *Informit.com* [online]. [cit. 2015-12-10]. Dostupné z: http://www.informit.com/library/content.aspx?b=STY_Sql_24hours

[23] About SQLite. *Sqlite.org* [online]. [cit. 2015-12-10]. Dostupné z: http://www.sqlite.org/about.html

[24] Overview. *Networkx.github.io* [online]. NetworkX Developers, 2015, 2015-10-26 [cit. 2015-12-10]. Dostupné z: https://networkx.github.io/documentation/latest/overview.html

[25] What is PyQt? *Riverbankcomputing.com* [online]. [cit. 2015-12-10]. Dostupné z: https://riverbankcomputing.com/software/pyqt/intro

[26] Qt framework: Signály a sloty. *Linuxsoft.cz* [online]. [cit. 2015-12-10]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1624

[27] WATZKE, David. *Grafické programy v Qt 4 - 3: Qt Designer* [online]. 2009 [cit. 2015-12-10]. Dostupné z: http://www.abclinuxu.cz/clanky/programovani/graficke-programy-v-qt-4-3-qt-creator-a-designer

[28] HUNTER, John. *Matplotlib* [online]. 2012 [cit. 2015-12-10]. Dostupné z: http://matplotlib.org/

[29] The Python Standard Library: Built-in Functions. *Python.org* [online]. [cit. 2016-03-12]. Dostupné z: https://docs.python.org/2/library/functions.html#chr

[30] Introdution. *Netaddr* [online]. David P. D. Moss [cit. 2016-05-10]. Dostupné z: https://netaddr.readthedocs.io/en/latest/introduction.html

[31] Introdution. *Matplotlib.org* [online]. John Hunter Technology Fellowship [cit. 2016-05-10]. Dostupné z: http://matplotlib.org/

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

RFC *Request For Comments* - Žádost o komentáře, dokumenty s doporučeními

IETF *Internet Engineering Task Force* - Komise pro technickou stránku internetu

BMWG *Benchmarking Methodology Working Group* - Pracovní skupina pro metodiky zátěžového testování

MTU *Maximum transmission unit* - Maximální přenosová jednotka

DUT *Device under test* - Testované zařízení

GUI *Graphical User Interface* - Grafické uživatelské rozhraní

QT knihovna pro tvorbu GUI

PyQt knihovna pro tvorbu GUI v programovacím jazyku Python

CPU *Central processing unit* - Centrální procesorová jednotka

IPv4 *Internet procol version 4* - Internetový protokol verze 4

IPv6 *Internet procol version 6* - Internetový protokol verze 6

BSD *Berkeley Software Distribution* - Unixový, vyvíjený v Berkley Software Distribution

ICMP *Internet Control Message Protocol* - protokol pro diagnostiku a kontrolu sítí

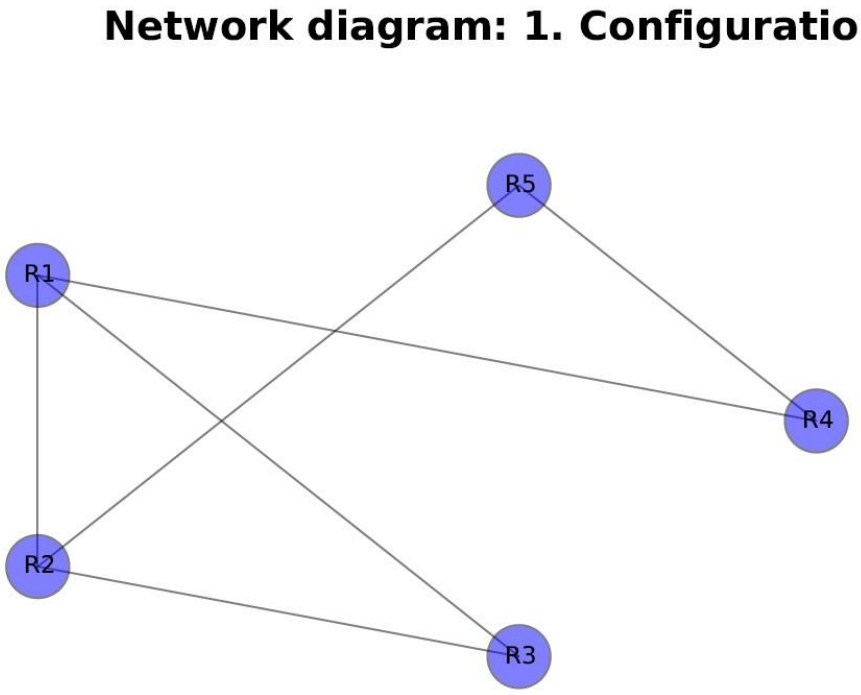
FTP *File Transfer Protocol* - protokol pro přenos souborů

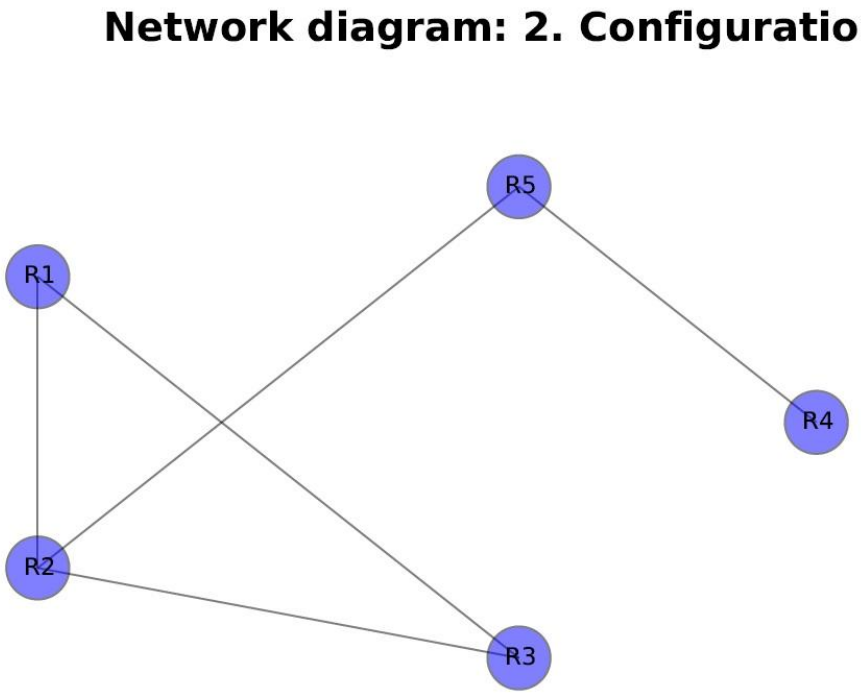
PDF *Portable Document Format* - formát dokumentů vyvinutý firmou Adobe

PŘÍLOHY

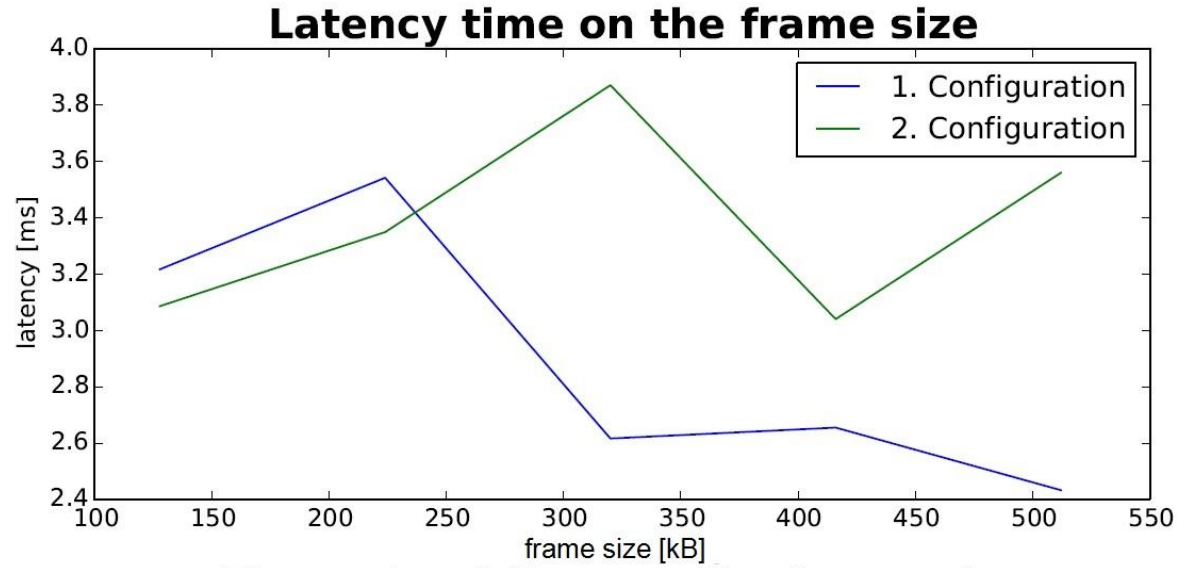
A PROTOKOL

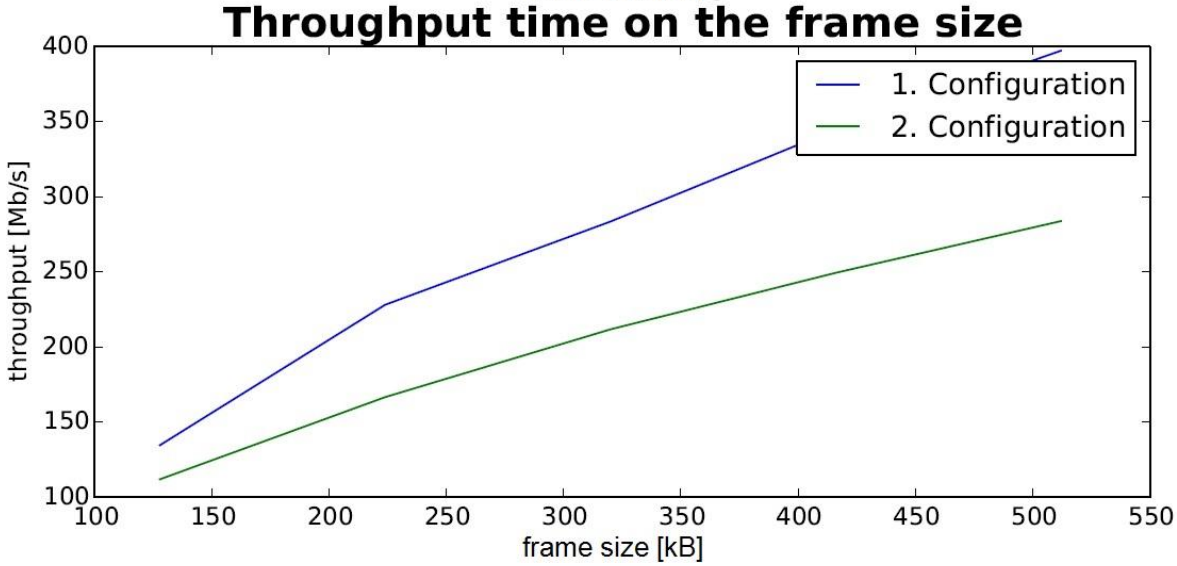
A.1. Ukázka diagramů konfigurací sítě při testování





A.2. Ukázka grafů s výsledky měření





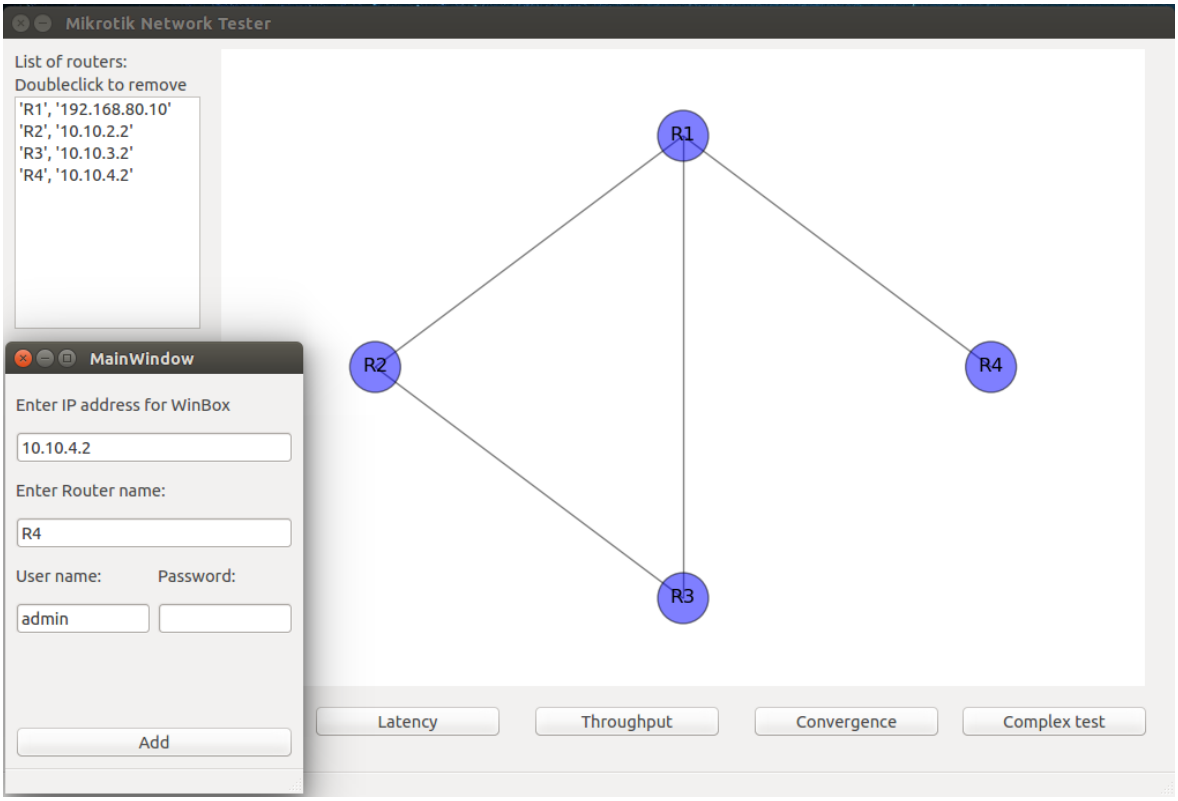
A.3. Ukázka konfigurace směrovače

backupR1.rsc

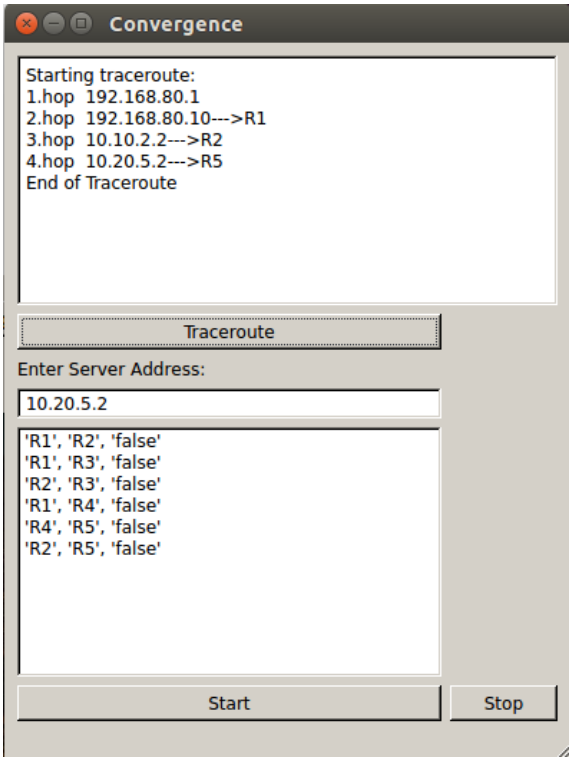
```
# may/06/2016 16:48:21 by RouterOS 6.32.2
# software id = F19C-FINM
#
/interface bridge
add name=bridge1
/interface wireless
set [ find default-name=wlan1 ] band=2ghz-onlyg country="czech republic" \
    disabled=false rx-chains=0 ssid=MikroTik tx-chains=0
/interface vpls
add arp=disabled mac-address=02:58:AB:99:79:88 name=dtz remote-peer=0.0.0.0 \
    vpls-id=0:0
/interface bonding
add disabled=true name=bonding1 slaves=ether2,ether4
/interface wireless security-profiles
set [ find default=true ] authentication-types=wpa-psk,wpa2-psk mode=\
    dynamic-keys supplicant-identity=MikroTik wpa-pre-shared-key=3A6502322625 \
    wpa2-pre-shared-key=3A6502322625
/ip dhcp-client option
add code=1 name=pokusny
/ip dhcp-server option
add code=2 name=test
/ip firewall layer7-protocol
add name=test21
/ip ipsec proposal
set [ find default=true ] enc-algorithms=3des
/ip pool
add name=pool1 ranges=10.0.0.1-10.0.0.2
add name=vpn ranges=10.1.1.2-10.1.1.100
add name=vpn_mmos ranges=10.10.1.2-10.10.1.20
add name=dhcp_pool1 ranges=192.168.80.2-192.168.80.254
add name=dhcp_pool2 ranges=10.10.1.2
add name=dhcp_pool3 ranges=172.10.10.2-172.10.10.254
/ip dhcp-server
add address-pool=dhcp_pool3 disabled=false interface=ether2 name=dhcp1
/mpls traffic-eng tunnel-path
add affinity-include-all=2 affinity-include-any=2 disabled=true name=tp1 \
    record-route=false use-cspf=false
/queue tree
add name=Koren parent=global queue=default
add name=http-up parent=Koren queue=default
add name=http-vut packet-mark=up-vut parent=http-up queue=default
add name=http-zbytek packet-mark=up parent=http-up queue=default
/routing bgp instance
add as=1 client-to-client-reflection=false cluster-id=0.0.0.0 confederation=2 \
    confederation-peers=2 disabled=true name=bgp1 out-filter=ospf-in \
    router-id=20.0.0.0
/routing ospf instance
set [ find default=true ] distribute-default=always-as-type-1 \
    redistribute-connected=as-type-1 router-id=1.1.1.1
/system logging action
```

B UKÁZKY BĚHU PROGRAMU

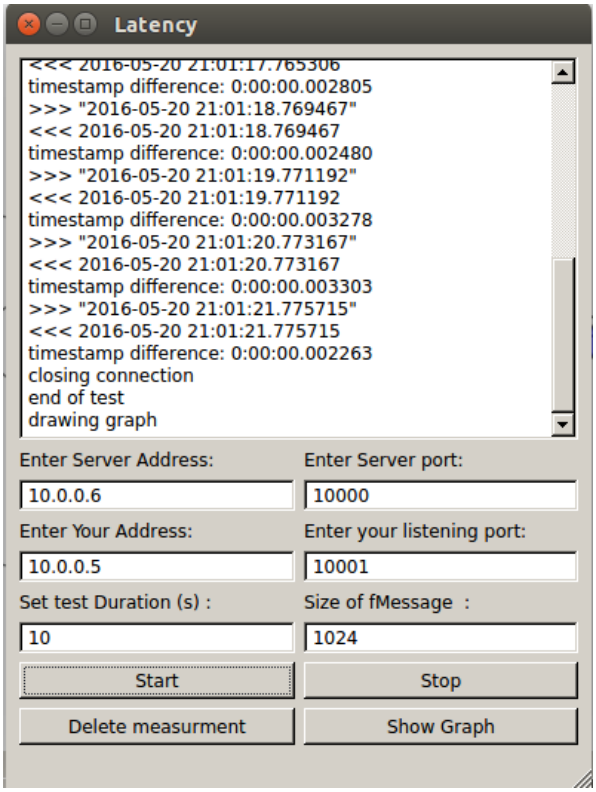
B.1. Přidání směrovače



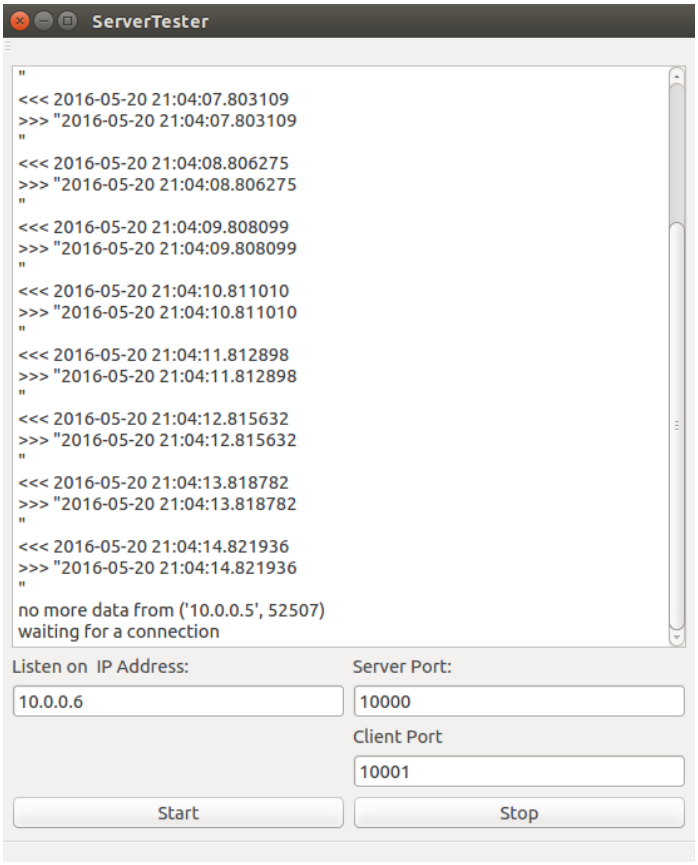
B.2. Mapování cesty



B.3. Klient při testování zpoždění



B.4. Server při testování zpoždění



C OBSAH PŘILOŽENÉHO DVD

TESTER – Adresář aplikace pro měření výkonnosti sítí s prvky mikrotik obsahující:

Client -Prázdný klient připravený pro nastavení testované sítě

Client_configured-Nakonfigurovaný klient s testovací sítí, protokolem a výsledky

Server-Serverová část aplikace

ConsoleServer-Serverová část v konzolové verzi

DIPLOMOVÁ_PRÁCE- elektronická forma textové části diplomové práce