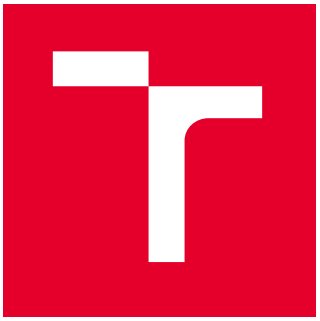


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

SÍŤOVÝ EMULÁTOR

NETWORK EMULATOR

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. František Bílek**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Václav Zeman, Ph.D.**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Ústav telekomunikací

Student: Bc. František Bílek ***ID:*** 230534

Ročník: 2 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Síťový emulátor

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je návrh a implementace síťového emulátoru, který umožní ovlivňovat přenosové parametry paketových sítí pracujících s protokolovou sadou TCP/IP. Student provede analýzu přenosových parametrů vhodných pro emulaci, rozbor technických řešení a návrh modulu splňujícího požadované parametry, který bude implementován a integrován do prostředí Apache JMeter. Práce bude zahrnovat testování funkčnosti emulátoru na příkladech reálných scénářů, vyhodnocení jeho přínosu a vytvoření dokumentace popisující návrh, implementaci a možnosti dalšího rozvoje.

DOPORUČENÁ LITERATURA:

[1] HALILI, Emily H. Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites. Packt Publishing Ltd, 2008.

[2] HEMMINGER, Stephan. Network Emulation with NetEm. Canberra, Australia: Open Source Development Labs, 2005, . Proceedings of the 6th Australia's National Linux Conference (LCA2005).

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 27.5.2025

Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá implementací síťového emulátoru, který umožňuje ovlivňovat přenosové parametry paketových sítí pracujících s protokolovou sadou TCP/IP. Práce zahrnuje analýzu přenosových parametrů a přehled metod pro síťové testování. Zvolený emulátor je integrován do prostředí Apache JMeter. Cílem práce je vytvořit nástroj, který umožňuje emulovat reálné síťové podmínky bez potřeby nákladného hardwarového vybavení.

KLÍČOVÁ SLOVA

Síťový emulátor, Apache JMeter, přenosové parametry, TCP, IP, NetEm

ABSTRACT

This thesis focuses on the implementation of a network emulator that enables the manipulation of network parameters in packet-switched networks using the TCP/IP protocol suite. The thesis includes an analysis of network parameters and an overview of methods for network testing. The selected emulator is integrated into the Apache JMeter environment. The goal of this thesis is to develop a tool that allows the emulation of real network conditions without the need for expensive hardware equipment.

KEYWORDS

Network emulator, Apache JMeter, network parameters, TCP, IP, NetEm

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

BÍLEK, František. *Síťový emulátor*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. František Bílek
VUT ID autora:	230534
Typ práce:	Diplomová práce
Akademický rok:	2024/25
Téma závěrečné práce:	Síťový emulátor

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Václavu Zemanovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod **14**

1 Analýza přenosových parametrů **15**
1.1 Základní prvky komunikační sítě . . . 15
1.2 Šířka pásma a maximální přenosová rychlost . . . 16
1.3 Propustnost . . . 16
1.3.1 Propustnost uzlu . . . 16
1.3.2 Propustnost trasy . . . 17
1.4 Zpoždění paketu . . . 18
1.4.1 Složky zpoždění paketu . . . 18
1.4.2 Zpoždění uzlu . . . 20
1.4.3 Zpoždění trasy . . . 20
1.4.4 Obousměrné zpoždění . . . 20
1.5 Kolísání zpoždění paketů . . . 21
1.6 Ztrátovost paketů . . . 21
1.7 Chybovost paketů . . . 22
1.8 Duplikace paketů . . . 23
1.9 Záměna pořadí paketů . . . 23

2 Metody síťového testování **24**
2.1 Síťová simulace . . . 24
2.1.1 Diskrétní simulátory . . . 24
2.2 Síťové testovací prostředí . . . 25
2.2.1 Softwarově definované sítě . . . 26
2.3 Síťová emulace . . . 26
2.3.1 Emulace síťového spojení . . . 27
2.4 Srovnání testovacích metod . . . 27

3 Přehled síťových emulátorů **28**
3.1 Emulátory založené na virtualizaci . . . 28
3.1.1 CORE . . . 28
3.1.2 CML . . . 28
3.2 Emulátory síťových spojení . . . 29
3.2.1 NetEm . . . 29
3.2.2 DummyNet . . . 30
3.2.3 NISTNet . . . 30
3.2.4 Srovnání emulátorů síťových spojení . . . 32

4 Realizace emulátoru **34**
4.1 Technologie pro vývoj . . . 34
4.2 Software Apache JMeter . . . 34
4.2.1 Architektura systému . . . 35
4.2.2 Registrace komponenty . . . 35
4.3 Řízení síťového provozu . . . 35
4.3.1 Emulace . . . 36
4.3.2 Filtrování . . . 37
4.4 Datová reprezentace . . . 38
4.4.1 Model emulátoru . . . 38
4.4.2 Model síťového rozhraní . . . 39
4.4.3 Model emulačního pravidla . . . 39
4.4.4 Model přenosových parametrů . . . 41
4.4.5 Model filtru . . . 43
4.5 Kontrolér . . . 43
4.6 Grafické uživatelské rozhraní . . . 45
4.6.1 Hlavní komponenta . . . 46
4.6.2 Stavový panel . . . 47
4.6.3 Ovládací panel . . . 47
4.6.4 Konfigurační panel . . . 48
4.6.5 Panel emulačního pravidla . . . 49
4.6.6 Panel síťového rozhraní . . . 51

5 Měření a výsledky **52**
5.1 Popis měření . . . 52
5.2 Měření přenosových parametrů . . . 53
5.2.1 Propustnost . . . 53
5.2.2 Zpoždění a kolísání zpoždění . . . 55
5.2.3 Ztrátovost . . . 62
5.2.4 Chybovost . . . 63
5.2.5 Duplikace . . . 66
5.3 Vyhodnocení výsledků . . . 67

Závěr **69**

Literatura **70**

Seznam symbolů a zkratek **74**

Seznam příloh **78**

A Sestavení a zprovoznění emulátoru **79**

B Skripty použité pro měření přenosových parametrů **80**

B.1 Instalace potřebných nástrojů 80

B.2 Měření propustnosti 80

B.3 Měření zpoždění 81

B.4 Měření chybovosti 82

B.5 Měření ztrátovosti 83

B.6 Měření duplikace 83

C Obsah elektronické přílohy **84**

Seznam obrázků

1.1 Schéma obecné komunikační sítě 15
1.2 Maximální přenosová rychlost a propustnost 17
1.3 Propustnost trasy 17
1.4 Složky zpoždění paketu 18
1.5 Kolísání zpoždění paketů 22
2.1 Diskrétní síťové simulátory 25
2.2 Princip softwarově definované sítě 26
3.1 Pozice systému front v Linuxovém jádře 30
3.2 Princip emulátoru DummyNet 31
3.3 Architektura emulátoru NISTNet 32
4.1 Třídní diagramy emulátoru, síťového rozhraní a emulačního pravidla . 40
4.2 Třídní diagramy přenosových parametrů 42
4.3 Třídní diagram filtru 44
4.4 Výchozí stav emulátoru po přidání do prostředí JMeter 45
4.5 Vzhled stavového panelu 47
4.6 Vzhled ovládacího panelu 48
4.7 Vzhled konfiguračního panelu 49
4.8 Vzhled panelu emulačního pravidla 50
4.9 Vzhled panelu síťového rozhraní 51
5.1 Schéma měřicí soustavy 52
5.2 Absolutní odchylky propustnosti 54
5.3 Relativní odchylky propustnosti 54
5.4 Absolutní odchylky konstantního zpoždění 56
5.5 Relativní odchylky konstantního zpoždění 56
5.6 Naměřené hodnoty zpoždění pro normální rozložení 58
5.7 Histogram zpoždění pro normální rozložení 58
5.8 Naměřené hodnoty zpoždění pro pareto rozložení 59
5.9 Histogram zpoždění pro pareto rozložení 59
5.10 Naměřené hodnoty zpoždění pro pareto-normální rozložení 60
5.11 Histogram zpoždění pro pareto-normální rozložení 60
5.12 Naměřené hodnoty zpoždění pro rovnoměrné rozložení 61
5.13 Histogram zpoždění pro rovnoměrné rozložení 61
5.14 Absolutní odchylky ztrátovosti 63
5.15 Relativní odchylky ztrátovosti 63
5.16 Absolutní odchylky chybovosti 65
5.17 Relativní odchylky chybovosti 65
5.18 Absolutní odchylky duplikace 67

5.19 Relativní odchylky duplikace 67

Seznam tabulek

2.1 Srovnání metod pro síťové experimenty 27
3.1 Srovnání emulátorů síťových spojení 33
4.1 Souhrn použitých technologií 34
4.2 Popis metod ve třídě `TrafficControl` 36
4.3 Argumenty příkazu `netem` implementované v emulátoru 37
4.4 Argumenty filtru `tc u32` implementované v emulátoru 39
4.5 Abstraktní metody třídy `AbstractJMeterGuiComponent` 46
5.1 Výsledky měření propustnosti 53
5.2 Výsledky měření konstantního zpoždění 55
5.3 Výsledky měření ztrátovosti 62
5.4 Výsledky měření chybovosti 64
5.5 Výsledky měření duplikace 66
5.6 Průměrné relativní odchylky přenosových parametrů 68

Seznam výpisů

4.1 Značkování paketů pomocí `iptables` a následná klasifikace `tc fw` . . 37
4.2 Použití filtru `tc u32` s bitovým porovnáním 38
4.3 Použití filtru `tc u32` s vyšší úrovní zápisu 38
4.4 Metoda `calculateDiffServValue ze třídy Filter` 43
A.1 Sestavení projektu emulátoru 79
A.2 Přesunutí JAR souboru 79
A.3 Spuštění JMeteru 79
B.1 Skript pro instalaci potřebných nástrojů 80
B.2 Klient pro měření propustnosti 80
B.3 Server pro měření propustnosti 80
B.4 Klient pro měření zpoždění 81
B.5 Server pro měření zpoždění 82
B.6 Klient pro měření chybovosti 82
B.7 Server pro měření chybovosti 82
B.8 Klient pro měření ztrátovosti 83
B.9 Klient pro měření duplikace 83

Úvod

Síťové emulátory jsou softwarové nebo hardwarové nástroje používané k napodobení reálných síťových vlastností pro účely testování. Pomáhají vytvářet kontrolovaná prostředí se specifickými vlastnostmi, jako je zpoždění, ztrátovost nebo propustnost, a umožňují tak uživatelům testovat výkon aplikací a zařízení v reálných síťových scénářích. Vzhledem k rostoucím nárokům na testování robustnosti a výkonnosti síťových aplikací v různých provozních podmínkách nabývá síťová emulace na významu jako efektivní nástroj pro napodobení chování reálné sítě bez nutnosti složité fyzické infrastruktury.

Cílem této práce je návrh a implementace síťového emulátoru, který umožní řízeně ovlivňovat přenosové parametry paketových sítí využívajících protokolovou sadu TCP/IP. Výsledný modul je integrován do prostředí Apache JMeter a umožňuje emulaci různých síťových podmínek s cílem testovat chování síťových aplikací a služeb v odlišných provozních scénářích.

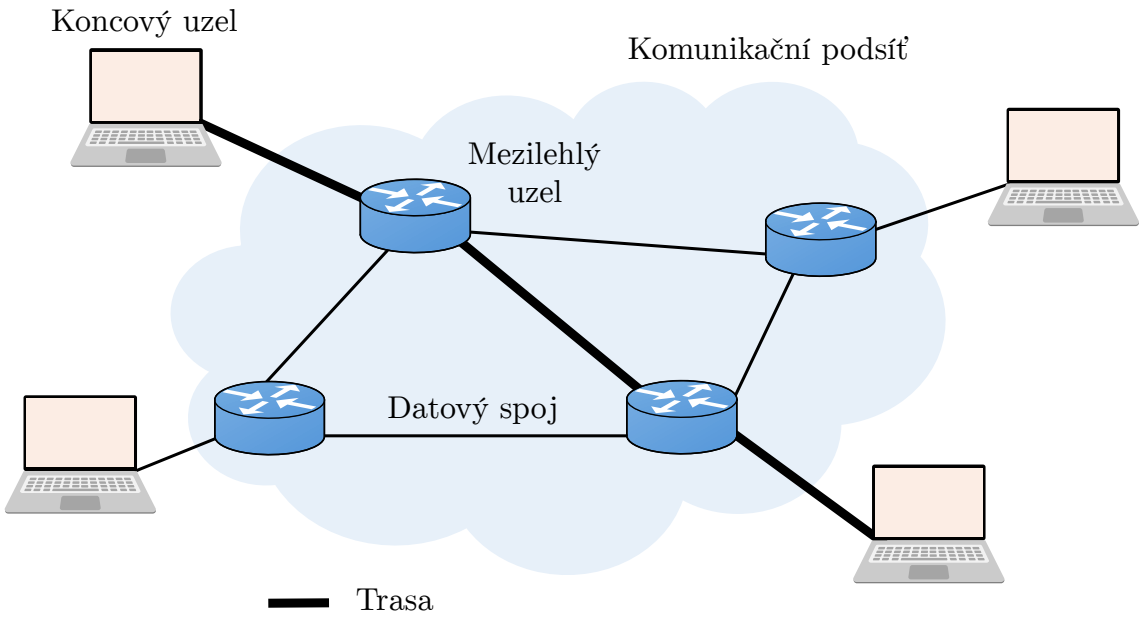
Práce je strukturována do pěti kapitol. První kapitola se věnuje analýze přenosových parametrů vhodných pro emulaci. Druhá kapitola popisuje metody síťového testování, do kterých spadá i síťová emulace. Třetí kapitola představuje existující síťové emulátory. Ve čtvrté kapitole je popsána samotná realizace emulátoru, od návrhu datových modelů po grafické uživatelské rozhraní. Poslední kapitola se zabývá měřením a vyhodnocením výsledků dosažených pomocí vytvořeného emulátoru.

1 Analýza přenosových parametrů

V sekci 1.1 jsou na schématu obecné komunikační sítě definovány klíčové pojmy, které jsou používány napříč touto kapitolou. Sekce 1.2 se věnuje rozdílu mezi šířkou pásma a maximální přenosovou rychlostí. V sekcích 1.3 až 1.7 jsou popsány vlastní parametry vhodné pro síťovou emulaci. Jako zdroje byly použity odborné publikace [1, 2, 3, 4, 5, 6, 7, 8], dokumenty IETF RFC [9, 10, 11] a dokument ČTÚ [12] popisující metodický postup pro měření a vyhodnocování přenosových parametrů v komunikačních sítích.

1.1 Základní prvky komunikační sítě

Obrázek 1.1 znázorňuje schéma obecné komunikační sítě, jehož popis následuje. **Mezilehlé uzly** jsou spínané systémy sloužící k propojení dvou nebo více datových spojů. **Datové spoje** jsou technické prostředky umožňující přenos dat mezi dvěma body. **Trasa** je sled datových spojů a mezilehlých uzlů, kterými prochází datová jednotka od odesílajícího koncového uzlu k přijímacímu koncovému uzlu. Každý koncový uzel je připojen k mezilehlému uzlu a množina všech mezilehlých uzlů se nazývá **komunikační podsíť**. Jako **koncové uzly** se označují zařízení připojená ke komunikační podsíti. Datové spoje vzájemně propojují mezilehlé uzly v komunikační podsíti a také k této podsíti připojují koncové uzly [1, 2, 12].



Obr. 1.1: Obecné schéma komunikační sítě

1.2 Šířka pásma a maximální přenosová rychlost

V kontextu analogových přenosů určuje šířka pásma B (anglicky bandwidth) rozdíl nejvyššího ($f_{\max}$) a nejnižšího ($f_{\min}$) kmitočtu přenášeného analogového signálu:

$$B = f_{\max} - f_{\min} \qquad [\text{Hz};\ \text{Hz},\ \text{Hz}]. \tag{1.1}$$

Maximální přenosová rychlost $r_{\max}$ (anglicky maximum bit rate) představuje v oblasti digitálních komunikací maximální množství dat, které je datový spoj schopen přenést za jednotku času. Někdy je též označována jako maximální datová rychlost (anglicky maximum data rate).

Šířku pásma a maximální přenosovou rychlost dává do souvislosti Nyqistův vzorec pro výpočet maximální dosažitelné přenosové rychlosti ideálního přenosového kanálu:

$$r_{\max} = 2 \cdot B \cdot \log_2 M \qquad [\text{bit/s};\ \text{Hz},\ -], \tag{1.2}$$

kde M je počet diskrétních signálových úrovní. Maximální přenosová rychlost přenosového kanálu tedy závisí na použité šířce pásma. To je zřejmě jeden z důvodů, proč bývá maximální přenosová rychlost datového spoje poněkud nepřesně označována rovněž jako šířka pásma[1] [1, 2, 3, 5, 6, 8].

1.3 Propustnost

Propustnost r (anglicky throughput) vyjadřuje naměřenou rychlost přenosu dat určitého přenosového systému. Tato rychlost může být okamžitá (naměřená za velmi krátký časový interval), případně průměrná (určena jako aritmetický průměr několika okamžitých hodnot). Na rozdíl od maximální přenosové rychlosti, která má teoretický význam, zahrnuje propustnost reálné síťové podmínky, jako jsou ztrátovost nebo zpoždění. Jak je naznačeno na obrázku 1.2, propustnost se může v čase měnit, zatímco maximální přenosová rychlost je vlastností daného přenosového systému a v čase je zpravidla konstantní.

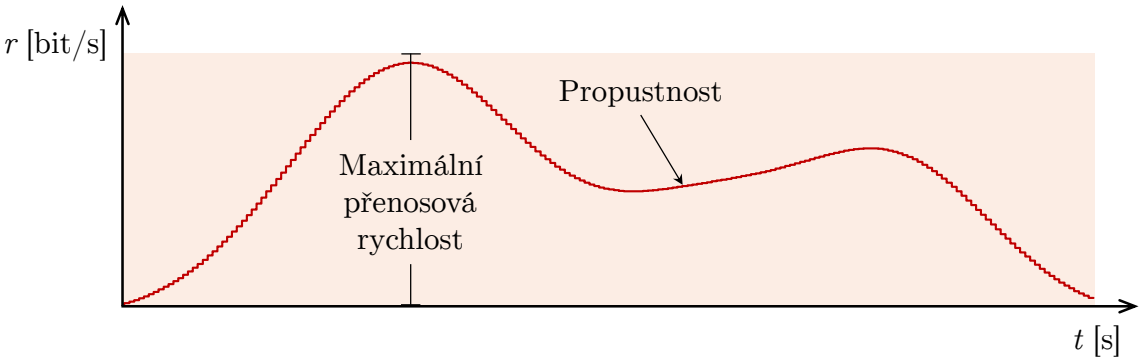
V některých případech se navíc uvádí tzv. reálná propustnost (anglicky goodput), která zohledňuje pouze množství užitečných dat přenášených aplikací a nezahrnuje režii spojenou s komunikačními protokoly a řízením přenosu [3, 4, 6, 8].

1.3.1 Propustnost uzlu

Propustnost dílčího uzlu r_{u} (anglicky nodal throughput) lze určit ze vztahu:

$$r_{\text{u}} = \frac{n}{d_{\text{u}}} \qquad [\text{bit/s};\ \text{bit},\ \text{s}], \tag{1.3}$$

[1] Zejména v anglicky psané odborné literatuře je pojem „bandwidth“ často používán jako synonymum pro maximální přenosovou rychlost.

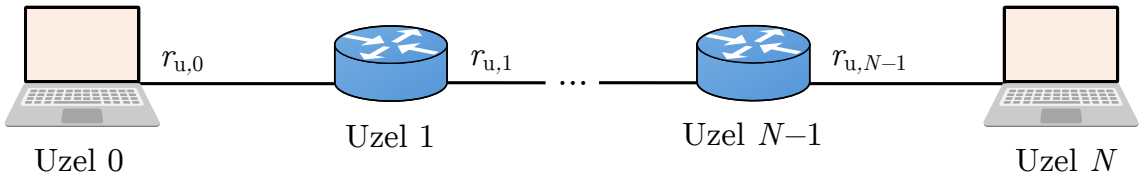


Obr. 1.2: Maximální přenosová rychlost a propustnost

kde n je množství přenášených dat a d_{u} je zpoždění uzlu popsané v podsekci 1.4.2.

1.3.2 Propustnost trasy

Na trase od odesílatele k příjemci paket typicky prochází více mezilehlými uzly, přičemž každý uzel může mít rozdílnou propustnost, jak je znázorněno na následujícím obrázku 1.3.



Obr. 1.3: Propustnost trasy

Paket přijatý určitým uzlem nemusí být z tohoto uzlu odeslán stejnou přenosovou rychlostí, jakou byl přenášen v předchozím uzlu. Místo toho může být nejdříve zařazen do tzv. fronty a následně odeslán nižší rychlostí. Propustnost trasy r_{t} (anglicky end-to-end throughput) je pak omezena rychlostí nejpomalejšího uzlu na trase:

$$r_{\mathrm{t}} = \min\left(r_{\mathrm{u},0},\ \ldots, r_{\mathrm{u},N-1}\right) \qquad [\mathrm{bit/s}; \mathrm{bit/s}], \tag{1.4}$$

kde $r_{\mathrm{u},0},\ \ldots, r_{\mathrm{u},N-1}$ jsou propustnosti jednotlivých uzlů na trase od odesílatele k příjemci. Uzel s nejnižší propustností se nazývá úzké hrdlo trasy (anglicky bottleneck) [3, 4].

1.4 Zpoždění paketu

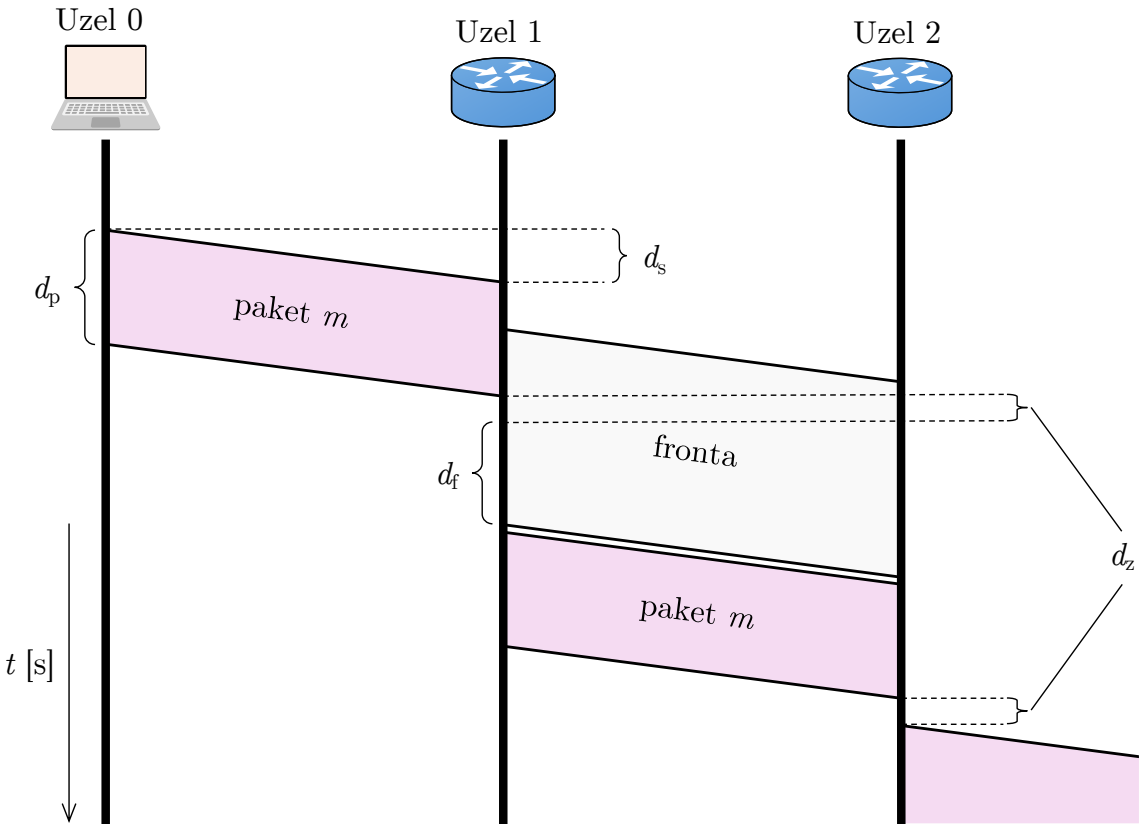
Zpoždění paketu d (anglicky packet delay) definuje dobu, za kterou je mezi dvěma koncovými uzly přenesen celý paket od okamžiku, kdy byl z odesílajícího koncového uzlu odeslán jeho první bit. Zpoždění m-tého paketu lze vyjádřit jako:

$$d_m = t_{\mathrm{p},m} - t_{\mathrm{o},m} \qquad [\mathrm{s};\ \mathrm{s},\ \mathrm{s}], \tag{1.5}$$

kde $t_{\mathrm{p},m}$ je čas přijetí m-tého paketu přijímacím koncovým uzlem a $t_{\mathrm{o},m}$ je čas odeslání m-tého paketu odesílajícím koncovým uzlem. Takto definované zpoždění je tedy tzv. koncové (anglicky end-to-end, one-way). Detailněji je popsáno v podsekci 1.4.3 věnující se zpoždění trasy.

1.4.1 Složky zpoždění paketu

Zpoždění paketu se skládá ze zpoždění signálu d_{s}, zpoždění přenosu d_{p}, zpoždění zpracování d_{z} a zpoždění ve frontě d_{f}. Tyto složky jsou znázorněny na obrázku 1.4 a popsány v následujícím textu.



Obr. 1.4: Složky zpoždění paketu

Zpoždění signálu

Zpoždění signálu d_{s} (anglicky propagation delay) je doba šíření signálu po datovém spoji mezi odesílající a přijímací stranou. Zpoždění signálu je definováno vztahem:

$$d_{\mathrm{s}} = \frac{l}{v} \qquad [\mathrm{s};\ \mathrm{m},\ \mathrm{m/s}], \tag{1.6}$$

kde l je délka fyzického spojení mezi komunikujícími stranami a v je rychlost šíření signálu, jež se liší v závislosti na použitém fyzickém médiu. U bezdrátových přenosů se blíží rychlosti světla ve vakuu ($3 \cdot 10^8$ m/s) zatímco u metalických a optických vedení má hodnotu přibližně $2,3 \cdot 10^8$ m/s, respektive $2 \cdot 10^8$ m/s [1, 3, 4, 7, 8].

Zpoždění přenosu

Zpoždění přenosu d_{p} (anglicky transmission delay) je čas potřebný k vložení paketu na datový spoj. Pokud je první bit paketu vložen v čase t_1 a poslední bit paketu v čase t_2, zpoždění přenosu nabývá hodnoty $t_2 - t_1$. Podle vzorce pro výpočet zpoždění přenosu je zřejmé, že kratší pakety a větší propustnosti vedou k nižší hodnotě zpoždění:

$$d_{\mathrm{p}} = \frac{m}{r} \qquad [\mathrm{s};\ \mathrm{bit},\ \mathrm{bit/s}], \tag{1.7}$$

kde m je velikost paketu a r je propustnost datového spoje. Zpoždění přenosu se v praxi pohybuje v řádu mikrosekund až milisekund [3, 4, 7, 8].

Zpoždění zpracování

Zpoždění zpracování d_{z} (anglicky processing delay) označuje čas, který potřebuje mezilehlý uzel nebo koncový uzel na zpracování paketu. Toto zpracování obnáší přijetí paketu ze vstupního portu, analýzu jeho záhlaví, detekci chyb a další úkony. V případě mezilehlého uzlu následuje odeslání paketu skrze některý z jeho výstupních síťových portů, v případě koncového uzlu dojde k předání zapouzdřené datové jednotky protokolu vyšší vrstvy. Jelikož zpoždění zpracování může pro jednotlivé pakety nabývat různých hodnot, zpravidla se udává jako průměrná hodnota. Velikost zpoždění zpracování je závislé na výkonu použitého hardwaru [3, 4].

Zpoždění ve frontě

Zpoždění ve frontě d_{f} (anglicky queueing delay) nastává při přepínání paketů mezilehlým uzlem. Každý mezilehlý uzel má vstupní frontu pro pakety čekající na zpracování a výstupní frontu pro pakety čekající na odeslání. Před samotným přenosem mezilehlý uzel nejprve shromáždí všechny bity daného paketu, uloží je do vyrovnávací paměti, určí výstupní port a poté čeká na vhodný okamžik k odeslání. V nejjednodušším případě je paket zařazen do výstupní fronty typu FIFO a čeká,

až budou odeslány pakety, které jsou ve frontě před ním. Celkové zpoždění ve frontě pro konkrétní paket lze vyjádřit jako:

$$d_{\mathrm{f}} = d_{\mathrm{e}} + d_{\mathrm{x}} \qquad [\mathrm{s};\ \mathrm{s},\ \mathrm{s}], \tag{1.8}$$

kde d_{e} je doba, kterou paket stráví ve vstupní frontě a d_{x} je doba, kterou paket stráví ve výstupní frontě. Při odhadu konkrétních hodnot zpoždění ve frontě se obvykle využívají statistické ukazatele, jako je průměrné zpoždění, variace zpoždění nebo pravděpodobnost, že zpoždění překročí určitou hodnotu [3, 4, 6].

1.4.2 Zpoždění uzlu

Zpoždění uzlu d_{uzel} (anglicky nodal delay) je součtem zpoždění signálu, zpoždění přenosu, zpoždění zpracování a zpoždění ve frontě [4]:

$$d_{\mathrm{u}} = d_{\mathrm{s}} + d_{\mathrm{p}} + d_{\mathrm{z}} + d_{\mathrm{f}} \qquad [\mathrm{s};\ \mathrm{s},\ \mathrm{s},\ \mathrm{s},\ \mathrm{s}]. \tag{1.9}$$

Ačkoliv jsou zpoždění signálu a zpoždění přenosu ovlivněna převážně příslušným datovým spojem a jeho implementací, obvykle představují minoritní složku ze všech popsaných zpoždění. Proto jsou v rámci této práce přidružena k celkovému zpoždění uzlu.

1.4.3 Zpoždění trasy

Zpoždění trasy d_{t} (anglicky end-to-end delay nebo one-way delay) udává celkové zpoždění paketu od odesílatele k příjemci. Za předpokladu, že jsou dílčí zpoždění všech uzlů na trase shodná, lze zpoždění trasy vypočítat jako:

$$d_{\mathrm{t}} = (N+1) \cdot (d_{\mathrm{s}} + d_{\mathrm{p}} + d_{\mathrm{z}}) + N \cdot d_{\mathrm{f}} \qquad [\mathrm{s};\ -,\ \mathrm{s},\ \mathrm{s},\ \mathrm{s},\ -,\ \mathrm{s}], \tag{1.10}$$

kde N představuje počet mezilehlých uzlů. Pokud je na trase N mezilehlých uzlů, existuje $(N+1)$ datových spojů. To znamená, že zpoždění trasy zahrnuje $(N+1)$ přenosových zpoždění určených N mezilehlými uzly a odesílatelem, $(N+1)$ zpoždění signálu určených $(N+1)$ datovými spoji, $(N+1)$ zpoždění zpracování určených N mezilehlými uzly a příjemcem a pouze N zpoždění ve frontě určených N mezilehlými uzly [3, 4].

1.4.4 Obousměrné zpoždění

Obousměrné zpoždění d_{o} (anglicky round-trip delay nebo two-way delay) definuje souhrnné zpoždění paketu od odesílatele k příjemci a zpět. Za předpokladu, že má

paket na cestě tam i zpět stejnou trasu a síťové podmínky na trase jsou neměnné, lze obousměrné zpoždění jednoduše vyjádřit jako dvojnásobek zpoždění trasy [1]:

$$d_{\mathrm{o}} = 2 \cdot d_{\mathrm{t}} \qquad [\mathrm{s};\ \mathrm{s}]. \tag{1.11}$$

V dokumentu RFC 958, který popisuje protokol NTP (Network Time Protocol) určený k synchronizaci vnitřních hodin uzlů v počítačové síti, je obousměrné zpoždění definováno jako:

$$d_{\mathrm{o}} = (t_4 - t_1) - (t_3 - t_2), \tag{1.12}$$

kde t_1 je čas, kdy klient odeslal požadavek, t_2 je čas příjmu požadavku serverem, t_3 je čas, kdy server odeslal odpověď a t_4 je čas příjmu odpovědi klientem. Čas $t_4 - t_1$ měřený klientem je doba od odeslání požadavku po přijetí odpovědi a čas $t_3 - t_2$ je doba zpracování požadavku serverem. Tato definice implicitně předpokládá, že zpoždění trasy tvoří polovinu obousměrného zpoždění, tedy že platí rovnost 1.11 [9].

1.5 Kolísání zpoždění paketů

Kolísání zpoždění paketů j (anglicky packet delay variation, jitter), někdy označovaný také jako variace zpoždění, popisuje rozdíl mezi referenčním časem doručení paketu $t_{\mathrm{r},n}$ a jeho skutečným časem doručení $t_{\mathrm{s},n}$ v přijímacím koncovém uzlu. Referenční čas doručení n-tého paketu je možné vyjádřit jako [12]:

$$t_{\mathrm{r},n} = t_{\mathrm{s},m} + \Delta t_{m,n} \qquad [\mathrm{s};\ \mathrm{s},\ \mathrm{s}], \tag{1.13}$$

kde $n = m + 1$, $t_{\mathrm{s},m}$ je skutečný čas doručení m-tého paketu a $\Delta t_{m,n}$ je interval mezi odesláním m-tého a n-tého paketu odesílajícím koncovým uzlem. Kolísání zpoždění n-tého paketu lze pak zapsat ve tvaru [12]:

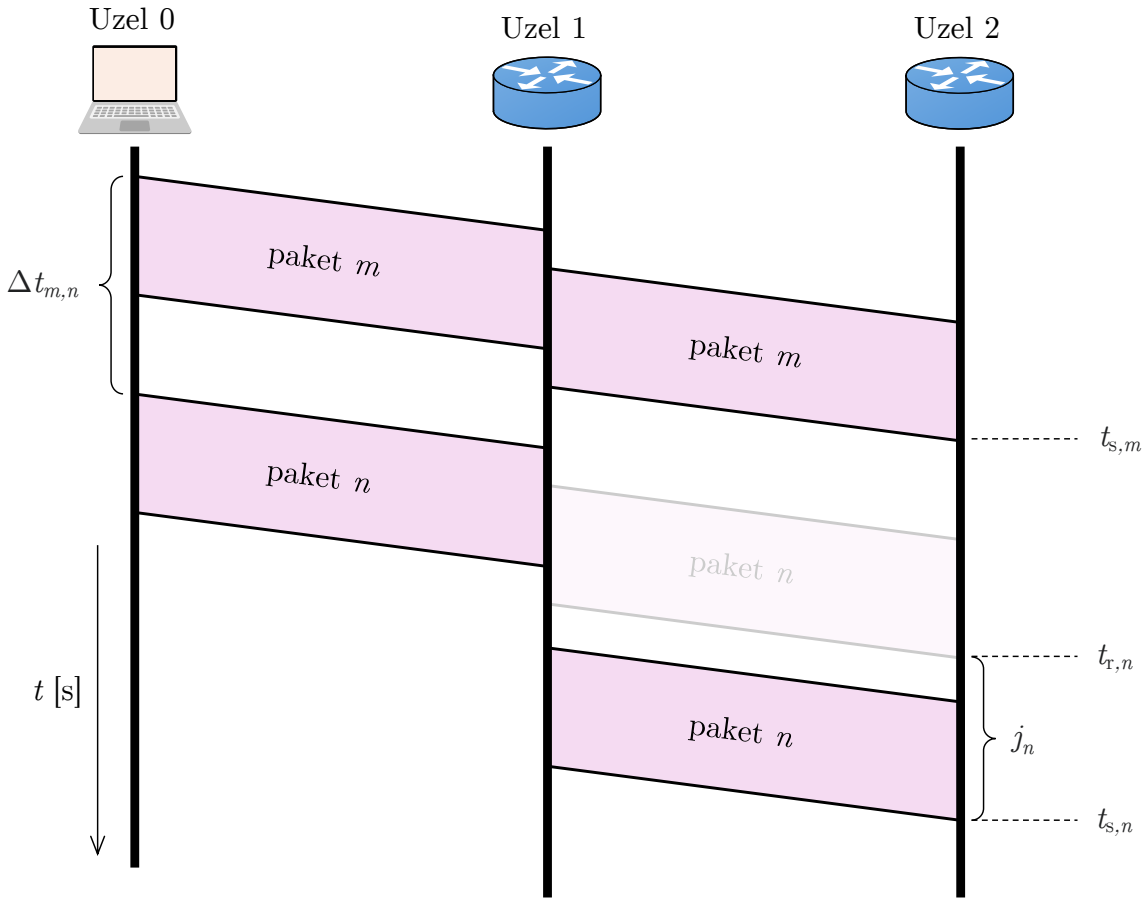
$$j_n = |t_{\mathrm{s},n} - t_{\mathrm{r},n}| \qquad [\mathrm{s};\ \mathrm{s},\ \mathrm{s}]. \tag{1.14}$$

Výše popsaná situace je znázorněna na obrázku 1.5.

1.6 Ztrátovost paketů

Ztrátovost paketů L (anglicky packet loss ratio) udává, jak velká část z celkového počtu odeslaných paketů nebyla úspěšně doručena. Je dána vztahem:

$$L = \frac{n_{\mathrm{z}}}{n_{\mathrm{o}}} \cdot 100 \qquad [\%;\ -,\ -], \tag{1.15}$$



Obr. 1.5: Kolísání zpoždění paketů

kde n_{z} je počet ztracených paketů a n_{o} je počet odeslaných paketů. Nejčastější příčinou vyšší ztrátovosti je přetížení sítě, při kterém se v mezilehlých uzlech tvoří fronty, ve kterých pakety čekají na zpracování a odeslání. Pokud dané zařízení vyčerpá své výpočetní a paměťové prostředky, jsou další příchozí pakety zahazovány. Z uživatelského hlediska se ztrátovost projevuje zpomalením provozované služby nebo úplným výpadkem síťového připojení. Podobně jako u kolísání zpoždění paketů, ztrátovostí paketů nejvíce trpí aplikace, které vyžadují přenos dat v reálném čase [3, 4, 5, 7, 8, 12].

1.7 Chybovost paketů

Chybovost paketů E (anglicky packet error ratio) udává, jak velká část z celkového počtu odeslaných paketů byla doručena s chybou. Je dána vztahem:

$$E = \frac{n_{\mathrm{ch}}}{n_{\mathrm{o}}} \cdot 100 \qquad [\%;\ -,\ -], \tag{1.16}$$

kde n_{ch} je počet doručených chybných paketů [12].

1.8 Duplikace paketů

Duplikace paketů D (anglicky packet duplication ratio) udává, jak velká část z celkového počtu odeslaných paketů byla přijata duplicitně. Lze ji definovat jako:

$$D = \frac{n_{\mathrm{d}}}{n_{\mathrm{p}}} \cdot 100 \qquad [\%;\ -,\ -], \tag{1.17}$$

kde n_{d} je počet doručených duplicitních paketů a n_{p} je počet přijatých paketů. Podle RFC 5560 jsou pakety považovány za duplicitní tehdy, když obsahují shodná informační pole a zároveň byly odeslány ze stejné zdrojové adresy na stejnou cílovou adresu. Takové pakety jsou z pohledu příjemce zaměnitelné [10].

1.9 Záměna pořadí paketů

Záměna pořadí paketů Z (anglicky packet reordering ratio) určuje, jak velká část z přijatých paketů byla doručena ve špatném pořadí. RFC 4737 ji definuje jako:

$$Z = \frac{n_{\mathrm{z}}}{n_{\mathrm{p}}} \cdot 100 \qquad [\%;\ -,\ -], \tag{1.18}$$

kde n_{z} je počet paketů, které byly příjemci doručeny v nesprávném pořadí. V případě duplicitních paketů je započítána pouze první kopie [11].

2 Metody síťového testování

Při návrhu a testování počítačových sítí, případně vývoji nových síťových protokolů, jsou využívány specifické softwarové nástroje. Tyto nástroje lze rozdělit do třech oblastí – síťové simulátory, síťová testovací prostředí a síťové emulátory. Vzhledem k značné rozsáhlosti je pro každou ze zmíněných oblastí uvedena pouze vybraná metoda, která je v současnosti často využívána. V případě síťové simulace jsou to diskrétní simulátory, u síťových testovacích prostředí jsou to softwarově definované sítě a v případě síťové emulace je to emulace síťového spojení. Metodám síťového testování se velmi podrobně věnuje studie [16], která do značné míry sloužila jako zdroj pro tuto kapitolu. Síťovým emulátorům je věnována celá následující kapitola.

2.1 Síťová simulace

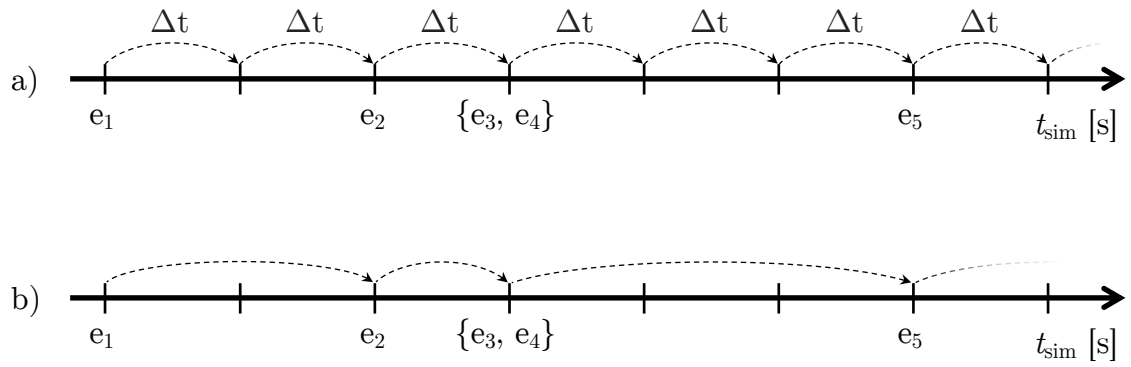
Síťová simulace (anglicky network simulation) poskytuje reprodukovatelné a kontrolované prostředí, ve kterém lze dynamicky měnit síťové parametry a zkoumat jejich dopad na chování celé simulované sítě. Díky této technice je možné vytvářet prototypy reálných sítí a vykonávat na nich širokou škálu síťových experimentů. Síťová simulace je realizována pomocí síťového simulátoru. Jedná se o počítačový software, který provádí výpočty vztahů mezi síťovými zařízeními i aplikačními entitami se snahou co nejvíce napodobit chování reálného systému. [16, 18, 24].

Síťový simulátor běží pouze v uživatelském prostoru operačního systému. Při každém spuštění daného experimentu produkuje stejné výsledky, a to nezávisle na specifikacích stroje, na kterém je simulace prováděna. Díky tomu jsou experimenty velice snadno přenositelné i na jiná zařízení za dosažení totožných výstupů. Další výhodou síťových simulátorů je vysoká míra škálovatelnosti simulací a nízké náklady na jejich opakované provádění [16, 21].

2.1.1 Diskrétní simulátory

Počítačové simulace lze obecně rozdělit na diskrétní a spojité. Spojité simulace se obvykle využívají při modelování přírodních procesů, přičemž časové změny jsou zde popisovány diferenčními rovnicemi. Tyto simulace bývají výpočetně náročné. V kontextu počítačových sítí jsou typicky simulovány pouze některé klíčové jevy, jako je odeslání/přijetí paketu, změna stavu zařízení či zahájení/ukončení spojení. Tyto jevy jsou diskrétní povahy – dějí se v určitých okamžicích a systém lze v mezilehlých časových intervalech považovat za neměnný. Z tohoto důvodu jsou síťové simulace prováděny téměř výhradně diskrétním způsobem. Diskrétní síťové simulátory se dále dělí

na simulátory řízené diskrétním časem a simulátory řízené diskrétními událostmi. Na obrázku 2.1 je znázorněna odlišnost v jejich fungování [16, 19, 20].



Obr. 2.1: Diskrétní síťové simulátory: a) simulátor řízený diskrétním časem, b) simulátor řízený diskrétními událostmi

Simulátor řízený diskrétním časem

Simulátor řízený diskrétním časem (anglicky discrete time simulator) během simulace postupně prochází časové úseky (na obrázku označeny jako Δt), přičemž události ($e_1, e_2, \ldots$) mohou být vykonány během každého časového úseku. Příkladem síťového simulátoru řízeného diskrétním časem je program Packet Tracer [16].

Simulátor řízený diskrétními událostmi

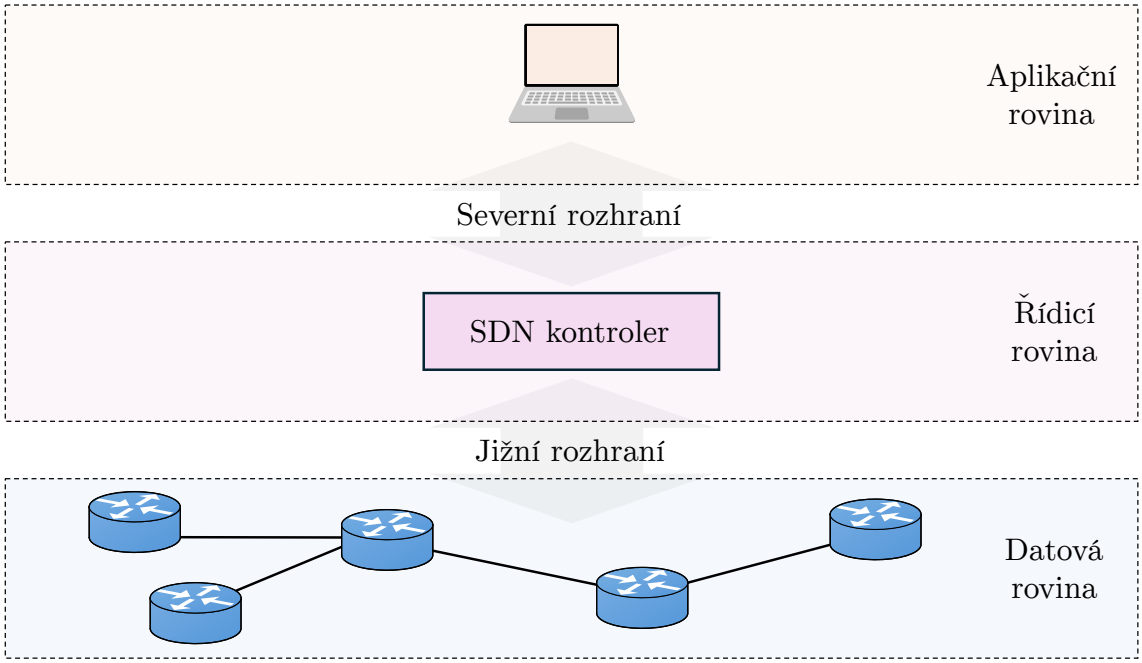
Simulátor řízený diskrétními událostmi (anglicky discrete event simulator) prochází naplánované události a posouvá čas simulace (t_{sim}) po vykonání každé dílčí události. Příkladem síťového simulátoru řízeného diskrétními událostmi je software ns-3 [16].

2.2 Síťové testovací prostředí

Síťová testovací prostředí (anglicky network testbed) využívají skutečný hardware alokovaný výhradně pro experimenty. Tento hardware může být i sdílený mezi více uživateli – v takovém případě jsou jednotlivé experimenty izolovány, aby se zamezilo jejich vzájemnému ovlivňování. Tento přístup zajišťuje vysokou realističnost experimentů a současně poskytuje dostatečný výpočetní výkon pro jejich vykonání. Testovací prostředí jsou hojně využívána v oblasti výzkumu, zejména při návrhu nových síťových architektur a protokolů, ale např. i při vývoji algoritmů strojového učení, blockchainových technologií, kybernetické bezpečnosti apod. Mnoho těchto testovacích prostředí implementuje principy tzv. softwarově definovaných sítí [16, 22, 23].

2.2.1 Softwarově definované sítě

Softwarově definované sítě (anglicky Software-Defined Networking, SDN), na rozdíl od konvenčních sítí, oddělují tzv. řídicí rovinu (anglicky control plane) a datovou rovinu (anglicky data plane), čímž zajišťují globalizovaný pohled a centralizovanou kontrolu. Síťová zařízení si mezi sebou například nevyměňují směrovací informace, jako je tomu u klasických sítí. O pravidla pro přenos dat se stará tzv. SDN kontroler. Na kontroler lze prostřednictvím aplikační roviny (anglicky application plane) snadno implementovat softwarově řízené metody, algoritmy a politiky pro správu celé SDN sítě. Princip softwarově definované sítě je na obrázku 2.2. K interní řídicí komunikaci mezi systémovými komponentami se používají specializovaná rozhraní. Tato rozhraní transformují konfigurační příkazy pro řízení hardwaru za účelem provedení požadovaných akcí. Rozhraní používaná ke komunikaci s aplikační rovinou se nazývá severní rozhraní, zatímco rozhraní používaná ke komunikaci směrem k datové rovině se nazývá jižní rozhraní [16, 22].



Obr. 2.2: Princip softwarově definované sítě

2.3 Síťová emulace

Síťová emulace (anglicky network emulation) je hybridní přístup, který kombinuje reálné síťové prvky a simulované nebo abstraktní prvky. Síťová emulace se často využívá k rychlému vývoji prototypů a testování chování aplikací v různých síťových

podmínkách. Události v síťovém emulátoru probíhají spojitě a využívají skutečný protokolový zásobník dostupný v hostitelském operačním systému. Síťové emulátory jsou praktickými testovacími nástroji díky své přenositelnosti mezi více zařízeními. Mohou být však limitovány hardwarovými prostředky hostitelských stanic. Síťová emulace představuje alternativu k síťové simulaci a zároveň nabízí cenově dostupnější řešení ve srovnání s testovacím prostředím. Perspektivní metodou síťové emulace je tzv. emulace síťového spojení [16, 24].

2.3.1 Emulace síťového spojení

Principem emulace síťového spojení (anglicky network link emulation) je změna chování příchozích a odchozích paketů na daném fyzickém nebo virtuálním síťovém rozhraní. Emulátory síťových spojení jsou poměrně jednoduché, však účinné nástroje, které dovolují zavést mj. zpoždění, ztrátovost či chybovost do síťového provozu. Za tímto účelem využívají služby operačních systémů s otevřeným kódem, jako jsou FreeBSD nebo Linux. [16].

Přehled konkrétních emulátorů síťových spojení je uveden v sekci 3.2.

2.4 Srovnání testovacích metod

Tabulka 2.1 srovnává tři hlavní metody pro síťové testování z hlediska klíčových vlastností. Síťová simulace vyniká flexibilitou, škálovatelností a reprodukovatelností, ale má omezenou realističnost. Emulace nabízí vyvážený kompromis s dobrou realističností a cenovou dostupností. Síťová testovací prostředí poskytují nejvyšší míru realističnosti, avšak za cenu nižší flexibility, škálovatelnosti a vyšších finančních nákladů.

Tab. 2.1: Srovnání metod pro síťové experimenty

	Síťová simulace	**Síťová emulace**	**Testovací prostředí**
Realističnost	Omezená	Dobrá	Výborná
Flexibilita	Výborná	Dobrá	Omezená
Škálovatelnost	Výborná	Částečná	Omezená
Reprodukovatelnost	Výborná	Částečná	Částečná
Cenová dostupnost	Dobrá	Výborná	Dobrá

3 Přehled síťových emulátorů

Síťové emulátory jsou klíčovými nástroji pro výzkum a vývoj síťových protokolů a aplikací. Díky emulaci sítě je možné provádět testy realistických scénářů v kontrolovaném prostředí, což s fyzickými zařízeními obecně není možné. Emulátory, oproti simulátorům, pracují s reálným síťovým provozem namísto simulovaného, díky čemuž lze přesněji napodobit chování skutečné počítačové sítě [14, 30].

3.1 Emulátory založené na virtualizaci

Platformy založené na virtualizaci mají za cíl poskytnout konzistentní a reprodukovatelné prostředí pro implementaci testovacích scénářů. Při virtualizaci je vytvořeno testovací prostředí, ve kterém může uživatel velmi snadno provádět experimenty a je současně izolován od hostitelského systému. To umožňuje otestovat a vyhodnotit dopad takřka libovolných síťových konfigurací na výsledné chování celé sítě a jednotlivých aplikací [16].

3.1.1 CORE

Common Open Research Emulator (CORE) je síťový emulátor fungující v reálném čase zaměřený na modelování hybridních topologií. CORE usnadňuje integraci virtualizovaných instancí s reálným hardwarem pomocí síťového zásobníku FreeBSD, což usnadňuje plánování, testování a vývoj síťových aplikací a protokolů. Cílem platformy je snížit potřebu drahého hardwaru pro provozování síťových experimentů. Autoři vyzdvihují škálovatelnost CORE na příkladu instalace více než stovky virtuálních emulovaných uzlů. Tyto uzly mohou běžet na běžném serveru [16, 17].

3.1.2 CML

Cisco Modeling Labs (CML) je emulační platforma, která umožňuje navrhovat sítě založené na Cisco pomocí virtuálních verzí operačních systémů Cisco. CML se skládá ze serveru CML a klienta CML. Společně poskytují experimentální prostředí, které usnadňuje rychlý a efektivní návrh, konfiguraci, vizualizaci a simulaci síťových topologií [16, 25].

Server CML je linuxová distribuce zabalená ve formátu VMware Open Virtual Appliance (`.ova`) obsahující všechny podpůrné soubory. Server CML je sdílený prostředek, který používají koncoví uživatelé ke spouštění backendových funkcí, jako jsou zaváděcí konfigurace směrovačů, spouštění směrovačů pro provoz s určenými operačními systémy a úpravy a testování konfigurací [25].

Klient CML je multiplatformní uživatelské rozhraní pro vytváření a úpravy návrhů sítí pomocí grafického editoru a simulaci těchto topologií sítí na serveru CML. Umožňuje z uživatelského rozhraní přímo interagovat s běžícími simulacemi a také poskytuje funkce pro uložení vlastních konfigurací směrovačů před spuštěním simulace [25].

3.2 Emulátory síťových spojení

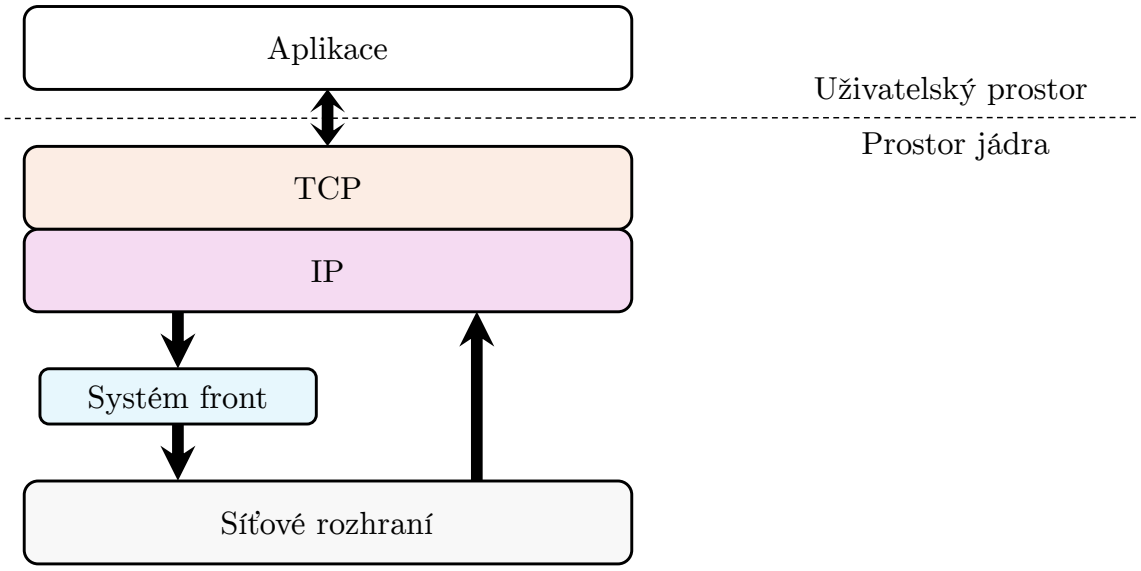
Emulátory síťových spojení využívají softwarové nástroje poskytované operačním systémem tak, aby se emulované síťové prvky chovaly, jako by byly ve skutečné síti. Tímto chováním je myšleno např. zpožďování nebo zahazování paketů na určitém síťovém rozhraní podle nastavených pravidel.

3.2.1 NetEm

NetEm je v současnosti jedním z nejpoužívanějších emulátorů síťových spojení. Skládá se ze dvou částí – modulu jádra implementujícího frontu pro tzv. systém front (anglicky queuing discipline) a nástroje pro příkazový řádek sloužícího k jeho konfiguraci. Modul byl oficiálně začleněn do linuxového jádra od verze 2.6.8[1] a konfigurační nástroj je součástí kolekce `iproute2`, konkrétně balíčku Traffic Control (zkráceně tc). Požadavky konfiguračního nástroje jsou převáděny do specifického formátu zprávy NetEm a přenášeny prostřednictvím socketového rozhraní Netlink. Systém front je umístěn mezi výstup zásobníku síťových protokolů a vstup síťového rozhraní, jak je znázorněno na obrázku 3.1. Výchozí frontou je jednoduchá fronta paketů FIFO [13, 14, 15].

Systém front je objekt tvořený dvěma základními rozhraními. První rozhraní zajišťuje zařazování odchozích paketů do fronty (anglicky enqueue) a druhé rozhraní slouží k jejich výběru a odeslání (anglicky dequeue). Systém front určuje, které pakety mají být odeslány podle svého aktuálního stavu a podle nastavených parametrů. Fronty lze rozdělit na nehierarchické (anglicky classless) a hierarchické (anglicky classful). Nehierarchické fronty fungují samostatně a nemají žádnou strukturu. Typickými příklady nehierarchických front v operačním systému Linux jsou FIFO a PFIFO. Naopak hierarchické fronty tvoří stromovou strukturu složenou z více tříd, které dědí parametry ze svých nadřazených tříd a mohou je dále rozšiřovat. Tato struktura umožňuje flexibilnější a přesnější řízení přenosu dat. Příkladem takové fronty v Linuxu je HTB, který nahradil starší frontu CBQ [13, 15].

[1] U starších verzí (2.4.x) lze modul aktivovat v konfiguraci jádra.



Obr. 3.1: Pozice systému front v Linuxovém jádře

3.2.2 DummyNet

DummyNet byl vyvinutý na konci 90. let jako součást unixového operačního systému FreeBSD za cílem zachycování TCP provozu a vyhodnocování výkonu síťových protokolů. Přestože byl tento modul původně navržený především pro výzkumné a didaktické účely, vzbudil velký zájem o jeho využití jako nástroje pro úpravu šířky pásma a zpoždění v síťových serverech [26, 27].

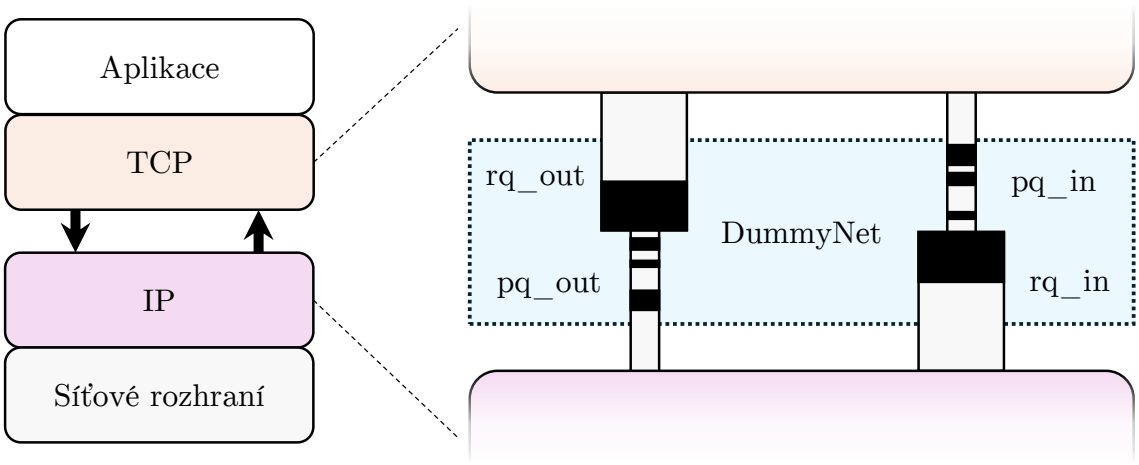
Na obrázku 3.2 je naznačen princip fungování DummyNet. Za účelem emulace síťového spojení DummyNet vkládá do toku dat následující dva segmenty:

- tzv. směrovače (anglicky routers) s nastavenou omezenou velikostí fronty a nastaveným řízením fronty (na obrázku `rq_out` a `rq_in`),
- tzv. roury (anglicky pipes) s nastavenými parametry emulace (na obrázku `pq_out` a `pq_in`).

Nejjednodušší konfigurace lze dosáhnout použitím jednoho směrovače a jedné roury. Tyto prvky jsou v DummyNet modelovány dvojicí front `rq` a `pq`, které jsou umístěny do protokolového zásobníku v modulu TCP/IP. DummyNet podporuje emulaci v obou směrech komunikace a pro každý směr je zapotřebí jedna taková dvojice front `rq`/`pq`.

3.2.3 NISTNet

Emulátor NISTNet je rozšíření Linuxového jádra, které poskytuje komplexní možnosti emulace zpoždění, ztrátovosti a dalších parametrů. Na obrázku 3.3 je naznačena



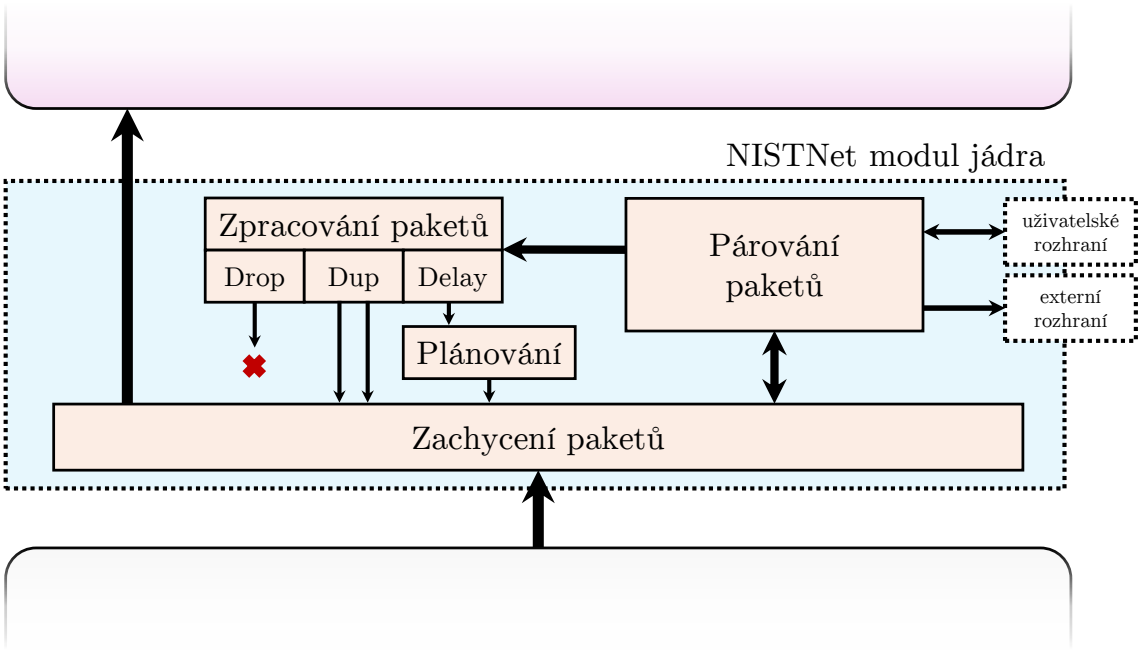
Obr. 3.2: Princip emulátoru DummyNet

jeho architektura. Jelikož je NISTNet volně dostupný, mnoho jeho funkcí bylo použito v NetEm. NISTNet pracuje s příchozími pakety ještě předtím, než jsou zařazeny do protokolového zásobníku. Stejně jako Dummynet provádí NISTNet veškeré filtrování a řazení do fronty. Jako časový zdroj používá hodiny reálného času (anglicky Real-Time Clock, RTC). Jeho fungování je postaveno na tabulce záznamů. Každý záznam se skládá z následujících tří částí:

- Pravidla definující, které pakety budou tímto záznamem zachyceny,
- sada úprav, které budou na zachycené pakety aplikovány,
- statistiky o paketech odpovídajících tomuto záznamu.

NISTNet dovoluje načíst tisíce takových záznamů najednou, přičemž každý záznam může mít přiřazenou vlastní sadu úprav. Díky tomu lze emulovat širokou škálu síťových podmínek i v malém laboratorním prostředí sestávajícím ze dvou nebo tří zařízení. Záznamy mohou být přidávány a upravovány ručně nebo automaticky i během provozu emulátoru. NISTNet umožňuje zachytit pakety na základě většiny důležitých polí IP protokolu a vyšších protokolů. Specifikovat lze například IP adresy odesílatele a příjemce, identifikátor protokolu vyšší vrstvy, typ služby, porty odesílatele a příjemce, typ a kód ICMP zpráv apod. Algoritmus emulátoru je podle autorů navržen tak, aby bez problémů fungoval za plné přenosové rychlosti i na tisících záznamů [13, 28].

Podle dostupných informací je NISTNet implementován v jádře 2.4, ale není již k dispozici v jádře 2.6. Jeho vývoj byl ukončen a většina funkcionalit byla začleněna do nástroje NetEm [15].



Obr. 3.3: Architektura emulátoru NISTNet

3.2.4 Srovnání emulátorů síťových spojení

V tabulce 3.1 je srovnání popsaných emulátorů síťových spojení. DummyNet, NIST Net a Netem používají stejný princip emulace. Zachytávají příchozí nebo odchozí pakety a pomocí sady pravidel a front ukládají pakety, dokud není určeno, zda mohou být uvolněny do operačního systému (v případě příchozích paketů) nebo do sítě (v případě odchozích paketů). NetEm zachytává pakety na výstupním toku předtím, než jsou předány síťovému rozhraní. NISTNet odchytává i příchozí pakety, než dosáhnou protokolového zásobníku. DummyNet umožňuje zavést konstantní zpoždění, nenabízí však možnost volby pravděpodobnostního rozložení, která jsou k dispozici v NISTNet a NetEm. DummyNet je tedy dostačující pro jednoduché experimenty v emulované síti, nedokáže však dostatečně přesně imitovat chování skutečné sítě, jako je tomu u NISTNet a NetEm [13, 29, 30].

Tab. 3.1: Srovnání emulátorů síťových spojení

	DummyNet	**NISTNet**	**NetEm**
Platforma	FreeBSD	Linux 2.4	Linux 2.6
Místo zachycení	Vstup i výstup	Pouze vstup	Pouze výstup
Časový zdroj	Systém	RTC	Systém
Zpoždění	Ano	Ano	Ano
Kolísání zpoždění	Ne	Ano (včetně korelace)	Ano (včetně korelace)
Propustnost	Ano	Ano	Ano
Ztrátovost	Ano (bez korelace)	Ano (včetně korelace)	Ano (včetně korelace)
Záměna pořadí	Ne	Ano (včetně korelace)	Ano (včetně korelace)
Duplikace	Ne	Ano (včetně korelace)	Ano (včetně korelace)
Chybovost	Ne	Ano (včetně korelace)	Ano (včetně korelace)

4 Realizace emulátoru

Na základě analýzy síťových emulátorů a jejich srovnání v předchozí kapitole se jako vhodný pro implementaci ukázal nástroj NetEm. Tento nástroj je součástí standardního Linuxového jádra, snadno se tedy integruje do existujících systémů a umožňuje emulaci klíčových přenosových parametrů.

4.1 Technologie pro vývoj

Za účelem integrování emulátoru do prostředí Apache JMeter bylo žádoucí emulátor vyvíjet v programovacím jazyce Java, ve kterém je kompletně naprogramován i samotný JMeter. Snadné sestavení projektu umožnil nástroj Gradle. Aplikace vyvíjené v Javě jsou multiplatformní a tedy plně funkční na libovolném operačním systému. Zvolený nástroj pro emulaci `netem` je však dostupný pouze v linuxovém jádře, proto byla jako operační systém zvolena linuxová distribuce, konkrétně Ubuntu Desktop. V tabulce 4.1 jsou uvedeny použité technologie a jejich verze.

Tab. 4.1: Souhrn použitých technologií

Technologie	**Účel**	**Verze**
Apache JMeter	Software pro zátěžové testování	5.6.3
Java OpenJDK	Vývojová sada	23.0.2
Gradle	Nástroj pro sestavení	8.10.2
Ubuntu Desktop	Operační systém	24.04.2 LTS

4.2 Software Apache JMeter

Apache JMeter je nástroj pro testování výkonnosti, který podporuje různé aplikace, servery a protokoly. Umožňuje simulovat zatížení na serveru, skupině serverů nebo síti, což pomáhá analyzovat jejich výkon pod různými typy zátěže. Mezi podporované protokoly patří HTTP, HTTPS, SOAP/REST webové služby, FTP, JDBC, LDAP, SMTP, TCP a další. JMeter nabízí bohaté testovací prostředí, které umožňuje rychlou tvorbu testovacích plánů, jejich nahrávání a ladění a podporuje také běh v příkazovém řádku pro testování na různých operačních systémech. Dále JMeter nabízí rozšiřitelnost skrze skriptovatelné samplery, možnost volby různých statistických nástrojů a pluginů pro analýzu a vizualizaci dat. JMeter umožňuje snadné

shromažďování a analýzu dat z různých formátů odpovědí (HTML, JSON, XML) a poskytuje kompletní HTML reporty pro přehledné zobrazení výsledků testování.

4.2.1 Architektura systému

JMeter díky své modulární architektuře nabízí vývojářům možnost vytvářet vlastní rozšiřující moduly, jako jsou samplery, posluchače, procesory nebo vizualizační prvky prostřednictvím otevřeného API. Struktura adresářů JMeteru je organizována následovně:

- **`bin`**: Obsahuje `.bat` and `.sh` skripty, prostřednictvím kterých se JMeter spouští, soubor `ApacheJMeter.jar` a konfigurační soubory.
- **`build/docs`**: Dokumentace.
- **`extras`**: Obsahuje rozšiřující soubory.
- **`lib`**: Obsahuje požadované JAR soubory pro JMeter.
- **`lib/ext`**: Obsahuje JAR soubory pro jednotlivé protokoly.
- **`src`**: Obsahuje podadresáře pro jednotlivé protokoly a komponenty.
- **`src/*/test`**: Obsahuje soubory určené pro automatické testování.

4.2.2 Registrace komponenty

JMeter při svém spuštění shromažďuje cesty ke všem JAR souborům, které se nacházejí v adresářích `lib`, `lib/ext`, `lib/junit` a tyto cesty ukládá do proměnné `classpath`. Následně JMeter prohledává každý JAR soubor v těchto cestách a detekuje komponenty (například funkce, samplery). Tento proces prohledávání je výpočetně náročný a může zpomalit start JMeteru, zejména pokud je v systému velké množství komponent. Proto se doporučuje vlastní komponentu registrovat pomocí anotace `@AutoService`. Tato anotace při sestavování automaticky vygeneruje soubor do adresáře `META-INF/services`. Aby JMeter neprováděl prohledávání `classpath` a používal místo toho takto zaregistrovanou službu, je zapotřebí do souboru vlastností `MANIFEST.MF` přidat atribut `JMeter-Skip-Class-Scanning: true`.

4.3 Řízení síťového provozu

O řízení síťového provozu se v linuxovém jádře stará nástroj Traffic Control (klíčové slovo `tc`). Systém front (`qdisc`) je hlavním stavebním kamenem, na kterém stojí veškeré řízení provozu. Hierarchické systémy front mohou dále obsahovat třídy (`class`) a k těmto třídám lze připojit filtry (`filter`) pomocí identifikátoru třídy (`handle`).

Výstavbu příkazů pro nástroj TrafficControl zajišťuje ve vyvíjeném emulátoru třída `TrafficControl` z balíčku `tc`. Jedná se o pomocnou třídu ke třídě kontroléru `EmulatorController` (viz sekce 4.5), která zajišťuje propojení mezi modelem emulace `Emulator` (viz podsekce 4.4.1) a systémovými změnami na úrovni síťového rozhraní. Všechny pomocné metody v této třídě jsou statické a jejich popis je v tabulce 4.2. Pro účely emulátoru je klíčové jednak správné nastavení parametrů emulace a jednak aplikování filtru, který zajistí, že bude těmito parametry ovlivněn pouze požadovaný síťový provoz. Oba tyto procesy jsou popsány v následujících dvou podsekcích.

Tab. 4.2: Popis metod ve třídě `TrafficControl`

Název	**Účel**
`initRootQdisc`	Inicializuje kořenovou frontu.
`setupParameters`	Vytvoří novou třídu a nastaví jí parametry emulace.
`setupFilter`	Nastaví filtr pro zařazení síťového provozu do dané třídy.
`getAllQDiscs`	Vrací výpis všech front.
`getQDisc`	Vrací výpis fronty nastavené na daném rozhraní.
`getFilter`	Vrací výpis filtrů nastavených na daném rozhraní.
`getQDiscAndFilters`	Vrací výpis aktivních tříd a přiřazených filtrů.
`restoreDefaults`	Obnoví výchozí nastavení.

4.3.1 Emulace

Jak již bylo zmíněno, síťová emulace je realizována s využitím nástroje `netem`, který představuje jednu z implementací fronty pro systém front v Linuxovém jádře. Konfigurace fronty se provádí prostřednictvím příkazu `tc qdisc`, parametry emulace jsou zadávány jako argumenty za klíčovým slovem `netem`. V tabulce 4.3 jsou uvedeny parametry implementované v emulátoru, struktura jejich příkazu `netem` a výchozí jednotky. Parametry zapsané v hranatých závorkách jsou volitelné.

Parametrům `delay`, `reorder`, `loss`, `duplication` a `corruption` je kromě povinné hodnoty parametru možné nastavit také volitelnou korelaci. Korelace umožňuje věrohodněji emulovat reálné síťové podmínky.

V případě parametru `delay` lze kromě konstantního zpoždění zadat také kolísání zpoždění a typ rozložení. Na výběr je z rovnoměrného (klíčové slovo `uniform`), normálního (`normal`), pareto (`pareto`) a paretonormálního (`paretonormal`) rozložení. Tato rozložení jsou uložena ve formě tabulek v adresáři `/usr/lib/tc`, případně `/usr/lib64/tc` jako soubory s příponou `.dist`. Do tohoto adresáře je také možné

Tab. 4.3: Argumenty příkazu `netem` implementované v emulátoru

Parametr	Struktura příkazu	Jednotky
Zpoždění	`delay` hodnota [kolísání [korelace [rozložení]]]	[µs, µs, %, –]
Záměna pořadí	`reorder` hodnota [korelace]	[%, %]
Ztrátovost	`loss` hodnota [korelace]	[%, %]
Propustnost	`rate` hodnota [přidaná data]	[bit/s, B]
Duplikace	`duplication` hodnota [korelace]	[%, %]
Chybovost	`corruption` hodnota [korelace]	[%, %]
Omezení	`limit` hodnota	[–]

vložit vlastní rozložení a v příkazu ho následně použít pomocí jména tohoto souboru bez přípony.

U parametru `rate` lze zadat množství přidaných dat v bajtech. To umožňuje emulovat např. přidaná záhlaví spojové vrstvy. Zadáním záporné hodnoty lze naopak emulovat odebrání tohoto záhlaví, či použití komprese. Parametr `limit` určuje maximální počet paketů, které může fronta pojmout při jejich zpožďování. Pokud je hodnota `limit` přesažena, jsou další pakety zahazovány.

Jednotky uvedené v tabulce jsou výchozí, které `netem` použije, pokud jsou zadány pouze hodnoty. Přesto byly při sestavování příkazu jednotky proaktivně zadány, aby bylo předejito jakýmkoli případným chybám. U parametrů `delay` a `rate`, kde jsou výchozí jednotky µs, respektive bit/s, pak byly zvoleny více praktické jednotky, a sice ms, respektive kbit/s.

4.3.2 Filtrování

Pro zařazení paketů do některé z dílčích front jsou zapotřebí filtry. Nástroj `tc` poskytuje několik filtrovacích mechanismů. Často využívané je značkování paketů pomocí firewallu `iptables` a jejich následné rozřazování na základě těchto značek klasifikátorem `fw` (firewall mark classifier), jak je ukázáno na výpisu 4.1. Toto řešení je vhodné pro uživatele, kteří jsou dobře obeznámeni s `iptables` a nechtějí psát filtrovací pravidla jiným způsobem.

Výpis 4.1: Značkování paketů pomocí `iptables` a následná klasifikace `tc fw`

```
$iptables -t mangle -A PREROUTING -s 10.0.0.1 -j MARK --set-mark 1
$tc filter add dev eth0 parent 1:0 protocol ip handle 1 fw \
    flowid 1:10
```

Dalším mechanismem je univerzální filtr `u32` (universal 32bit filter), který umožňuje rozhodování na základě libovolných polí ze záhlaví IP paketu a některých vybraných polí ze záhlaví protokolu čtvrté a druhé vrstvy. Výhodou tohoto řešení oproti `fw` je, že filtrování i klasifikace jsou provedeny v jednom kroku, což má kladný dopad na výkon a umožňuje snazší správu filtrovacích pravidel. Filtrování lze definovat na nízké úrovni posloupností bitů, maskou a posunutím. Na výpisu 4.2 je ukázka filtrování IPv4 adresy `192.168.1.100` a prefixu `/32`.

Výpis 4.2: Použití filtru `tc u32` s bitovým porovnáním

```
$tc filter add dev ens33 protocol ip parent 1:0 prio 1 u32 \
    match u32 0xC0A80164 0xFFFFFFFF at 12 \
    flowid 1:10
```

Pro lepší uživatelskou přívětivost však existují abstrahující příkazy, kterými je možné definovat filtrovací pravidla na vyšší úrovni. Filtr z předchozího výpisu lze pak ekvivalentně zapsat jako na výpisu 4.3. Tento přístup umožňuje použít validované údaje z GUI emulátoru přímo do příkazu, aniž by bylo nutné je jakkoliv dále modifikovat. Díky své jednoduchosti a přímočarosti byl právě tento přístup implementován do vyvíjeného emulátoru.

Výpis 4.3: Použití filtru `tc u32` s vyšší úrovní zápisu

```
$tc filter add dev ens33 protocol ip parent 1:0 prio 1 u32 \
    match ip src 192.168.1.100/32 \
    flowid 1:10
```

Přidání každé další položky do filtru `u32` začíná klíčovým slovem `match` a předponou použitého protokolu IP. Protokol IPv4 používá předponu `ip`, protokol IPv6 používá předponu `ip6`. V tabulce 4.4 je přehled polí ze záhlaví IP protokolu implementovaných v emulátoru. K dispozici jsou i další pole, jako např. délka paketu nebo příznaky DF (Don't Fragment) a MF (More Fragments), tyto však nebyly posouzeny jako relevantní pro účely emulátoru a nebyly proto implementovány.

4.4 Datová reprezentace

V balíčku `model` jsou obsaženy definice datových struktur, které reprezentují jednotlivé komponenty emulátoru. Jejich popis je v následujících podsekcích.

4.4.1 Model emulátoru

Datová reprezentace emulátoru je ve třídě `Emulator`. Účelem této třídy je pouze uchovávat seznam síťových rozhraní. K tomu byla využita datová struktura `List` z

Tab. 4.4: Argumenty filtru `tc u32` implementované v emulátoru

Filtr	Struktura příkazu
Zapouzdřený protokol	`protocol` [protokol] `0xff`
Zdrojová IP adresa, prefix	`src` [adresa/prefix]
Cílová IP adresa, prefix	`dst` [adresa/prefix]
Zdrojový port	`sport` [port] `0xffff`
Cílový port	`dport` [port] `0xffff`
Typ ICMP zprávy	`icmp_type` [typ] `0xff`
Kód ICMP zprávy	`icmp_code` [kód] `0xff`
Typ služby (IPv4)	`dsfield` [typ] `0xff`
Třída provozu (IPv6)	`priority` [třída] `0xff`
Identifikace toku dat (IPv6)	`flowlabel` [tok] `0x000fffff`

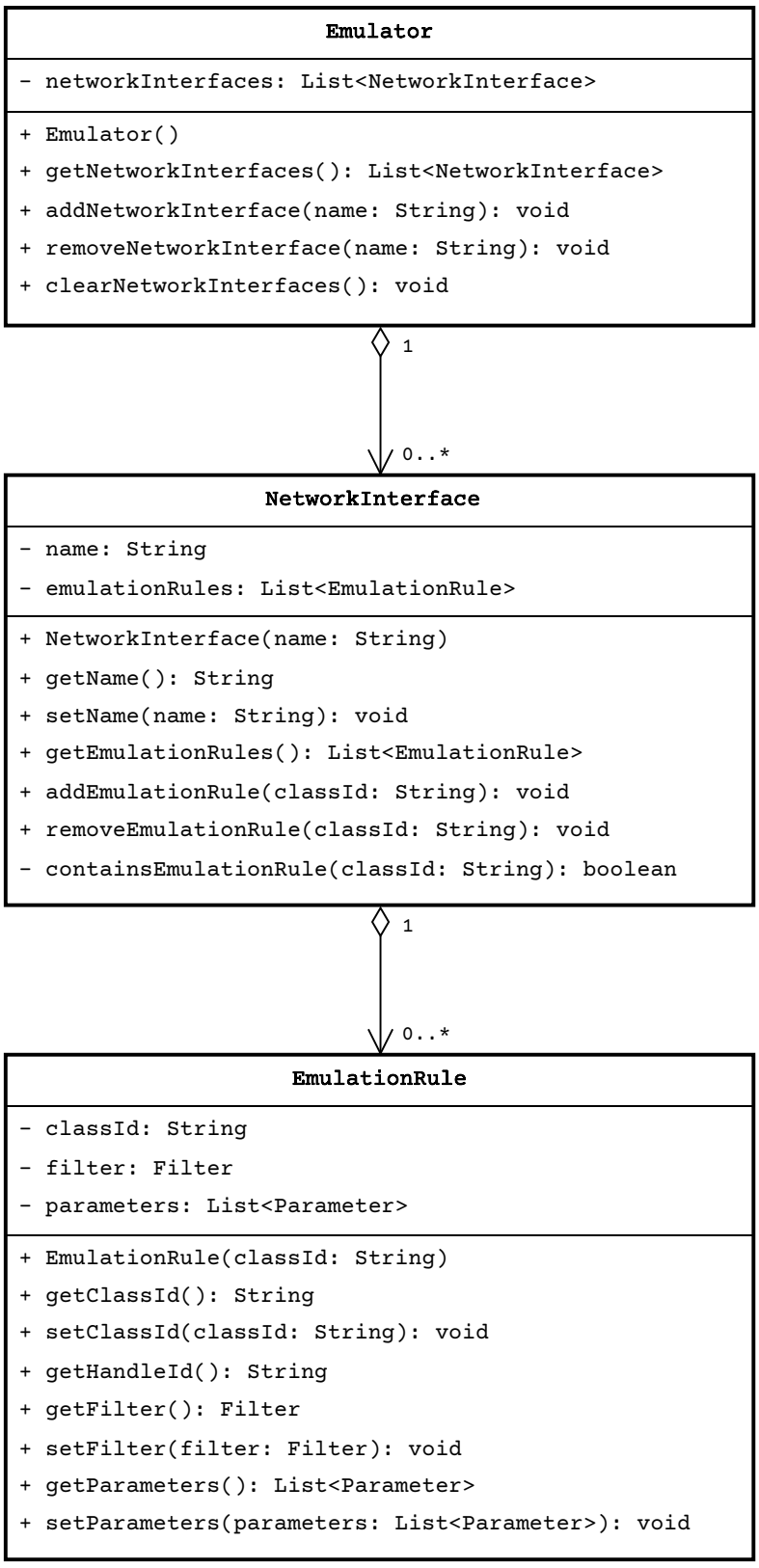
balíčku `java.util`. Součástí třídy jsou metody pro získání seznamu síťových rozhraní, vymazání seznamu, přidání síťového rozhraní do seznamu, respektive jeho odebrání. Konstruktor slouží k inicializaci seznamu síťových rozhraní.

4.4.2 Model síťového rozhraní

Model síťového rozhraní je reprezentován třídou `NetworkInterface`. Každá instance této třídy uchovává název síťového rozhraní a seznam přiřazených emulačních pravidel tomuto rozhraní. Třída poskytuje veřejné metody pro získání a nastavení jména rozhraní, získání emulačních pravidel a přidání, respektive odebrání emulačního pravidla. Soukromá metoda `containsEmulationRule` slouží ke kontrole přítomnosti daného pravidla v seznamu.

4.4.3 Model emulačního pravidla

Třída `EmulationRule` uchovává údaje související s jednotlivými emulačními pravidly. Každé pravidlo je charakterizováno identifikátorem odpovídající třídy, filtrem a seznamem parametrů emulace. Kromě konvenčních metod pro získání a nastavování jednotlivých polí třída poskytuje také metodu pro získání číselného identifikátoru `handleId`. Třídní diagram na obrázku 4.1 znázorňuje vztah mezi třídami `Emulator`, `NetworkInterface` a `EmulationRule`. Z tohoto diagramu lze vyčíst, že objekt typu `Emulator` agreguje libovolný počet instancí třídy `NetworkInterface` a každá instance `NetworkInterface` agreguje libovolný počet instancí třídy `EmulationRule`.

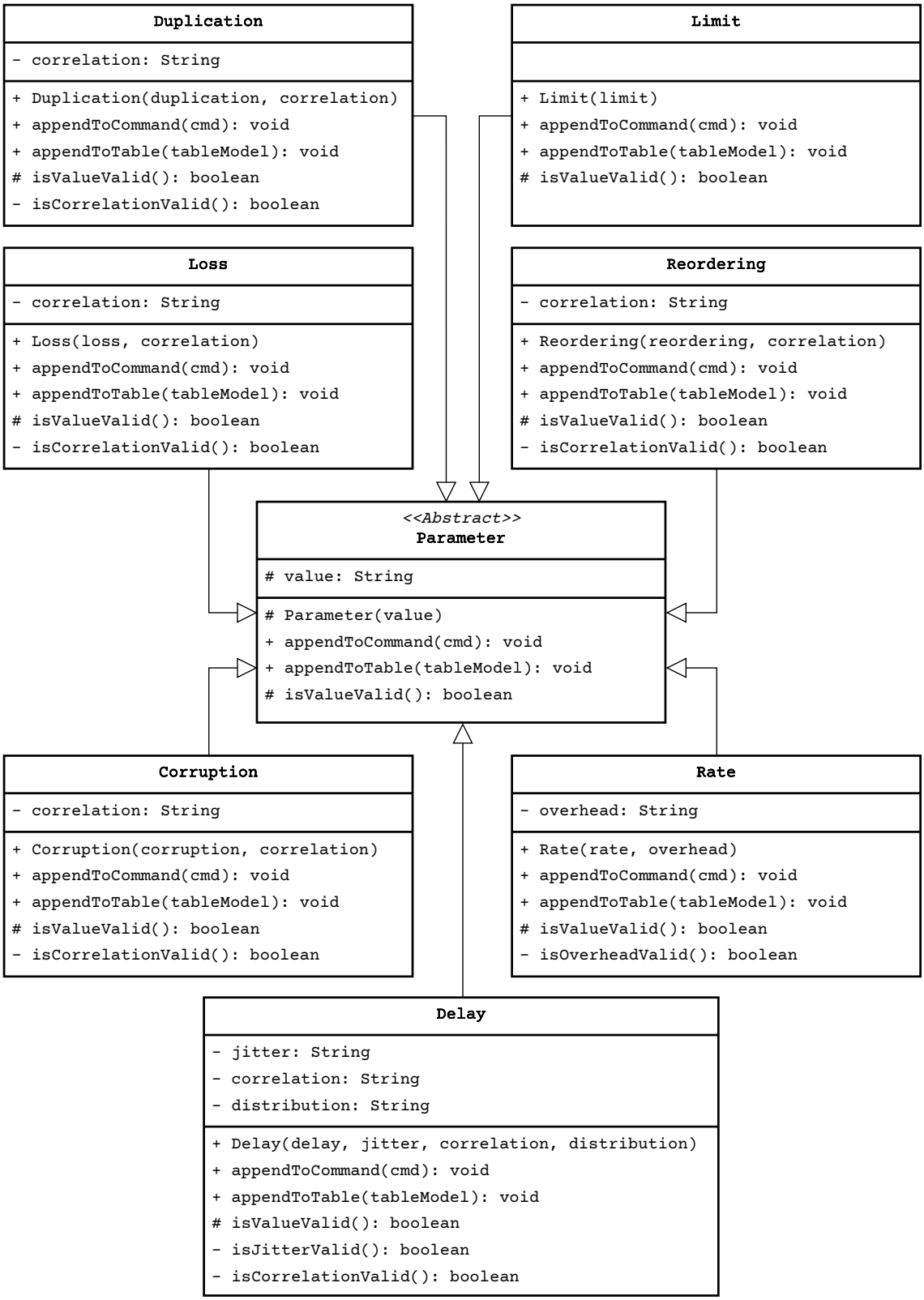


Obr. 4.1: Třídní diagramy emulátoru, síťového rozhraní a emulačního pravidla

4.4.4 Model přenosových parametrů

Každý přenosový parametr určený k emulaci je reprezentován vlastní třídou v balíčku `model/parameters`. Všechny tyto třídy dědí z abstraktní třídy `Parameter`, která definuje obecný parametr, jenž má hodnotu, validátor této hodnoty a konstruktor. Každý potomek této třídy musí navíc implementovat metody `appendToCommand`, respektive `appendToTable`. Tyto metody slouží k připojení parametru do příkazu `tc` (viz podsekce 4.3.1), respektive připojení parametru do informační tabulky zobrazované v panelu síťového rozhraní (viz podsekce 4.6.6). Výsledný diagram tříd přenosových parametrů je uveden na obrázku 4.2.

Současná verze emulátoru pracuje s předem stanovenými jednotkami dílčích parametrů. Hodnota parametru zpoždění (na obrázku `Delay`) a přidružená hodnota kolísání zpoždění jsou uváděny v milisekundách (ms). Propustnost (`Rate`) má jednotku kilobit za sekundu (kbit/s) a přidaná data jsou v bajtech (B). Omezení (`Limit`) udává povolený počet paketů ve frontě a je tedy bez jednotky (–). Parametry duplikace (`Duplication`), ztrátovosti (`Loss`), chybovosti (`Corruption`) a záměny pořadí (`Reordering`) jsou uváděny v procentech (%). Všechny korelace parametrů jsou rovněž v procentech. Nad parametry nejsou kromě validace prováděny žádné další výpočty či operace, proto mohou být v paměti uloženy jako textové řetězce (`String`). Toto řešení svědčí čistotě kódu a umožňuje snadnou výměnu dat mezi GUI prvky a datovou reprezentací.



Obr. 4.2: Třídní diagramy přenosových parametrů

4.4.5 Model filtru

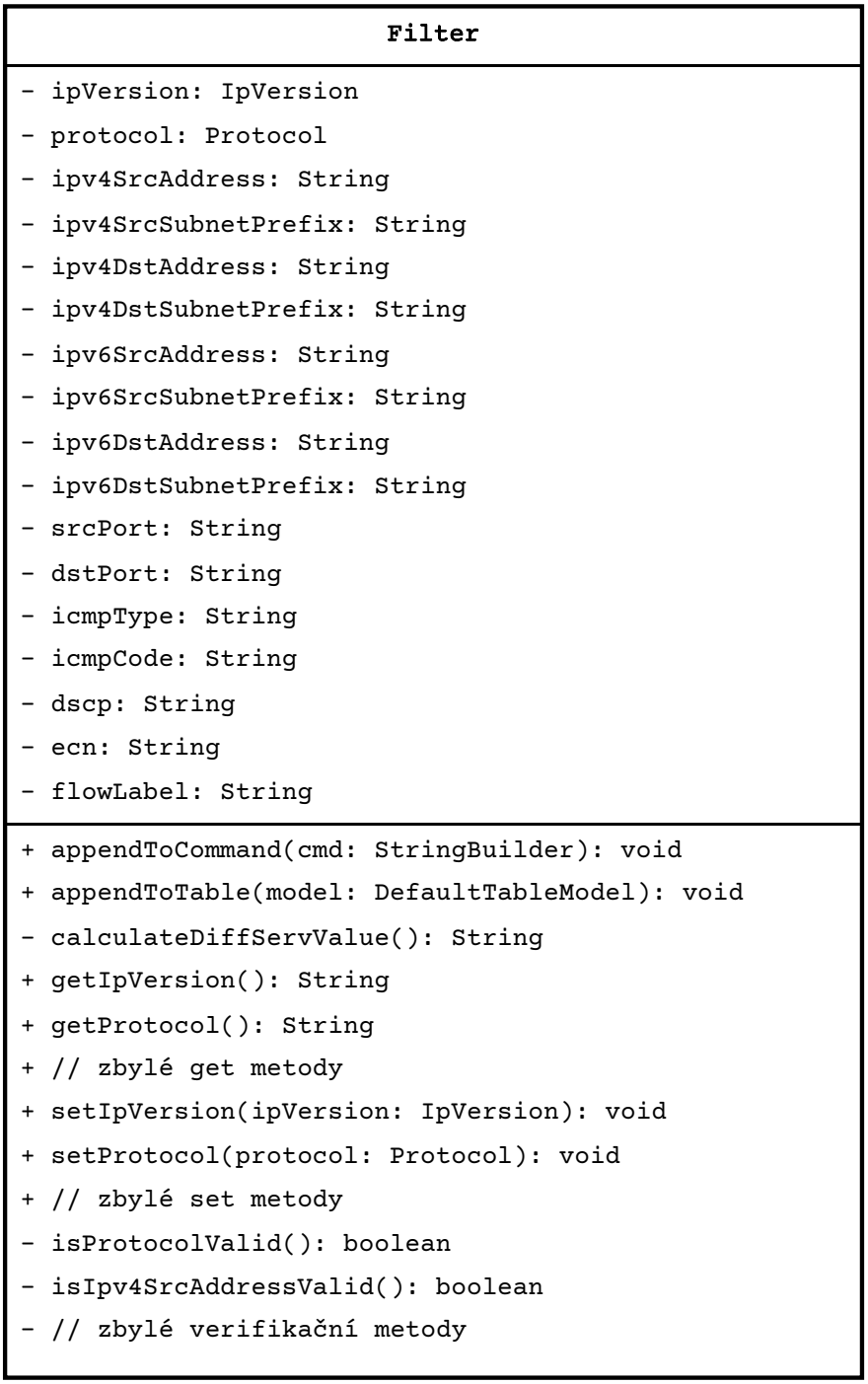
Každé emulační pravidlo obsahuje kromě parametrů také filtr. Datová reprezentace filtru je definována ve třídě `Filter` umístěné v balíčku `filter`. Diagram této třídy je na obrázku 4.3. Třída obsahuje soukromé atributy uchovávající jednotlivá filtrovací kritéria. Všechny atributy jsou datového typu `String`. Jediná metoda, ve které byly prováděny výpočty nad uloženými daty, je soukromá metoda `calculateDiffServValue` (viz výpis 4.4). Tato metoda slouží k výpočtu osmibitového pole Type of Service, respektive Traffic Class ze záhlaví IPv4, respektive IPv6 paketu. Výsledná hodnota je složena z šesti bitů pro DSCP (Differentiated Services Code Point) a dvou bitů pro ECN (Explicit Congestion Notification). Vynásobením hodnoty DSCP čtyřikrát (ekvivalentní posunu o dva bity doleva) se připraví šest nejvýznamnějších bitů. Přičtením hodnoty ECN se doplní dva nejméně významné bity. Třída dále obsahuje výše již vysvětlené metody `appendToCommand` a `appendToTable`.

Výpis 4.4: Metoda `calculateDiffServValue` ze třídy `Filter`

```
197 private String calculateDiffServValue() {
198         int dscpInt;
199         int ecnInt;
200 
201         try {
202             dscpInt = Integer.parseInt(dscp);
203         } catch (NumberFormatException e) {
204             dscpInt = 0;
205         }
206 
207         try {
208             ecnInt = Integer.parseInt(ecn);
209         } catch (NumberFormatException e) {
210             ecnInt = 0;
211         }
212 
213         return Integer.toString(dscpInt * 4 + ecnInt);
214 }
```

4.5 Kontrolér

Třída `EmulatorController` z balíčku `controller` slouží jako centrální řídicí jednotka síťového emulátoru. Tato třída zprostředkovává komunikaci mezi modelem emulace (`Emulator`) a nástrojem `tc` prostřednictvím třídy `TrafficControl`. Pro zajištění existence pouze jediné instance kontroléru v rámci celého emulátoru byl použit návrhový vzor Singleton.

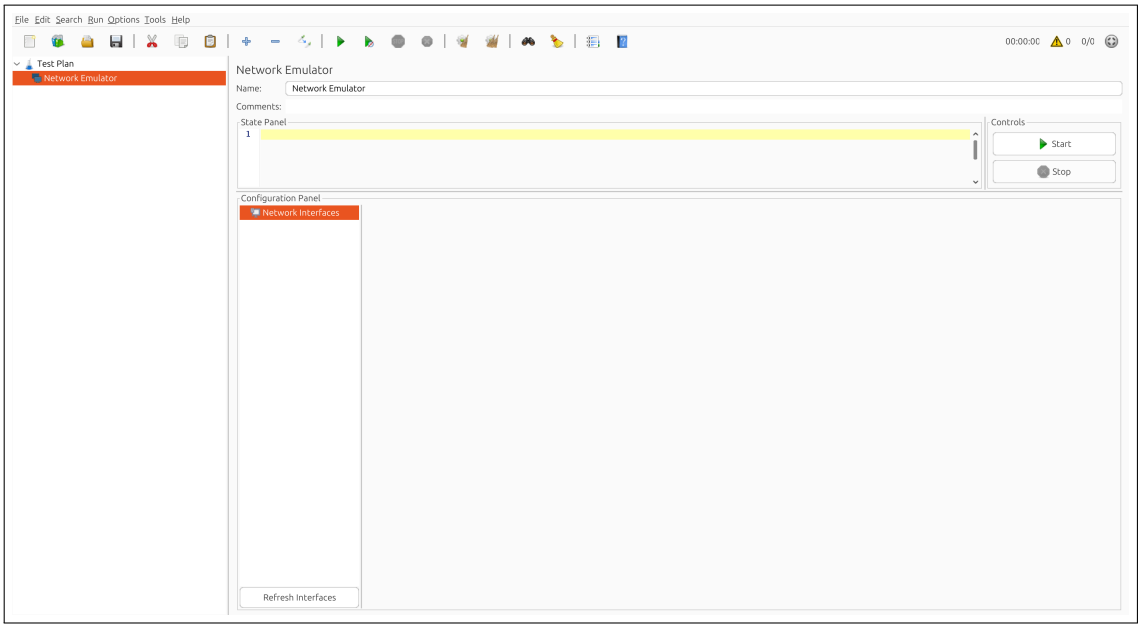


Obr. 4.3: Třídní diagram filtru

Spuštění nebo zastavení emulace uživatelem v ovládacím panelu (viz podsekce 4.6.3) vyvolá odpovídající akce, které jsou prostřednictvím kontroléru aplikovány na fyzická síťová rozhraní pomocí metod `startEmulation` a `stopEmulation`. Kontrolér zajišťuje načtení dostupných síťových rozhraní, konfiguraci emulačních pravidel, spuštění či ukončení emulace a další funkce.

4.6 Grafické uživatelské rozhraní

Kód definující grafické uživatelské rozhraní je obsažen v balíčku `gui`. Jednotlivé části rozhraní (panely) jsou rozděleny do tříd, přičemž hlavní komponentu definuje třída `EmulatorGui` (viz podsekce 4.6.1). Na obrázku 4.4 je podoba nově přidaného emulátoru do testovacího plánu JMeteru. Jednotlivé části jsou popsány v podsekcích 4.6.2 až 4.6.6. Pro tvorbu grafického rozhraní byly použity zejména uživatelské prvky z knihovny `Swing` a některé doplňující komponenty a funkcionality ze starší knihovny `AWT` (Abstract Windowing Toolkit). Hlavní komponenta implicitně dědí ze třídy `JPanel` a jednotlivé `Swing` elementy se přidávají v konstruktoru metodou `add`.



Obr. 4.4: Výchozí stav emulátoru po přidání do prostředí JMeter

Pro správné rozmístění uživatelských prvků v rámci panelu bylo zapotřebí zvolit vhodný správce rozvržení (anglicky layout manager). `Swing` poskytuje několik správců rozvržení jako `BorderLayout`, `FlowLayout` nebo `BoxLayout`. Většina z nich je však vhodná pouze pro určitý způsob rozmístění a při tvorbě složitějších uživatelských rozhraní nenabízí příliš velkou flexibilitu. Z tohoto důvodu byl ve všech dílčích kontejnerech použit externí správce rozvržení `MigLayout`, který dovoluje jednoduchým a přímočarým způsobem definovat takřka libovolné rozmístění. Dále byl v několika případech použit správce `CardLayout`, díky kterému mohou více panelů sdílet stejné místo v kontejneru a je možné mezi nimi přepínat.

JMeter pro své uživatelské rozhraní nabízí několik vzhledů (anglicky Look and Feel, LaF). Kromě systémového LaF jsou to např. CDE/Motif, Nimbus nebo Darcula. Vzhledy ovlivňují barevné schéma, tvary ovládacích prvků a celkový vizuální

dojem z aplikace. Uživatel může mezi těmito vzhledy volně přepínat. Proto bylo GUI emulátoru vyvíjeno tak, aby bylo plně kompatibilní se změnami vzhledu a zachovalo správnou funkčnost i estetiku bez ohledu na zvolený LaF.

4.6.1 Hlavní komponenta

Grafická reprezentace každé rozšiřující komponenty JMeteru musí dědit ze třídy `AbstractJMeterGuiComponent` a implementovat požadované metody. Seznam těchto metod a jejich účel je v tabulce 4.5. Podrobnější popis metod klíčových pro správnou integraci do prostředí JMeteru následuje pod tabulkou.

Tab. 4.5: Abstraktní metody třídy `AbstractJMeterGuiComponent`

Název	**Účel**
`getLabelResource`	Vrací klíč pro popisek komponenty.
`getStaticLabel`	Vrací zobrazený název komponenty.
`makeTestElement`	Vytváří testovací prvek.
`modifyTestElement`	Upravuje testovací prvek podle stavu GUI.
`configure`	Upravuje GUI podle stavu testovacího prvku.
`createPopupMenu`	Vytváří kontextové menu.
`getMenuCategories`	Určuje umístění v nabídce JMeteru.

Metoda `makeTestElement` slouží k vytvoření nové instance testovacího prvku `EmulatorTestElement`, který reprezentuje emulátor na úrovni testovacího plánu JMeteru. Tato metoda je volána při přidání emulátoru do grafického stromu JMeteru a vrací neinicializovanou instanci komponenty.

Metoda `modifyTestElement` upravuje danou instanci testovacího prvku na základě aktuálního stavu GUI komponenty. Naopak metoda `configure` slouží k nastavení GUI komponent na základě dat uložených v testovacím prvku. Tyto metody jsou volány při akcích uživatele, jako jsou přepnutí položky ve stromu, uložení testovacího plánu nebo spuštění testu.

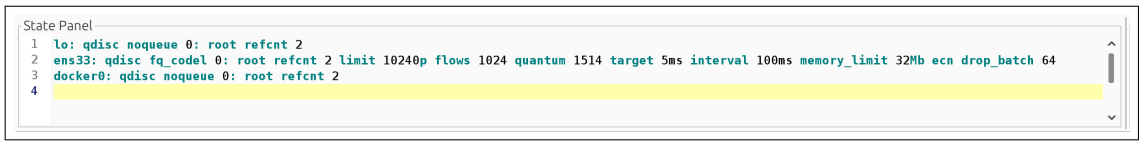
Metoda `createPopupMenu` vytváří vyskakovací nabídku, která se uživateli zobrazí při kliknutí pravým tlačítkem myši na komponentu v rámci testu JMeteru. Tato metoda vrací instanci třídy `JPopupMenu` do které lze jednotlivé položky přidávat jako instance třídy `JMenuItem`. Tímto způsobem bylo sestaveno kontextové menu pro vytvářený emulátor.

Metoda `getMenuCategories` určuje umístění vytvářené komponenty v kontextovém menu testovacího plánu JMeteru. Tato metoda je klíčová, protože vrací seznam kategorií, ze kterých je možné komponentu přidat do uživatelského rozhraní JMeter.

Komponenta emulátoru byla zařazena do kategorie „Non-Test Elements“ s využitím konstanty `MenuFactory.NON_TEST_ELEMENTS`.

4.6.2 Stavový panel

Vzhled stavového panelu je na obrázku 4.5. Účelem tohoto panelu je zobrazit uživateli přehled o aktuálním nastavení front a filtrů na jednotlivých síťových rozhraních. Jako textové pole byla využita komponenta JMeteru `JSyntaxTextArea`, která dědí z `RSyntaxTextArea`. Jedná se o uživatelský prvek určený k přehlednému zobrazování kódu nebo formátovaného textu. Dle zvolené konfigurace je možné zvýraznit klíčová slova, zobrazit čísla řádků, zalamovat řádky apod.



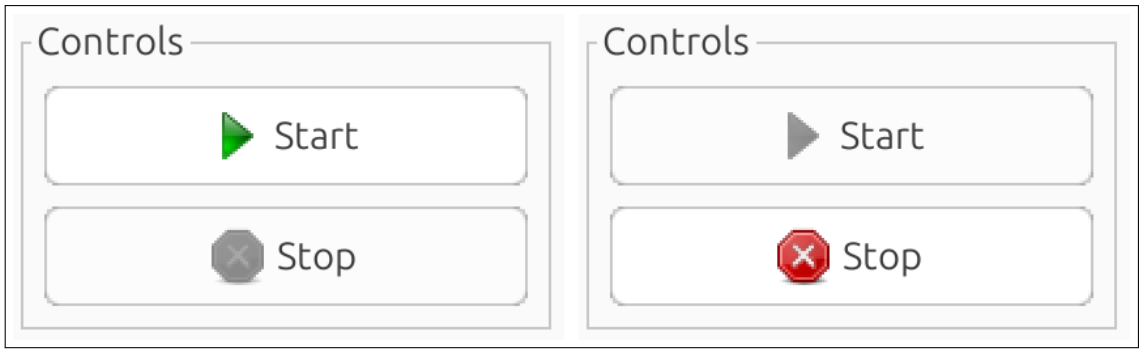
Obr. 4.5: Vzhled stavového panelu

Obsah tohoto textového pole je aktualizován vždy při změně stavu emulátoru, v metodách `onEmulationStarted` a `onEmulationStopped`. Pro případ, že by text byl příliš dlouhý, je textové pole vloženo do posuvného panelu `JScrollPane`. Dále je možnost celý horní panel vertikálně roztáhnout na úkor konfiguračního panelu díky použité `JSplitPane`.

4.6.3 Ovládací panel

Ovládací panel obsahuje tlačítko pro spuštění emulace a tlačítko pro ukončení emulace. Aby bylo na první pohled zřejmé, zda je emulace právě zapnuta či vypnuta, je jedno z tlačítek zneaktivněno, jak je vidět na následujícím obrázku 4.6.

Jako posluchač událostí byl využit posluchač `ActionRouter` definovaný JMeterem. Tlačítku „Start“ byl nastaven akční příkaz `start_emulation` a tlačítku „Stop“ akční příkaz `stop_emulation`. Tyto příkazy volají obsluhy událostí definované ve třídách `StartEmulation`, respektive `StopEmulation` umístěných v balíčku `action`.



Obr. 4.6: Ovládací panel – emulace je vypnuta (vlevo), emulace běží (vpravo)

4.6.4 Konfigurační panel

Kód konfiguračního panelu je obsažen ve třídě `ConfigurationPanel`. Konfigurační panel je s využitím `JSplitPane` horizontálně rozdělen na levý panel a pravý panel, viz obrázek 4.7.

V levém panelu je ve stromové struktuře přehled všech aktuálně nalezených síťových rozhraní a jim nastavených emulačních pravidel. Ve spodní části levého panelu jsou tlačítka pro aktualizaci síťových rozhraní, přidání nového emulačního pravidla na dané síťové rozhraní a odebrání aktuálně označeného emulačního pravidla. Tato tlačítka jsou přidána do `CardLayout` kontejneru a zobrazují se dle aktuálně vybrané položky ve stromu. Pokud je emulace zapnuta, tak jsou tato tlačítka zneaktivněna.

Pravý panel slouží jako kontejner pro panel síťového rozhraní (popsaného v podsekci 4.6.5) a jednotlivé instance panelu emulačního pravidla (podsekce 4.6.6). Oba tyto typy panelů jsou do pravého panelu vloženy jako `JScrollPane`, která zajišťuje možnost posouvání obsahu v případě, že jejich velikost překročí dostupný prostor panelu. To zajišťuje kromě rozšiřitelnosti emulátoru také jeho kompatibilitu napříč různými LaF. Mezi panely se lze přepínat výběrem příslušné položky ve stromové struktuře.

Pro přizpůsobení chování stromu potřebám vytvářeného emulátoru bylo nutné vytvořit vlastní implementaci jednotlivých komponent definujících strom v knihovně `Swing`. Za tímto účelem byly vytvořeny třídy `EmulatorTree`, `EmulatorTreeNode` a `EmulatorTreeNodeRenderer` umístěné v balíčku `gui/tree`. Třída `EmulatorTree` nad rámec své rodičovské třídy `JTree` přidává funkcionality pro programové označení položky ve stromu a rozbalení zvolených položek. Také umožňuje ostatním třídám emulátoru získat právě označenou položku, případně seznam právě rozbalených položek. Třída `EmulatorTreeNode` dědí ze třídy `DefaultMutableTreeNode` a kromě nastavení uživatelského objektu dané položce umožňuje položku pojmenovat. Právě jméno je důležité pro rozlišení dílčích emulačních pravidel. Uživatelský objekt zde

Obr. 4.7: Vzhled konfiguračního panelu

představuje panel, který je při výběru položky zobrazen v pravé části konfiguračního panelu. Třída `EmulatorTreeNodeRenderer` je potomek `DefaultTreeCellRenderer` a jednotlivým položkám ve stromu zajišťuje správné zobrazení jejich ikon.

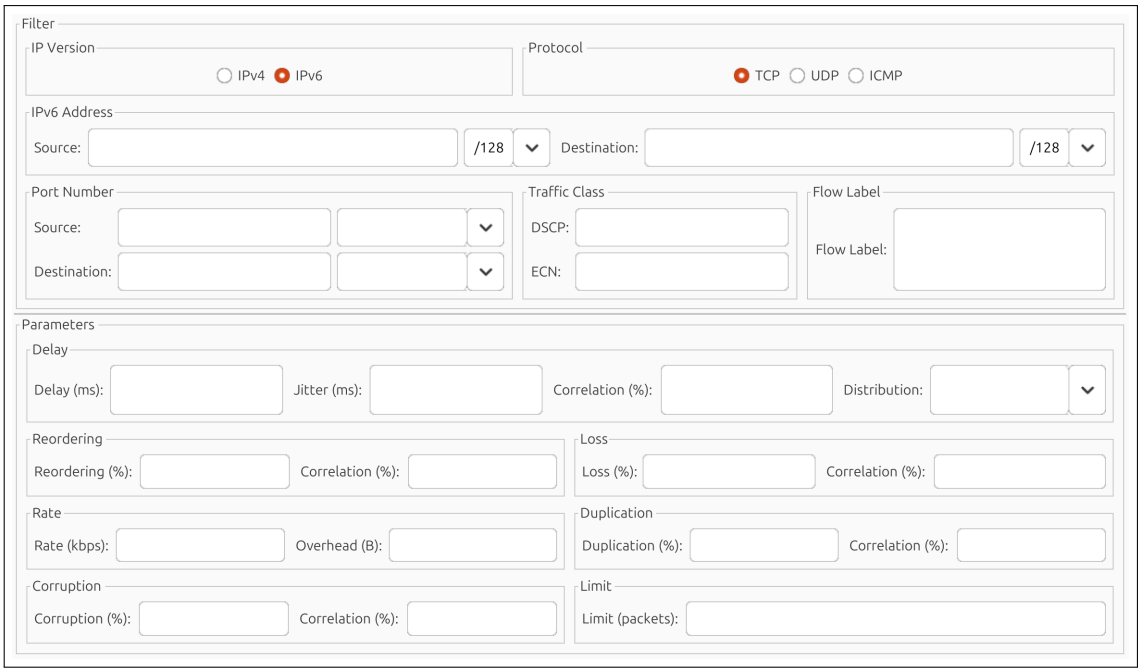
4.6.5 Panel emulačního pravidla

Panel emulačního pravidla je definován ve třídě `EmulationRulePanel`. Tento panel je klíčovou součástí emulátoru, jelikož dává uživateli možnost nastavení přenosových parametrů určených k emulaci a definování filtrovacích kritérií. Vzhled panelu emulačního pravidla je na obrázku 4.8. V horní části je umístěn panel filtru a ve spodní části je umístěn panel parametrů. Oba panely je možné dle potřeby vertikálně zvětšit či zmenšit, všechny vnitřní komponenty se této úpravě dynamicky přizpůsobí. Při běhu emulace jsou veškeré komponenty emulačního pravidla zneaktivněny, měnit nastavení emulačního pravidla je možné pouze pokud je emulace zastavena.

Panel filtru

Panel reprezentující filtr je realizován ve třídě `FilterPanel` v balíčku `gui/filter`. V prvním řádku panelu se nachází volba verze protokolu IP (verze 4 a verze 6, výchozí je verze 4) a výběr zapouzdřeného protokolu v IP paketu (implementovány jsou protokoly TCP, UDP a ICMP, výchozí je protokol TCP). Volby jsou vytvořeny jako přepínače `JRadioButton`.

Ve druhém řádku panelu je možné nastavit zdrojovou a/nebo cílovou IP adresu a příslušné délky prefixu. Jako vstupní textové pole byl použit standardní `JTextField`



Obr. 4.8: Vzhled panelu emulačního pravidla

a pro výběr délky prefixu rozbalovací seznam `JComboBox` s předvolenými délkami prefixu, které se mění v závislosti na zvolené verzi protokolu IP. Uživatel má možnost zadat konkrétní adresu (s délkami prefixu 32, respektive 128), případně celou síť, na kterou má být emulace aplikována.

Podoba třetího řádku je proměnná jak podle verze IP, tak podle protokolu. V případě zvoleného protokolu TCP nebo UDP je v prvním sloupci možnost specifikovat zdrojový a/nebo cílový port. Port je také možné zvolit z rozbalovacích seznamů vedle textových polí. Na výběr jsou často používané aplikační protokoly, jako HTTPS, FTP, DNS a další. Při výběru protokolu se automaticky do příslušného textového pole doplní odpovídající číslo portu. Pokud je zvolen protokol ICMP, volbu portů nahradí volba typu a kódu ICMP zprávy. Ve druhém sloupci je možnost nastavení polí DSCP a ECN. Pokud je zvolen IP protokol verze 6, tak se navíc zobrazí textové pole pro filtrování na základě identifikátoru toku dat (anglicky flow label).

Panel parametrů

Grafická reprezentace každého parametru je za účelem přehlednosti kódu a případné rozšiřitelnosti emulátoru realizována jako samostatná třída, přičemž všechny tyto třídy jsou soustředěny v balíčku `gui/parameters`. Panel emulačního pravidla tyto jednotlivé komponenty spojuje do jednoho celku. Pro zadání hodnot parametrů slouží opět `JTextField`. Výběr pravděpodobnostního rozložení v panelu zpoždění je

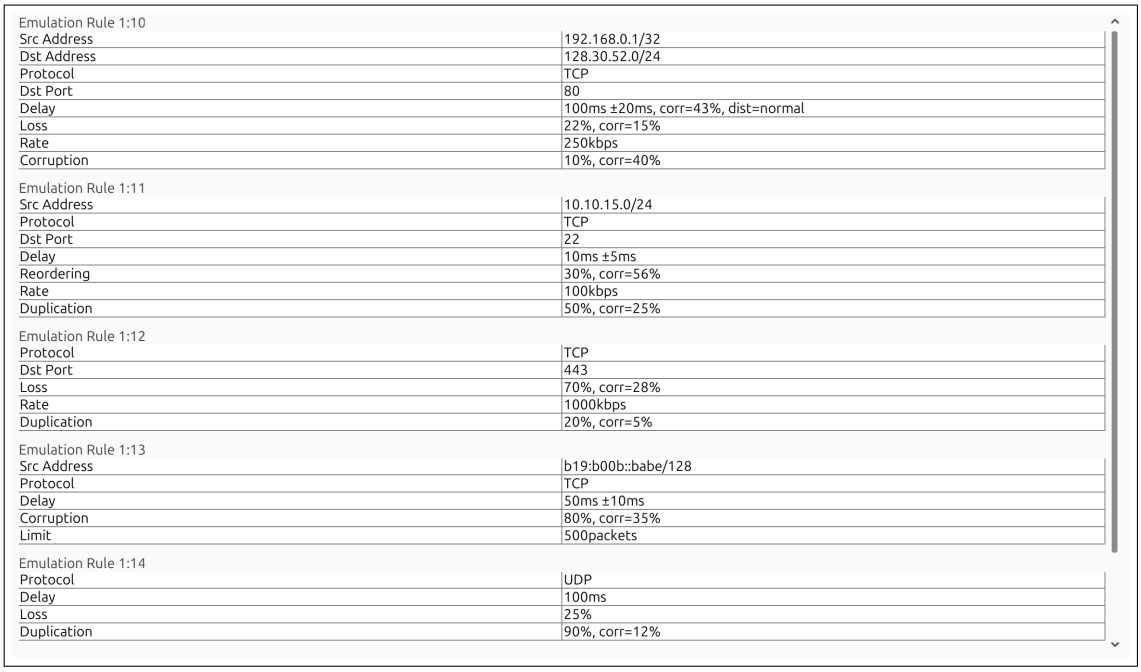
řešen rozbalovacím seznamem `JComboBox`.

Verifikace

Aby bylo zajištěno, že uživatel zadá do textových polí pouze platné hodnoty, je textovým polím nastaven odpovídající verifikátor s využitím metody `setInputVerifier`. Pro ověření IP adresy slouží verifikátor `IpAddressVerifier`, k ověření číselných vstupů v zadaném rozsahu je určen verifikátor `RangeVerifier`. Oba tyto verifikátory dědí ze třídy `InputVerifier` a nachází se v balíčku `verification`. V případě zadání neplatné hodnoty se zobrazí informační okno a uživateli není umožněno pokračovat, dokud chybu neopraví. Pokud není v textovém poli uvedena platná hodnota, případně je pole prázdné, filtrování/emulace na základě tohoto pole se neuplatní.

4.6.6 Panel síťového rozhraní

Panel síťového rozhraní (viz obrázek 4.9) poskytuje uživateli souhrn všech emulačních pravidel nastavených na vybraném síťovém rozhraní. Ve formě tabulky jsou pro každé emulační pravidlo vypsány jak údaje z panelu filtru, tak i údaje z panelu parametrů. Každá tabulka je opatřena nadpisem s identifikátorem `classId`. Za účelem získání hodnot z příslušných datových struktur je využita metoda `appendToTable`.



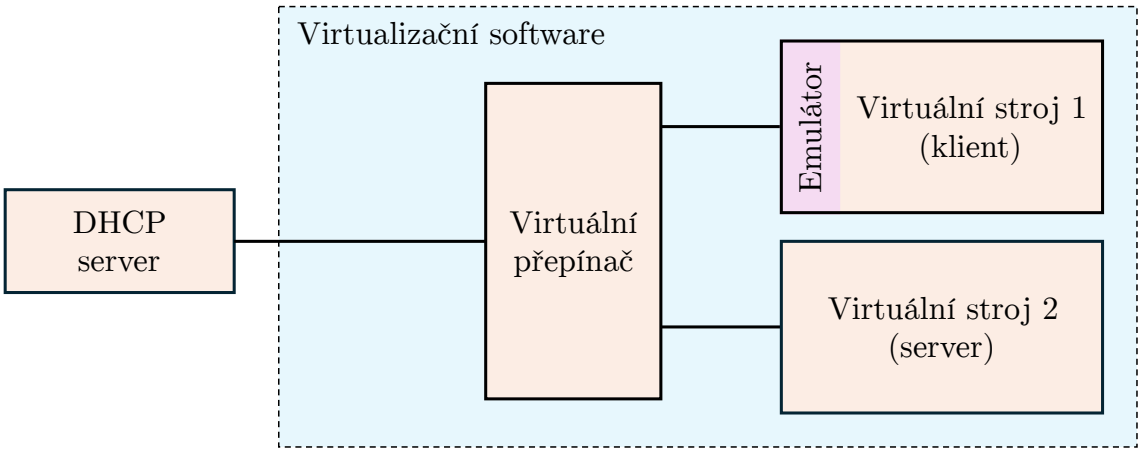
Obr. 4.9: Vzhled panelu síťového rozhraní

5 Měření a výsledky

V předchozí kapitole byl popsán vývoj síťového emulátoru. Postup jeho sestavení a zakomponování do prostředí Apache JMeter je popsán v příloze A. Cílem této části práce je otestovat funkčnost emulátoru na několika vybraných parametrech a testovacích scénářích. V sekci 5.1 je nejprve popsána měřicí soustava, vlastnímu měření je pak věnována sekce 5.2. V poslední sekci (5.3) jsou vyhodnoceny výsledky.

5.1 Popis měření

Měření přenosových parametrů bylo provedeno mezi dvěma virtuálními stroji. Schéma měřicí soustavy je na obrázku 5.1. Virtuální stroj 1 (klient) byl použit pro generování síťového provozu. Na tomto stroji byl také spuštěn JMeter s vytvořeným emulátorem. Na virtuálním stroji 2 (server) byl přijímán síťový provoz a měřeny jednotlivé parametry. Virtuální síťový adaptér obou virtuálních strojů byl nastaven do režimu síťového mostu. K tomu ve virtualizačním softwaru slouží tzv. virtuální přepínač (anglicky virtual switch), který propojuje virtuální stroje mezi sebou a zároveň umožňuje jejich komunikaci s vnější sítí. Stroje obdržely své IP adresy od DHCP serveru. Aby byly přidělené adresy v průběhu všech měření vždy stejné, byl na DHCP serveru nastaven daným virtuálním rozhraním statický DHCP binding.



Obr. 5.1: Schéma měřicí soustavy

Pro virtualizaci byl použit software VMware Workstation 17 Player ve verzi 17.6.3. Na virtuálním stroji 1 byl nainstalován operační systém **Ubuntu Desktop**, na virtuálním stroji 2 byl nainstalován **Ubuntu Server**, oba ve verzi 24.04.2 LTS.

5.2 Měření přenosových parametrů

K měření byly použity známé nástroje pro testování a vyhodnocování síťového provozu, jmenovitě `iperf3`, `ping`, `tcpdump` a `tshark`. Pro zautomatizování měření některých parametrů byly vytvořeny skripty v programovacích jazycích Python a Bash. Instalace všech potřebných nástrojů lze docílit spuštěním skriptu na výpisu B.1.

V následujících podsekcích je pro každý měřený parametr nejprve popsána metodika měření, poté jsou prezentovány naměřené hodnoty s vypočtenými odchylkami ve formě tabulek, které jsou doplněné grafickými výstupy. Veškeré hodnoty uvedené v tabulkách jsou zaokrouhleny na pět platných číslic.

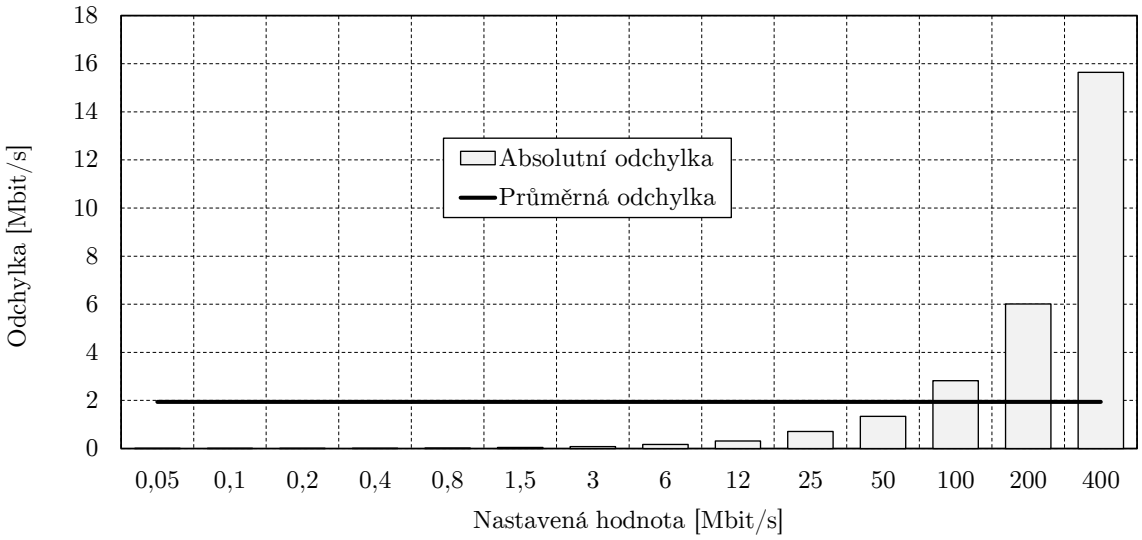
5.2.1 Propustnost

Emulace propustnosti byla otestována nástrojem `iperf3` na čtrnácti hodnotách od 50 kbit/s do 400 Mbit/s. Příkazy spuštěné na straně klienta a serveru jsou na výpisech B.2, respektive B.3. Pro každou nastavenou hodnotu propustnosti bylo provedeno celkem pět set měření. Naměřená hodnota je aritmetickým průměrem ze všech měření. Výsledky měření jsou shrnuty v tabulce 5.1. V prvním sloupci tabulky jsou hodnoty nastavené v rozhraní emulátoru, ve druhém sloupci jsou naměřené hodnoty, ve třetím a čtvrtém sloupci jsou absolutní, respektive relativní odchylky.

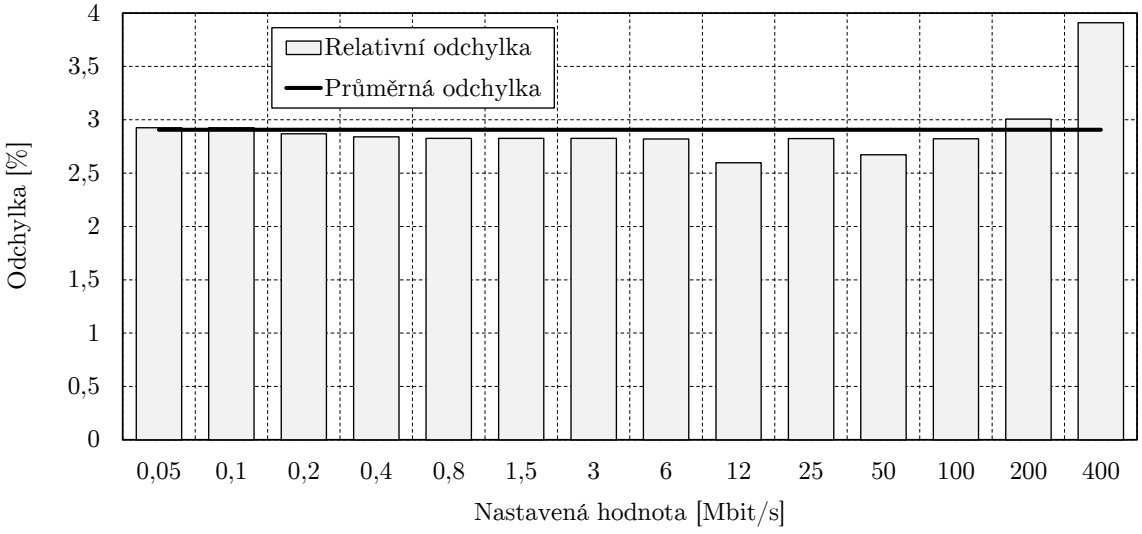
Tab. 5.1: Výsledky měření propustnosti

Nastavená hodnota [Mbit/s]	**Naměřená hodnota [Mbit/s]**	**Absolutní odchylka [kbit/s]**	**Relativní odchylka [%]**
0,05	0,048537	1,4630	2,9256
0,1	0,097074	2,9260	2,9258
0,2	0,19426	5,7380	2,8689
0,4	0,38864	11,357	2,8392
0,8	0,7774	22,598	2,8248
1,5	1,4576	42,387	2,8258
3	2,9152	84,776	2,8259
6	5,8308	169,23	2,8204
12	11,688	311,73	2,5977
25	24,294	705,72	2,8229
50	48,664	1336,1	2,6722
100	97,179	2821,2	2,8212
200	193,99	6012,7	3,0064
400	384,36	15637	3,9093

Vypočtené hodnoty relativní a absolutní odchylky byly vyneseny do sloupcových grafů na obrázcích 5.2, respektive 5.3. Z těchto grafů je patrné, že s vyššími nastavenými hodnotami propustnosti absolutní odchylka stoupá, nicméně relativní odchylka zůstává přibližně konstantní. Do obou grafických závislostí byla dále vynesena průměrná odchylka. Ta v absolutní míře nabývá hodnoty přibližně 1,94 Mbit/s, což odpovídá relativní průměrné odchylce přibližně 2,9 %.



Obr. 5.2: Absolutní odchylky propustnosti



Obr. 5.3: Relativní odchylky propustnosti

5.2.2 Zpoždění a kolísání zpoždění

Zpoždění bylo měřeno podle metodiky uvedené v sekci 1.4, naměřené hodnoty zpoždění jsou tedy koncové. Za účelem měření emulace zpoždění byl na straně klienta spuštěn skript B.4 a na straně serveru skript B.5. Při každém dílčím měření bylo odesláno celkem deset tisíc UDP paketů. Klient před odesláním každého paketu zaznamenal aktuální čas a server ve své odpovědi klientovi vrátil čas, kdy tento paket přijal. Klient po obdržení odpovědi vypočítal koncové zpoždění jako rozdíl času odeslání a času přijetí. Tento přístup měření zpoždění vyžaduje, aby byly hodiny obou virtuálních strojů synchronizované. Toho bylo docíleno s využitím nástroje `chrony`, který implementuje protokol NTP zmíněný v podsekci 1.4.4.

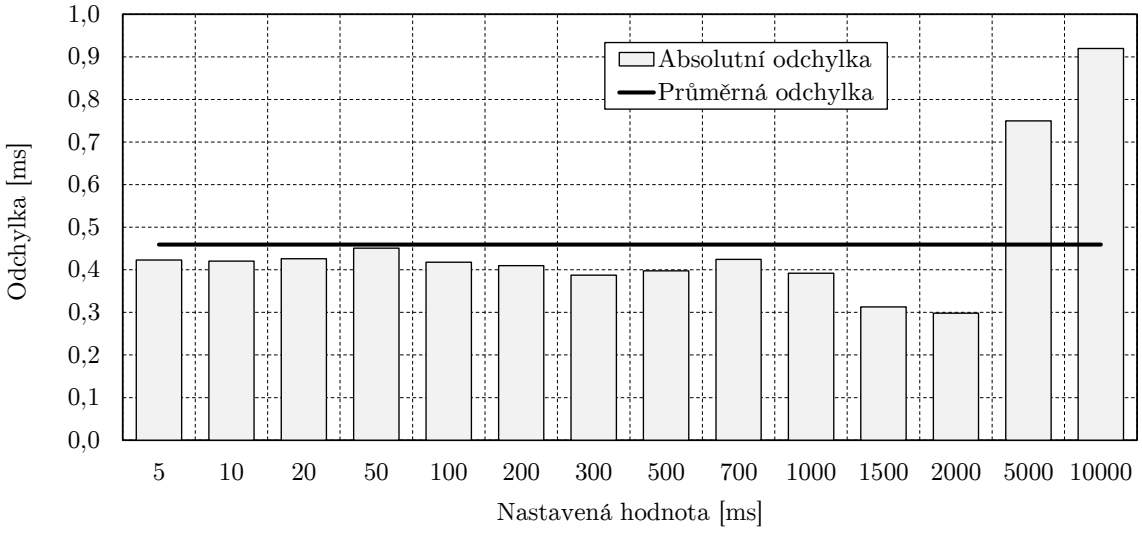
V první části měření zpoždění byla ověřena schopnost emulátoru zavádět konstantní zpoždění. Nastaveno bylo postupně čtrnáct hodnot zpoždění od pěti milisekund do deseti sekund. Nastavené a naměřené hodnoty jsou spolu s příslušnými odchylkami v tabulce 5.2.

Tab. 5.2: Výsledky měření konstantního zpoždění

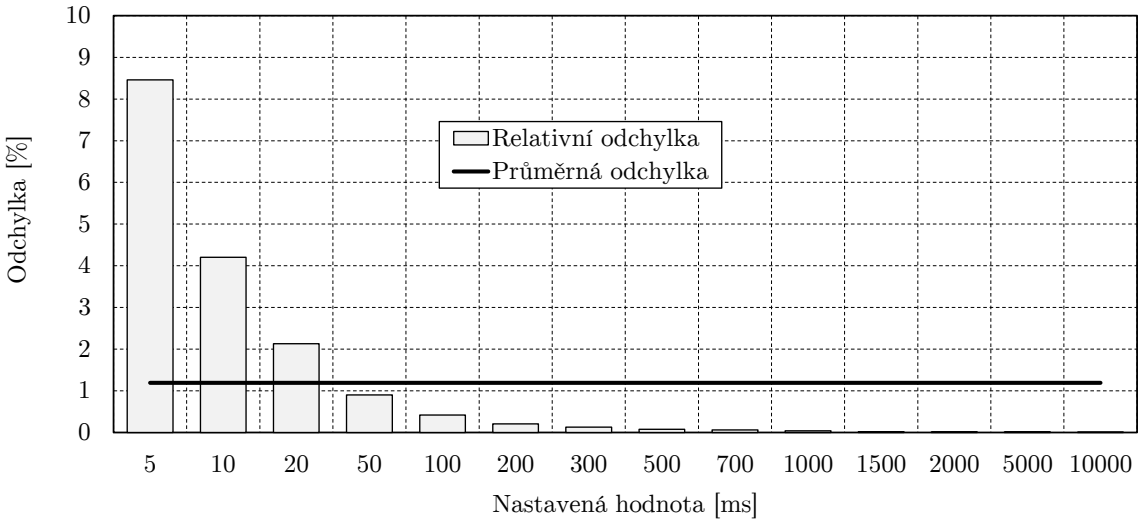
Nastavená hodnota [ms]	**Naměřená hodnota [ms]**	**Absolutní odchylka [ms]**	**Relativní odchylka [%]**
5	5,4229	0,42288	8,4576
10	10,42	0,4202	4,2020
20	20,426	0,42602	2,1301
50	50,451	0,45092	0,9018
100	100,42	0,41767	0,4177
200	200,41	0,40982	0,2049
300	300,39	0,3875	0,1292
500	500,4	0,3973	0,0795
700	700,42	0,4242	0,0606
1000	1000,4	0,39172	0,0392
1500	1500,3	0,31276	0,0209
2000	2000,3	0,29832	0,0149
5000	5000,7	0,74968	0,0150
10000	10000,9	0,91959	0,0092

Na obrázcích 5.4 a 5.5 jsou graficky vyneseny absolutní, respektive relativní odchylky konstantního zpoždění. Absolutní odchylka nepřekročila hodnotu jedné milisekundy. Pro nastavené hodnoty zpoždění pěti sekund a deseti sekund absolutní odchylka poněkud vzrostla a lze očekávat, že pro větší nastavené hodnoty zpoždění by tento trend pokračoval. V relativní míře jsou však tyto odchylky stále více

zanedbatelné, jak je zřejmé z grafu relativní odchylky.



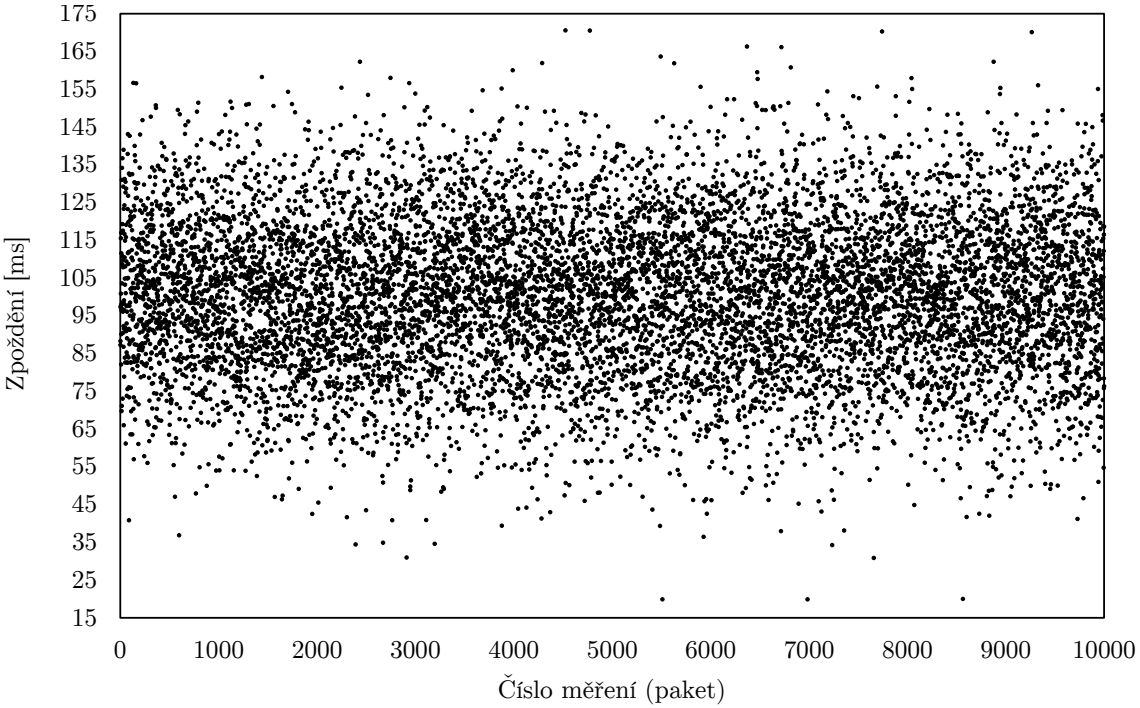
Obr. 5.4: Absolutní odchylky konstantního zpoždění



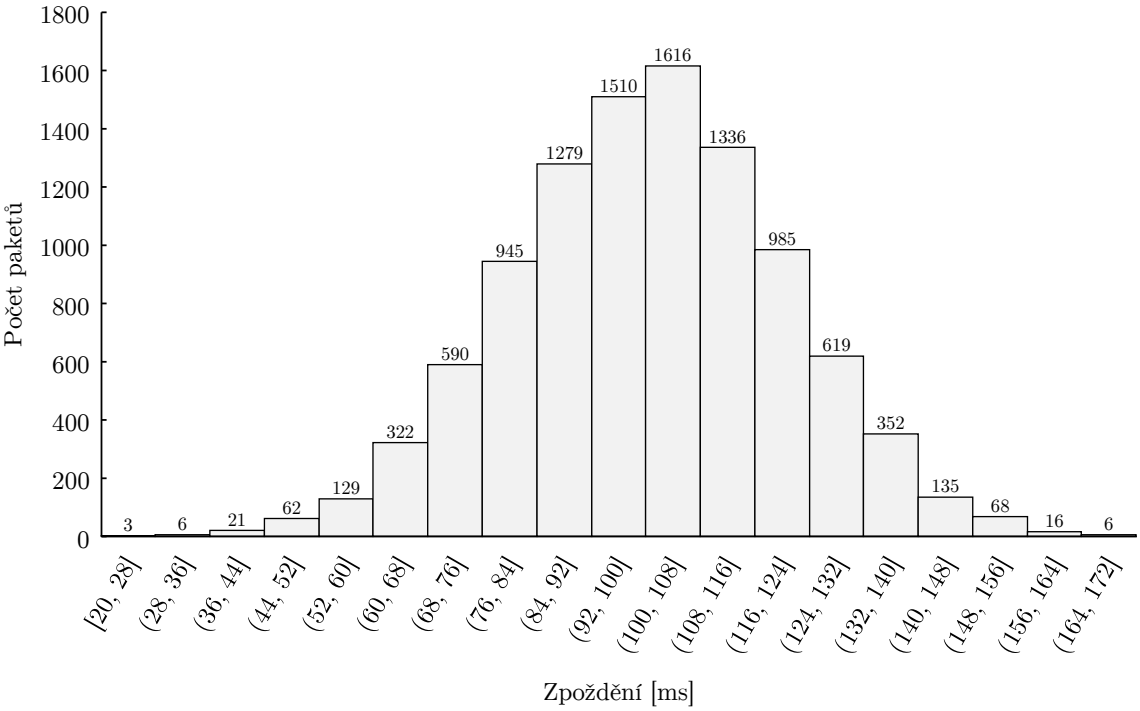
Obr. 5.5: Relativní odchylky konstantního zpoždění

Ve druhé části měření zpoždění byla ověřena schopnost emulátoru zavádět proměnné zpoždění. Zpoždění bylo nastaveno na sto milisekund s kolísáním zpoždění dvacet milisekund. Na obrázku 5.6 byla do bodového grafu pro každý odeslaný paket vynesena naměřená hodnota zpoždění při zvoleném normálním rozložení. Jiný pohled na stejné výsledky měření nabízí histogram na obrázku 5.7, ze kterého je zřejmé, že kolísání zpoždění se skutečně řídí tvarem křivky normálního rozložení. Stejné grafické výstupy jsou na obrázcích 5.8, 5.9 pro pareto rozložení, na obrázcích

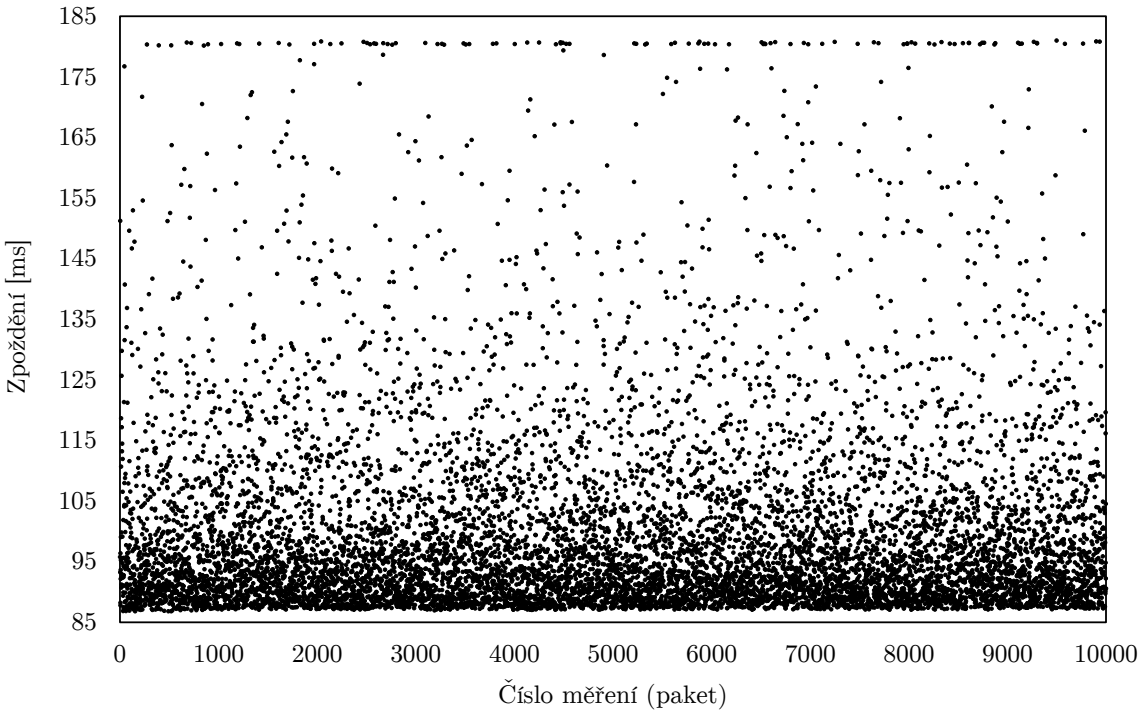
5.10, 5.11 pro pareto-normální rozložení a na obrázcích 5.12, 5.13 pro rovnoměrné rozložení.



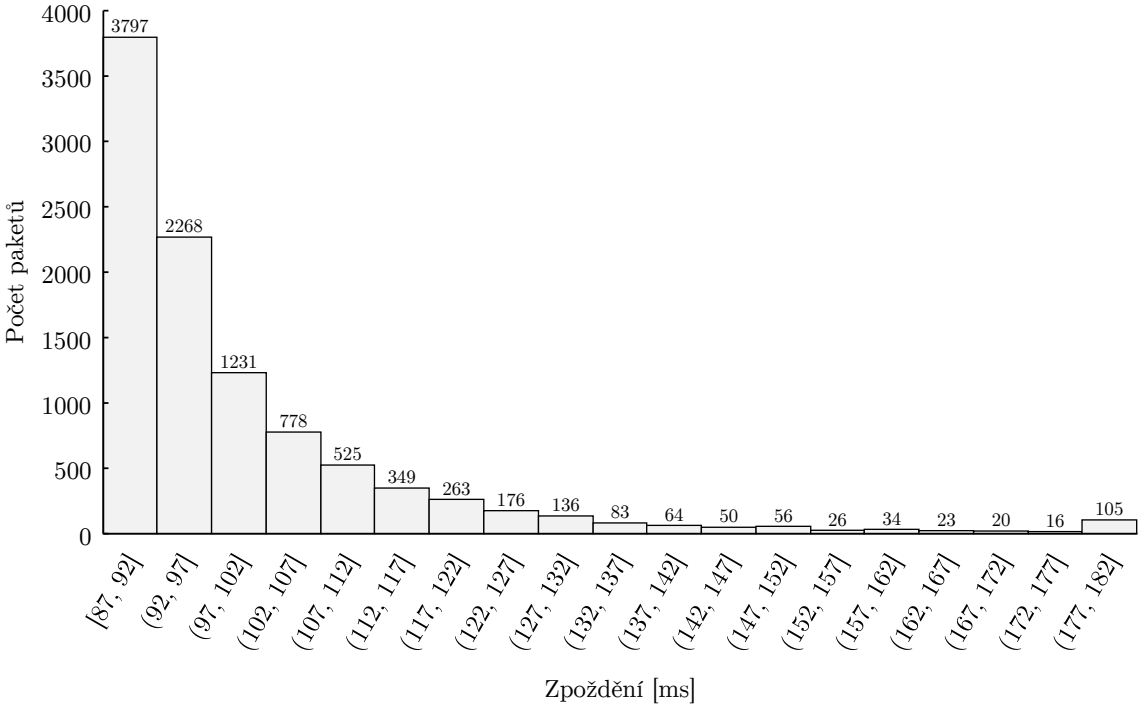
Obr. 5.6: Naměřené hodnoty zpoždění pro normální rozložení (100 ± 20 ms)



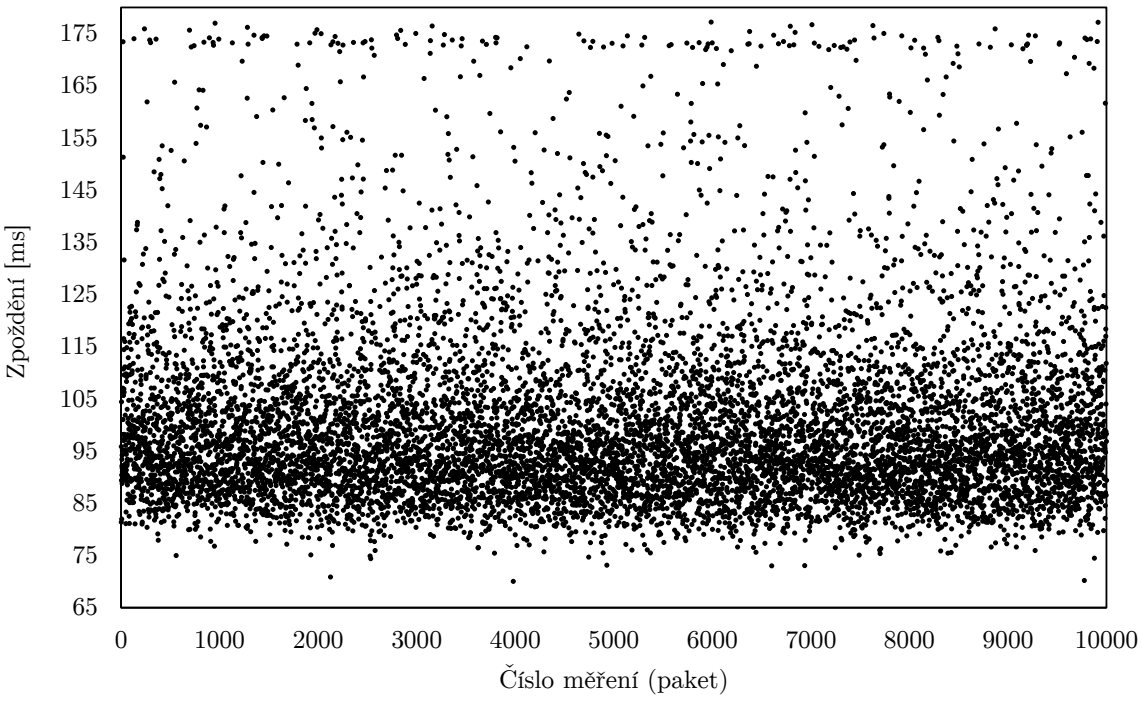
Obr. 5.7: Histogram zpoždění pro normální rozložení (100 ± 20 ms)



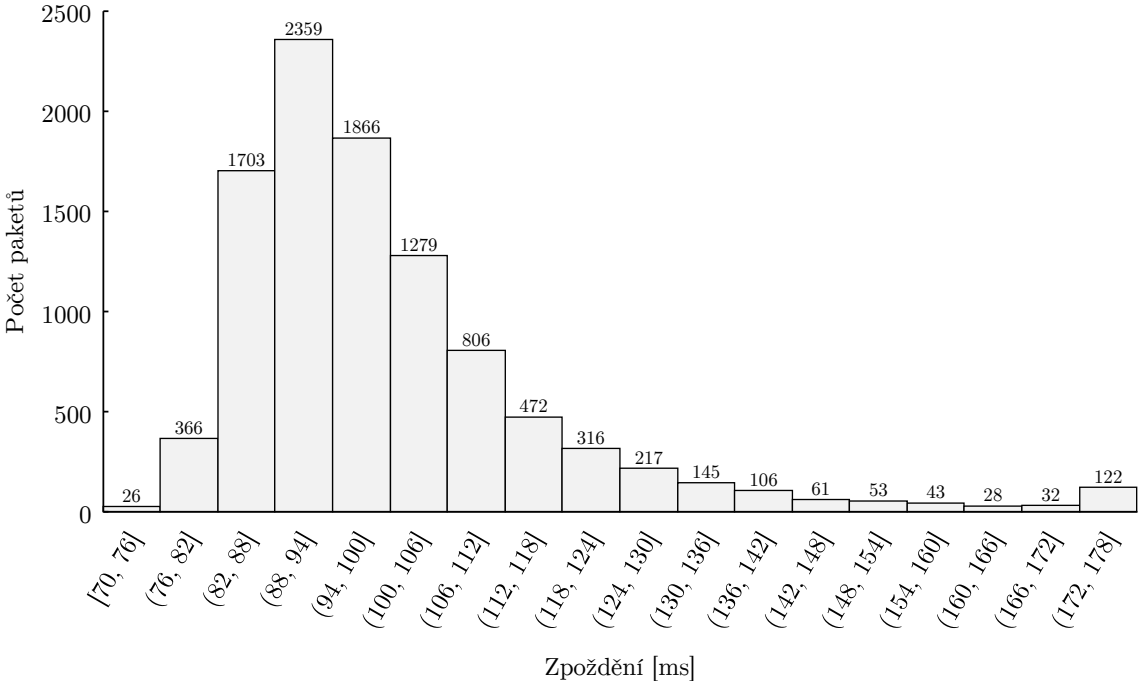
Obr. 5.8: Naměřené hodnoty zpoždění pro pareto rozložení (100 ± 20 ms)



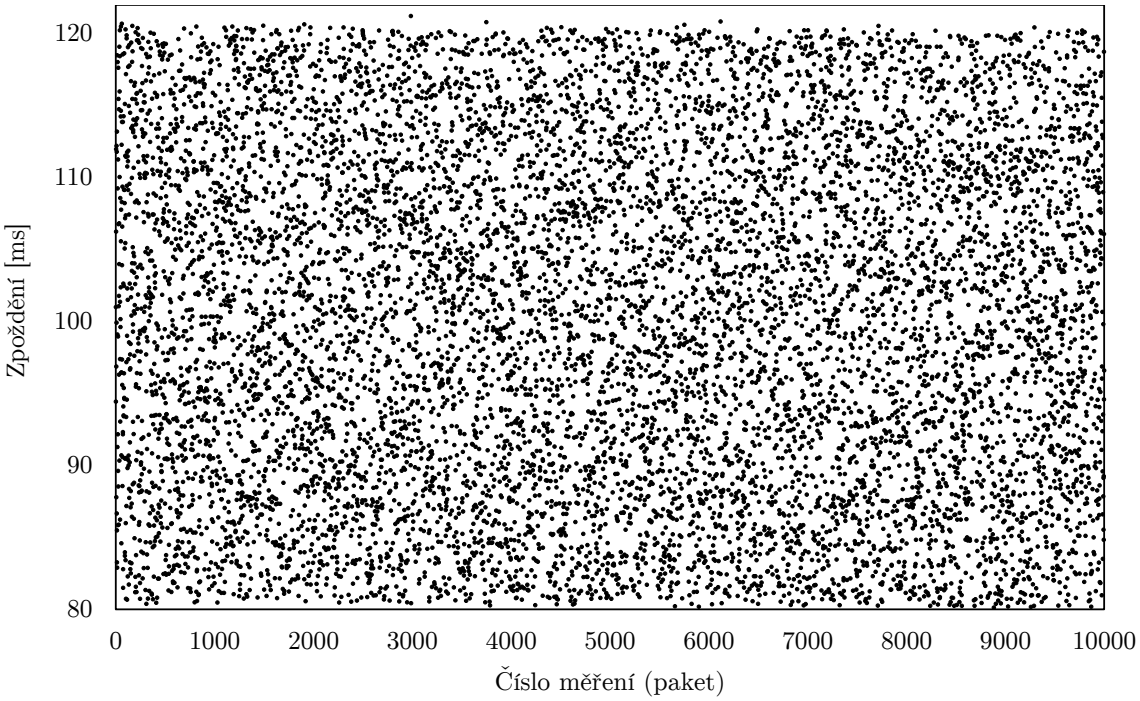
Obr. 5.9: Histogram zpoždění pro pareto rozložení (100 ± 20 ms)



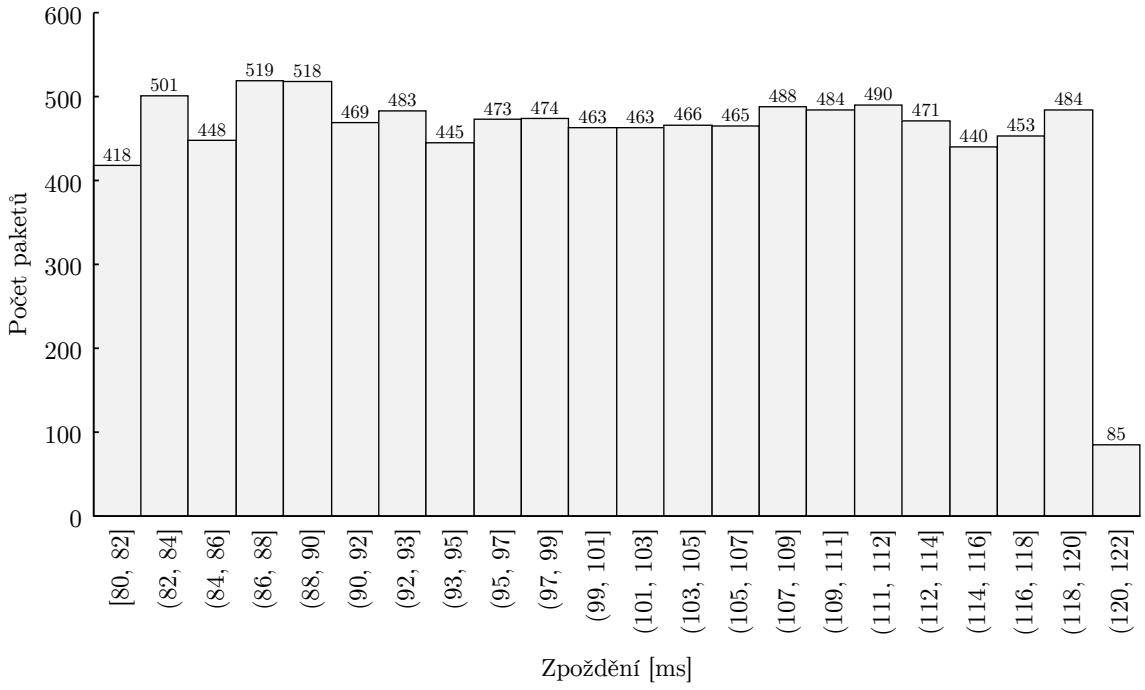
Obr. 5.10: Naměřené hodnoty zpoždění pro pareto-normální rozložení (100 ± 20 ms)



Obr. 5.11: Histogram zpoždění pro pareto-normální rozložení (100 ± 20 ms)



Obr. 5.12: Naměřené hodnoty zpoždění pro rovnoměrné rozložení (100 ± 20 ms)



Obr. 5.13: Histogram zpoždění pro rovnoměrné rozložení (100 ± 20 ms)

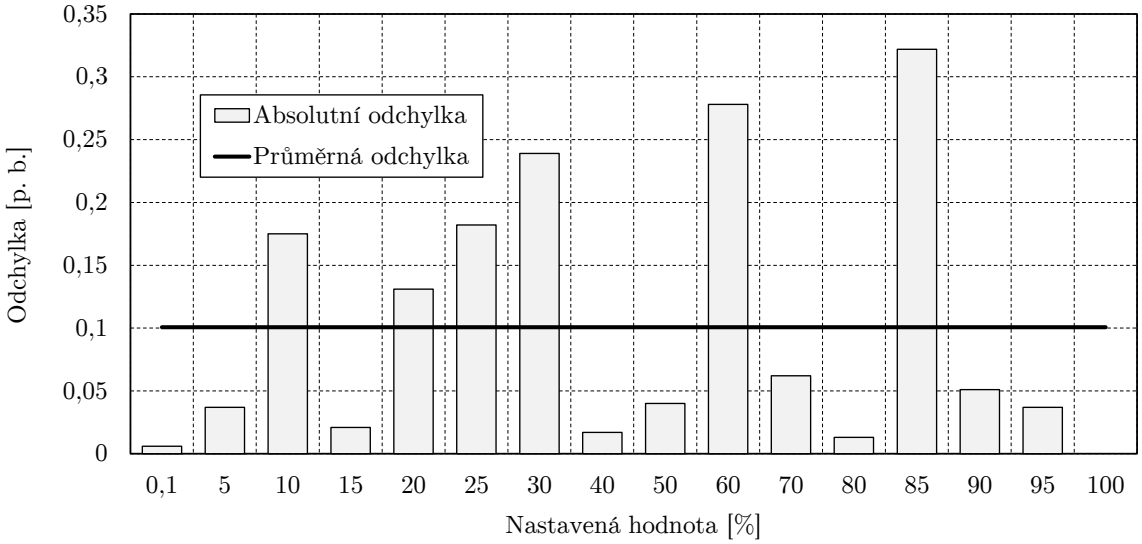
5.2.3 Ztrátovost

Ztrátovost byla měřena nástrojem `ping`. Tento nástroj se běžně využívá k testování konektivity mezi zařízeními v síti pomocí ICMP (Internet Control Message Protocol) zpráv, zobrazuje však další užitečné údaje, jako počet přijatých odpovědí na ICMP dotazy. Právě počet přijatých, respektive odeslaných ICMP zpráv byl použit k určení skutečné ztrátovosti. Konfigurace příkazu `ping` pro měření ztrátovosti je na výpisu B.8. Pro každou nastavenou hodnotu ztrátovosti bylo odesláno celkově sto tisíc ICMP paketů. Výsledky měření ztrátovosti jsou v tabulce 5.3. V rozhraní emulátoru bylo postupně nastaveno šestnáct hodnot ztrátovosti v rozmezí od 0,1 % do 100 %, viz první sloupec tabulky. Ve druhém sloupci jsou naměřené hodnoty ztrátovosti, ve zbylých sloupcích jsou vypočtené absolutní a relativní odchylky.

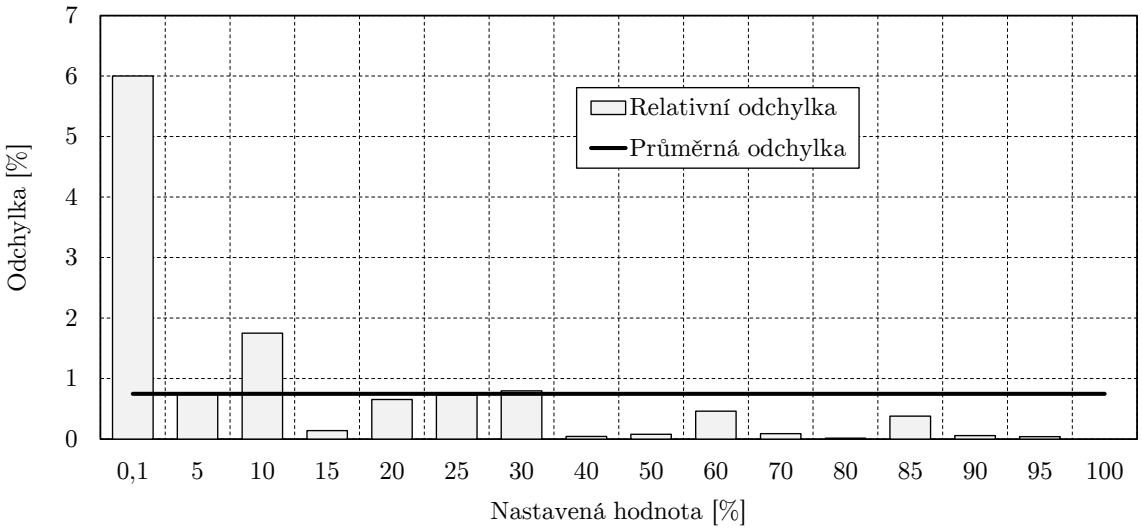
Tab. 5.3: Výsledky měření ztrátovosti

Nastavená hodnota [%]	**Naměřená hodnota [%]**	**Absolutní odchylka [p. b.]**	**Relativní odchylka [%]**
0,1	0,094	0,006	6
5	4,963	0,037	0,74
10	10,175	0,175	1,75
15	14,979	0,021	0,14
20	19,869	0,131	0,655
25	25,182	0,182	0,728
30	30,239	0,239	0,797
40	39,983	0,017	0,0425
50	49,96	0,04	0,08
60	59,722	0,278	0,463
70	70,062	0,062	0,089
80	79,987	0,013	0,0163
85	85,322	0,322	0,379
90	90,051	0,051	0,0567
95	95,037	0,037	0,039
100	100	0	0

Z grafu absolutní odchylky ztrátovosti na obrázku 5.14 je vidět, že při některých nastavených hodnotách odchylka poněkud vzrostla, při žádném z měření však nepřekročila hodnotu 0,35 procentního bodu. Jak je patrné z grafu na obrázku 5.15, relativní odchylka se u téměř všech měření pohybovala pod průměrnou odchylkou (0,75 %) nebo v jejím okolí, až na měření nastavené hodnoty ztrátovosti 0,1 %, kdy dosáhla 6 %.



Obr. 5.14: Absolutní odchylky ztrátovosti



Obr. 5.15: Relativní odchylky ztrátovosti

5.2.4 Chybovost

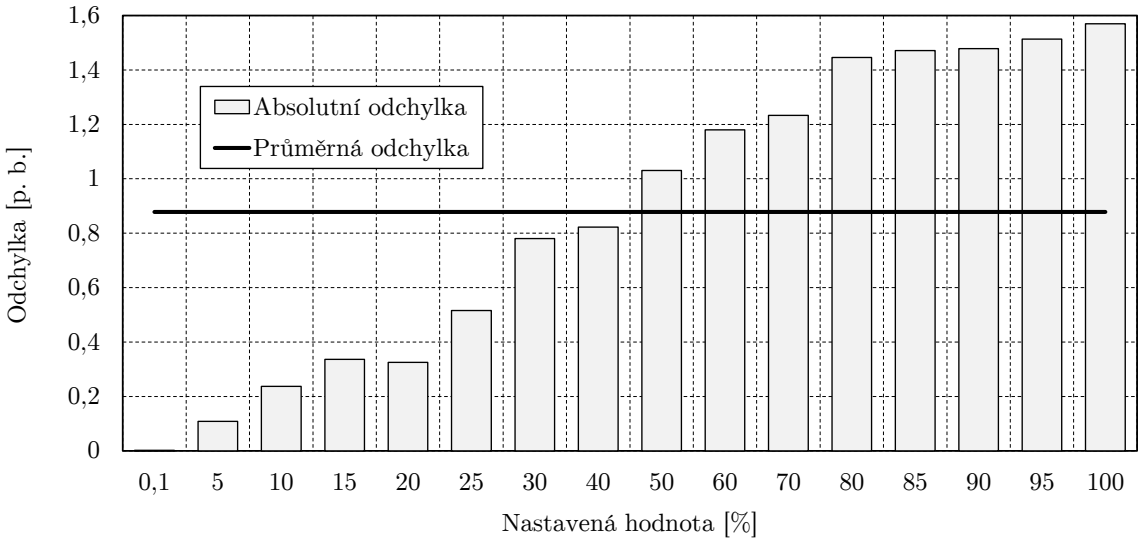
Za účelem generování paketů pro měření chybovosti byl použit nástroj `iperf3` v režimu klienta, viz výpis B.6. Velikost jedné zprávy a celkové množství odeslaných dat bylo nastaveno tak, aby v rámci dílčího měření odešlo přibližně půl miliónu paketů. Příjem paketů na druhém virtuálním stroji obstaral opět nástroj `iperf3`, tentokrát v režimu serveru. Pro ukládání paketů do `.pcap` souborů byl použit program `tcpdump`.

Výsledný skript v jazyce Bash spuštěný na straně serveru je na výpisu B.7. V tabulce 5.4 jsou nastavené a naměřené hodnoty chybovosti společně s vypočtenými odchylkami.

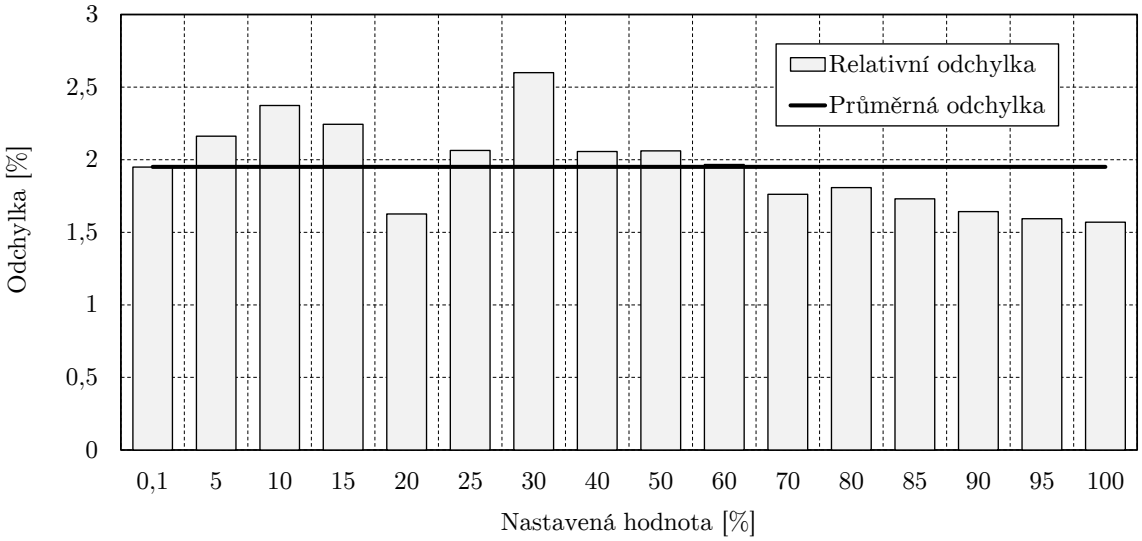
Tab. 5.4: Výsledky měření chybovosti

Nastavená hodnota [%]	**Naměřená hodnota [%]**	**Absolutní odchylka [p. b.]**	**Relativní odchylka [%]**
0,1	0,098051	0,0019495	1,9495
5	4,8919	0,10811	2,1622
10	9,7627	0,23734	2,3734
15	14,663	0,33667	2,2445
20	19,675	0,3251	1,6255
25	24,484	0,51604	2,0642
30	29,22	0,77996	2,5999
40	39,177	0,82261	2,0565
50	48,97	1,0304	2,0609
60	58,82	1,1798	1,9663
70	68,767	1,2329	1,7613
80	78,554	1,4462	1,8077
85	83,529	1,4711	1,7307
90	88,521	1,4788	1,6431
95	93,487	1,5133	1,5929
100	98,43	1,5698	1,5698

Na obrázku 5.16 je graf absolutní odchylky chybovosti. Absolutní odchylka rostla pro vyšší nastavené hodnoty chybovosti, nepřekročila však hodnotu 1,6 procentního bodu. Z grafu na obrázku 5.17 lze pozorovat, že relativní odchylka se ve všech případech pohybovala kolem průměrné odchylky (1,95 %) a při žádném z měření nepřekročila hodnotu 3 %.



Obr. 5.16: Absolutní odchylky chybovosti



Obr. 5.17: Relativní odchylky chybovosti

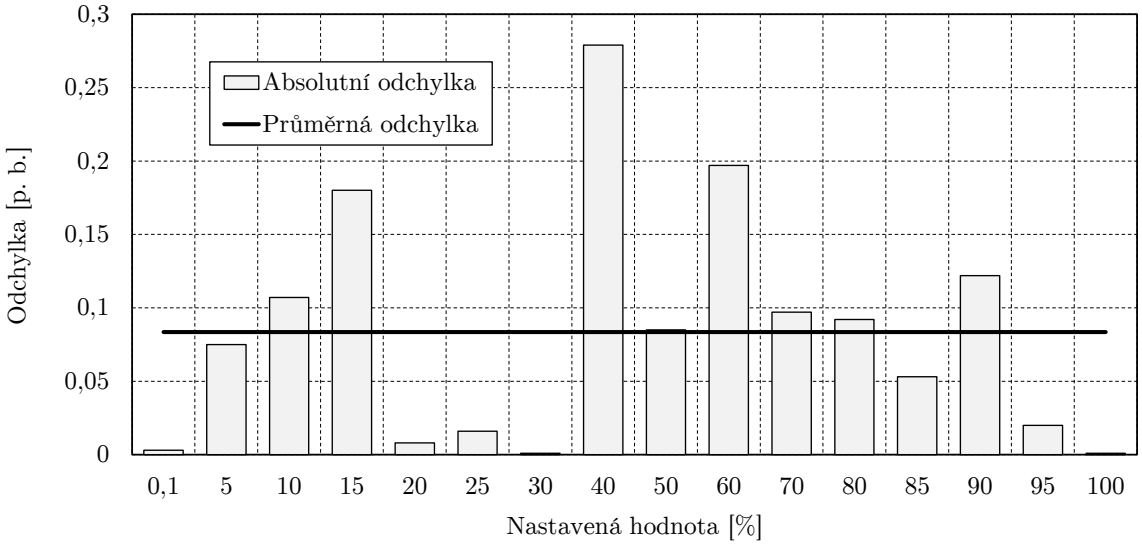
5.2.5 Duplikace

Nástroj `ping` použitý při měření ztrátovosti zobrazuje také počet duplicitních odpovědí na ICMP dotazy. Proto byl využit i pro měření parametru duplikace. Konfigurace příkazu `ping` spuštěného na straně klienta je na výpisu B.9. Odesláno bylo celkem půl miliónu ICMP paketů pro každou nastavenou hodnotu duplikace. Naměřená data jsou uvedena v tabulce 5.5. Stejně jako v předchozích případech byla dílčí měření provedena pro šestnáct hodnot v rozsahu od 0,1 % do 100 %.

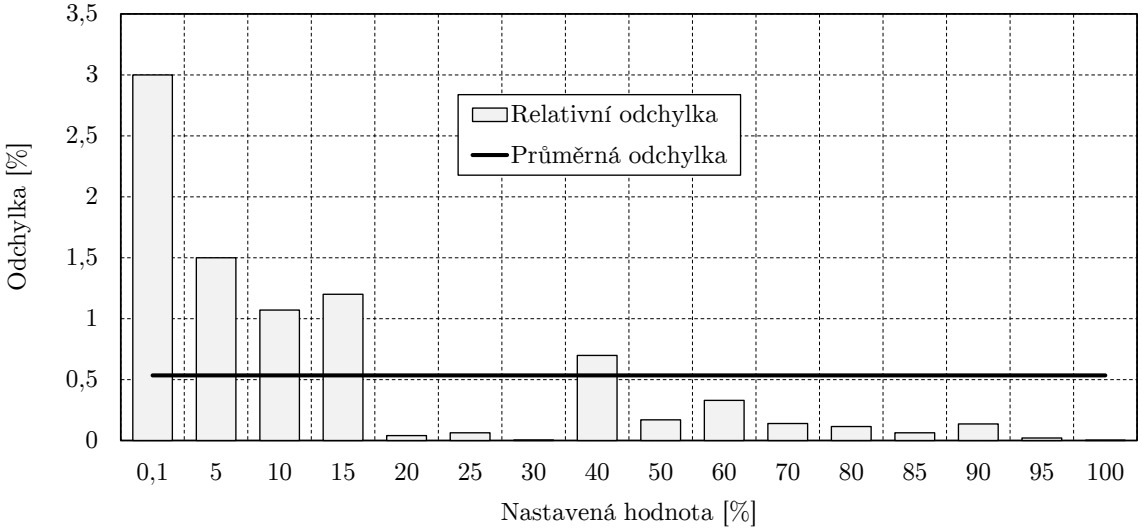
Tab. 5.5: Výsledky měření duplikace

Nastavená hodnota [%]	**Naměřená hodnota [%]**	**Absolutní odchylka [p. b.]**	**Relativní odchylka [%]**
0,1	0,097	0,003	3
5	4,925	0,075	1,5
10	10,107	0,107	1,07
15	14,82	0,18	1,2
20	19,992	0,008	0,04
25	24,984	0,016	0,064
30	29,999	0,001	0,0033
40	40,279	0,279	0,6975
50	50,085	0,085	0,17
60	60,197	0,197	0,3283
70	69,903	0,097	0,1386
80	79,908	0,092	0,115
85	85,053	0,053	0,0624
90	90,122	0,122	0,1356
95	94,98	0,02	0,0211
100	99,999	0,001	0,001

Vypočtené absolutní, respektive relativní odchylky z předchozí tabulky byly vyneseny do grafů na obrázcích 5.18, respektive 5.19. Absolutní odchylka naměřené hodnoty od nastavené hodnoty se pohybovala v rozmezí do 0,3 procentního bodu s průměrnou hodnotou odchylky přibližně 0,08 procentního bodu. Tyto odchylky jsou s vyššími nastavenými hodnotami duplikace stále více zanedbatelné, jak lze pozorovat z grafu relativní odchylky. Jedná se tedy o podobný trend, jako byl pozorován u parametru konstantního zpoždění a parametru ztrátovosti.



Obr. 5.18: Absolutní odchylky duplikace



Obr. 5.19: Relativní odchylky duplikace

5.3 Vyhodnocení výsledků

V předchozí sekci byla otestována funkčnost emulátoru měřeními několika vybraných parametrů. V průběhu měření byly požadované hodnoty nastavovány v uživatelském rozhraní emulátoru. Pro vyhodnocení přesnosti emulace jsou podstatné zejména relativní odchylky, které udávají míru chyby vůči nastavené hodnotě. V některých krajních případech relativní odchylka dosáhla hodnot 6 % až 8 %, v naprosté většině

měření se však pohybovala na hranici 3 % nebo daleko pod ní. Přehled průměrných relativních odchylek měřených parametrů je uveden v tabulce 5.6. Z výsledků měření lze konstatovat, že aplikace emulátoru správně ovlivňuje přenosové parametry definované uživatelem v jejím rozhraní.

Tab. 5.6: Průměrné relativní odchylky přenosových parametrů

Přenosový parametr	**Průměrná relativní odchylka** [%]
Propustnost	2,9061
Konstantní zpoždění	1,1916
Ztrátovost	0,74842
Chybovost	1,9505
Duplikace	0,53417

Nelze opomenout, že přesnost emulace NetEm je do určité míry dána hardwarovou konfigurací stroje, na kterém je spuštěna. Výsledky měření jsou rovněž ovlivněny použitou měřicí soustavou. Účelem této kapitoly bylo především otestování funkčnosti vyvinutého modulu síťového emulátoru zakomponovaného do prostředí Apache JMeter. Komplexním vyhodnocením přesnosti samotného nástroje NetEm se podrobněji zabývají publikace [13], [14] nebo [31].

Závěr

V úvodní kapitole práce byly nejprve definovány a ujasněny některé klíčové pojmy. Následně byla provedena analýza přenosových parametrů vhodných k emulaci. Při analýze byly využity odborné anglické i české publikace a dokumentace. Pro definici parametru duplikace a parametru záměna pořadí byly využity dokumenty RFC, jelikož zmíněná literatura tyto parametry nepopisuje dostatečně detailně.

Druhá kapitola se věnuje metodám síťového testování, konkrétně síťové simulaci, síťovým testovacím prostředím a síťovým emulátorům. Smyslem této kapitoly je uvést síťovou emulaci, jakožto hlavní téma této práce, do širšího kontextu. Protože je síťové testování velmi rozsáhlá oblast, byla pro každou ze zmíněných kategorií popsána pouze jedna vybraná metoda. Důvodem bylo především ušetření místa.

Ve třetí kapitole je uveden přehled v současnosti používaných síťových emulátorů. Více prostoru bylo věnováno emulátorům síťového spojení, protože bylo zřejmé, že právě tímto směrem se práce bude ubírat. Jedná se o emulátory DummyNet, NIST-Net a NetEm. Na konci kapitoly byly zmíněné emulátory síťového spojení porovnány.

Samotná realizace síťového emulátoru je popsána ve čtvrté kapitole. Podle zadání práce má být emulátor zakomponován do systému Apache JMeter. S tímto vědomím, a na základě srovnání emulátorů síťových spojení z předchozí kapitoly, byl pro tento účel zvolen emulátor NetEm. Na začátku kapitoly je krátce vysvětlena architektura softwaru JMeter. Následně je popsán způsob ovládání nástroje pro emulaci a filtrování. Zbylé dvě části kapitoly se věnují sestavení datové struktury emulátoru a tvorbě grafického uživatelského rozhraní.

V poslední kapitole je ověřena funkčnost emulátoru na několika vybraných scénářích. Pro testování byly zvoleny parametry propustnost, zpoždění (konstantní a proměnné), ztrátovost, chybovost a duplikace. Měření bylo uskutečněno s využitím dvou virtuálních strojů. Naměřené hodnoty a vypočtené odchylky ukazují, že emulace dílčích parametrů je funkční a dosahuje adekvátní přesnosti.

Síťový emulátor integrovaný do prostředí Apache JMeter poskytuje uživatelsky přívětivé rozhraní pro konfiguraci parametrů emulace a filtrování realizovaných nástroji balíčku Traffic Control. Software emulátoru byl navržen s důrazem na jeho budoucí rozšiřitelnost o další parametry emulace, filtrovací podmínky a další funkce.

Literatura

[1] JEŘÁBEK, Jan. *Komunikační technologie*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. ISBN 978-80-214-4713-4.

[2] STALLINGS, William. *Data and computer communications*. 10th Edition. Boston: Pearson, 2014. ISBN 978-1-292-01438-8.

[3] FOROUZAN, Behrouz A. *Data Communications and Networking with TCP/IP Protocol Suite*. 6th Edition. McGraw Hill, 2021. ISBN 978-0078022098.

[4] KUROSE, James a ROSS, Keith. *Computer Networking: A Top-Down Approach*. 7th Edition. Pearson, 2016. ISBN 978-0133594140.

[5] PETERSON, Larry L. a DAVIE, Bruce S. *Computer networks: a systems approach*. 5th Edition. Burlington, Mass.: Morgan Kaufmann, 2012. ISBN 978-0-12-385059-1.

[6] COMER, Douglas E. *Computer Networks and Internets*. 6th Edition. Pearson, 2014. ISBN 978-0133587937.

[7] DUMAN, İbrahim a ELİİYİ, Uğur. Performance Metrics and Monitoring Tools for Sustainable Network Management. Online. *Bilişim Teknolojileri Dergisi*. 2021, roč. 14, č. 1, s. 37-51. ISSN 1307-9697. Dostupné z: `https://doi.org/10.17671/gazibtd.780504`. [cit. 2024-10-08].

[8] ELAHI, Ata a CUSHMAN, Alex. *Computer Networks: Data Communications, Internet and Security*. Online. Cham: Springer International Publishing, 2024. ISBN 978-3-031-42017-7. Dostupné z: `https://doi.org/10.1007/978-3-031-42018-4`. [cit. 2024-10-12].

[9] MILLS, David L. Network Time Protocol (NTP). Online. 1985. RFC 958. Request for Comments. RFC Editor. Dostupné z: `https://www.rfc-editor.org/info/rfc958`. [cit. 2025-04-25]

[10] UIJTERWAAL, Henk A.J. A One-Way Packet Duplication Metric. Online. 2009. RFC 5560. Request for Comments. RFC Editor. Dostupné z: `https://www.rfc-editor.org/info/rfc5560`. [cit. 2025-04-27]

[11] MORTON, Al; RAMACHANDRAN, Gomathi; SHALUNOV, Stanislav; CIAVATTONE, Len; PERSER, Jerry. Packet Reordering Metrics. Online. 2006. RFC 4737. Request for Comments. RFC Editor. Dostupné z: `https://www.rfc-editor.org/info/rfc4737`. [cit. 2025-04-28]

[12] ČESKÝ TELEKOMUNIKAČNÍ ÚŘAD. ČTÚ-6 145 / 2017-620, *Metodika pro měření a vyhodnocení datových parametrů pevných sítí elektronických komunikací*. Verze 2.1. Praha, 2021.

[13] HEMMINGER, Stephen. Network emulation with NetEm. Online. *Linux Conf Au*. 2005, s. 1-8. Dostupné z: `https://www.researchgate.net/publication/228619146_Network_emulation_with_NetEm`. [cit. 2024-10-08].

[14] JURGELIONIS, Audrius; LAULAJAINEN, Jukka-Pekka; HIRVONEN, Matti a WANG, Alf Inge. An Empirical Study of NetEm Network Emulation Functionalities. Online. In: *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, s. 1-6. ISBN 978-1-4577-0637-0. Dostupné z: `https://doi.org/10.1109/ICCCN.2011.6005933`. [cit. 2024-10-08].

[15] GRABOVSKÝ, Štěpán; ZEMAN, Václav a ČLUPEK, Vlastimil. Síťový emulátor přenosových parametrů datových sítí. Online. *Elektrorevue*. 2018, roč. 20, č. 4. ISSN 1213 - 1539. [cit. 2025-05-11].

[16] GOMEZ, Jose; KFOURY, Elie F.; CRICHIGNO, Jorge a SRIVASTAVA, Gautam. A survey on network simulators, emulators, and testbeds used for research and education. Online. *Computer Networks*. 2023, roč. 237, article 110054, s. 1–42. ISSN 13891286. Dostupné z: `https://doi.org/10.1016/j.comnet.2023.110054`. [cit. 2024-10-05].

[17] AHRENHOLZ, Jeff; DANILOV, Claudiu; HENDERSON, Thomas R. a KIM, Jae H. CORE: A real-time network emulator. Online. In: *MILCOM 2008 - 2008 IEEE Military Communications Conference*. IEEE, 2008, s. 1-7. ISBN 978-1-4244-2676-8. Dostupné z: `https://doi.org/10.1109/MILCOM.2008.4753614`. [cit. 2024-12-11].

[18] GURUPRASAD, S.; RICCI, R. a LEPREAU, J. Integrated Network Experimentation using Simulation and Emulation. Online. In: *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*. IEEE, 2005, s. 204-212. ISBN 0-7695-2219-X. Dostupné z: `https://doi.org/10.1109/TRIDNT.2005.21`. [cit. 2024-12-09].

[19] SOFTWARE SOLUTIONS STUDIO. *Introduction to continuous-time simulation*. Online. Software Solutions Studio. 2022. Dostupné z: `https://softwaresim.com/blog/introduction-to-continuous-time-simulation/`. [cit. 2024-12-10].

[20] SOFTWARE SOLUTIONS STUDIO. *Introduction to discrete-time simulation*. Online. Software Solutions Studio. 2022. Dostupné z: https://softwaresim.com/blog/introduction-to-discrete-time-simulation/. [cit. 2024-12-10].

[21] HELLER, Brandon David. *Reproducible network research with high-fidelity emulation*. Online, Disertace, vedoucí Nick McKeown. Stanford: Stanford University, Department of Computer Science, 2013. Dostupné také z: https://purl.stanford.edu/zk853sv3422.

[22] TSAI, Pang-Wei; PICCIALLI, Francesco; TSAI, Chun-Wei; LUO, Mon-Yen a YANG, Chu-Sing. Control frameworks in network emulation testbeds: A survey. Online. *Journal of Computational Science*. 2017, roč. 22, s. 148-161. ISSN 18777503. Dostupné z: https://doi.org/10.1016/j.jocs.2017.03.003. [cit. 2024-12-09].

[23] JAHROMI, Hamed Z.; HINES, Andrew a DELANEV, Declan T. Towards Application-Aware Networking: ML-Based End-to-End Application KPI/QoE Metrics Characterization in SDN. Online. In: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2018, s. 126-131. ISBN 978-1-5386-4646-5. Dostupné z: https://doi.org/10.1109/ICUFN.2018.8436625. [cit. 2024-12-09].

[24] PATEL, Ronit L.; PATHAK, Maharshi J. a NAYAK, Amit J. Survey on Network Simulators. Online. *International Journal of Computer Applications*. 2018, roč. 182, č. 21, article 0975 — 8887, s. 1-8. Dostupné z: https://www.researchgate.net/publication/333262623_Survey_on_Network_Simulators. [cit. 2024-11-13].

[25] *Cisco Modeling Labs User Guide*. Online. Cisco Systems, 2014. Dostupné z: https://www.cisco.com/c/en/us/td/docs/cloud_services/cisco_modeling_labs/v100/configuration/guide/b_cml_user_guide.html. [cit. 2024-12-10].

[26] RIZZO, Luigi. Dummynet and forward error correction. Online. *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. 1998, s. 1-9. Dostupné z: https://www.usenix.org/conference/1998-usenix-annual-technical-conference/dummynet-and-forward-error-correction. [cit. 2024-10-08].

[27] HAMDANI, Mohammad; GIYANA, Rifqi Fajar; ANANDA KUSUMA, A. A. N. a PALOKOTO, Toto Bachtiar. On Modeling a Bottleneck Link using Dummynet. Online. In: *2023 International Conference on Radar, Antenna,*

Microwave, Electronics, and Telecommunications (ICRAMET). IEEE, 2023, s. 73-78. ISBN 979-8-3503-4389-2. Dostupné z: https://doi.org/10.1109/ICRAMET60171.2023.10366590. [cit. 2024-10-08].

[28] CARSON, Mark a SANTAY, Darrin. NIST Net: a Linux-based network emulation tool. Online. *ACM SIGCOMM Computer Communication Review*. 2003, roč. 33, č. 3, s. 111-126. ISSN 0146-4833. Dostupné z: https://doi.org/10.1145/956993.957007. [cit. 2024-10-08].

[29] NUSSBAUM, Lucas a RICHARD, Olivier. A Comparative Study of Network Link Emulators. Online. *Communications and Networking Simulation Symposium (CNS'09)*. 2009, s. 1-8. Dostupné z: https://inria.hal.science/inria-00425613/. [cit. 2024-10-08].

[30] HANEBERG, Lukas; HAUSER, Eric a GALLENMÜLLER, Sebastian. Increase the Latency: Emulation Approaches for Real Network Conditions. Online. *Network*. 2023, roč. 41, s. 1-5. Dostupné z: https://api.semanticscholar.org/CorpusID:269793418. [cit. 2024-10-08].

[31] MOE, Anders G. *Implementing Rate Control in NetEm: Untying the NetEm/tc tangle*. Online, Master thesis, vedoucí Andreas Petlund, Pål Halvorsen, Carsten Griwodz. Oslo: University of Oslo, 2013. Dostupné také z: https://www.duo.uio.no/bitstream/handle/10852/37459/Moe-Master.pdf.

Seznam symbolů a zkratek

TCP	spolehlivý transportní protokol – Transmission Control Protocol
IP	internetový protokol – Internet Protocol
IETF	organizace pro standardizaci internetových technologií – Internet Engineering Task Force
RFC	dokumenty popisující internetové standardy – Request For Comments
ČTÚ	Český telekomunikační úřad
B	šířka pásma
$f_{\max}$	maximální kmitočet
$f_{\min}$	minimální kmitočet
$r_{\max}$	maximální přenosová rychlost
M	počet signálových úrovní
r	propustnost
r_{u}	propustnost uzlu
n	množství dat
d_{u}	zpoždění uzlu
r_{t}	propustnost trasy
t_{p}	čas přijetí paketu
t_{o}	čas odeslání paketu
d_{s}	zpoždění signálu
d_{p}	zpoždění přenosu
d_{z}	zpoždění zpracování
d_{f}	zpoždění ve frontě
l	délka fyzického spoje
v	rychlost šíření signálu

v	velikost paketu
FIFO	fronta typu „první dovnitř, první ven“ – First In, First Out
d_{e}	zpoždění ve vstupní frontě
d_{x}	zpoždění ve výstupní frontě
d_{t}	zpoždění trasy
N	počet mezilehlých uzlů
d_{o}	obousměrné zpoždění
NTP	protokol pro synchronizaci hodin v počítačových sítích – Network Time Protocol
j	kolísání zpoždění paketu
t_{r}	referenční čas doručení paketu
t_{s}	skutečný čas doručení paketu
Δt	interval mezi odesláním dvou paketů
L	ztrátovost paketů
n_{z}	počet ztracených paketů
n_{o}	počet odeslaných paketů
E	chybovost paketů
n_{ch}	počet chybných paketů
D	duplikace paketů
n_{d}	počet duplicitních paketů
n_{p}	počet přijatých paketů
Z	záměna pořadí paketů
n_{z}	počet paketů doručených ve špatném pořadí
UDP	nespolehlivý transportní protokol – User Datagram Protocol
SDN	softwarově definovaná síť – Software-Defined Networking

PFIFO	paketová fronta typu FIFO – Packet FIFO
HTB	hierarchické rozdělování přenosové rychlosti – Hierarchical Token Bucket
CBQ	řízení přenosu podle tříd provozu – Class Based Queuing
RTC	hodiny reálného času – Real-Time Clock
ICMP	protokol řídících zpráv – Internet Control Message Protocol
HTTP	hypertextový protokol – HyperText Transfer Protocol
HTTPS	zabezpečený hypertextový protokol – HyperText Transfer Protocol Secure
SOAP	protokol pro výměnu strukturovaných informací – Simple Object Access Protocol
REST	architektonický styl pro návrh API – REpresentational State Transfer
FTP	protokol pro přenos souborů – File Transfer Protocol
JDBC	rozhraní pro připojení k databázi v Javě – Java DataBase Connectivity
LDAP	protokol pro přístup k adresářovým službám – Lightweight Directory Access Protocol
SMTP	protokol pro přenos elektronické pošty – Simple Mail Transfer Protocol
HTML	značkovací jazyk pro webové stránky – HyperText Markup Language
JSON	formát pro výměnu dat – JavaScript Object Notation
XML	značkovací jazyk pro strukturovaná data – eXtensible Markup Language
API	aplikační programové rozhraní – Application Programming Interface
JAR	archiv tříd Java – Java ARchive
GUI	grafické uživatelské rozhraní – Graphical User Interface
DSCP	kód pro rozlišení kvality služeb – Differentiated Services Code Point

ECN	oznámení o přetížení – Explicit Congestion Notification
LaF	vzhled a chování aplikace – Look and Feel
DNS	systém doménových jmen – Domain Name System
DHCP	protokol pro dynamickou konfiguraci uzlů v síti – Dynamic Host Configuration Protocol

Seznam příloh

A Sestavení a zprovoznění emulátoru **79**

B Skripty použité pro měření přenosových parametrů **80**

B.1 Instalace potřebných nástrojů 80

B.2 Měření propustnosti 80

B.3 Měření zpoždění 81

B.4 Měření chybovosti 82

B.5 Měření ztrátovosti 83

B.6 Měření duplikace 83

C Obsah elektronické přílohy **84**

A Sestavení a zprovoznění emulátoru

Po sestavení projektu (výpis A.1), přesunutí vygenerovaného JAR souboru do adresáře JMeteru (výpis A.2) a spuštění JMeteru (výpis A.3) je možné emulátor přidat do testovacího plánu JMeteru pravým kliknutím na Test Plan → Add → Non-Test Elements → Network Emulator.

Výpis A.1: Sestavení projektu emulátoru

```
$ cd emulator
$ ./gradlew build
```

Výpis A.2: Přesunutí JAR souboru

```
$ mv build/libs/jmeter-network-emulator.jar \
  ~/adresar/jmeteru/lib/ext
```

Výpis A.3: Spuštění JMeteru

```
$ cd ~/adresar/jmeteru
$ ./bin/jmeter
```

B Skripty použité pro měření přenosových parametrů

B.1 Instalace potřebných nástrojů

Výpis B.1: Skript pro instalaci potřebných nástrojů

```
1 #!/bin/bash
2 apt update
3 apt install -y iperf3
4 apt install -y iputils-ping
5 apt install -y tcpdump
6 apt install -y tshark
7 apt install -y python3
```

B.2 Měření propustnosti

Výpis B.2: Klient pro měření propustnosti

```
$ iperf3 -c 10.0.0.100 -p 65495 -u -b 0 -t 500
```

Výpis B.3: Server pro měření propustnosti

```
$ iperf3 -s -p 65495 --json output.json
```

B.3 Měření zpoždění

Výpis B.4: Klient pro měření zpoždění

```
import socket
import time
import struct
import csv

SERVER_IP = '10.0.0.100'
PORT = 65495
MEASUREMENTS = 10000
CSV_FILE = 'results.csv'
results = []

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    for i in range(MEASUREMENTS):
        t1 = time.time()
        s.sendto(b'ping', (SERVER_IP, PORT))
        data, _ = s.recvfrom(1024)
        t2 = struct.unpack('d', data)[0]
        delay = t2 - t1

        print(f"[{i+1}] Delay (client -> server): {delay:.6f} s")
        results.append([i + 1, delay])

        time.sleep(0.1)

with open(CSV_FILE, 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['Measurement', 'Delay (s)'])
    writer.writerows(results)
```

Výpis B.5: Server pro měření zpoždění

```
import socket
import time
import struct

HOST = '10.0.0.100'
PORT = 65495

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.bind((HOST, PORT))
    print(f"Server runs on {HOST}:{PORT}")

    while True:
        data, addr = s.recvfrom(1024)
        t2 = time.time()
        s.sendto(struct.pack('d', t2), addr)
```

B.4 Měření chybovosti

Výpis B.6: Klient pro měření chybovosti

```
$ iperf3 -c 10.0.0.100 -p 65495 -u -l 1400 -n 700000000 -b 0
```

Výpis B.7: Server pro měření chybovosti

```
#!/bin/bash
SRC_IP="10.0.0.200"
DST_IP="10.0.0.100"
DST_PORT=65495
DUMP_FILE="corrupt.pcap"

tcpdump udp and src host "$SRC_IP" and dst host "$DST_IP" and dst
   port "$DST_PORT" -w "$DUMP_FILE" &
TCPDUMP_PID=$!
sleep 1

iperf3 -s -p $DST_PORT --one-off

kill "$TCPDUMP_PID"
```

B.5 Měření ztrátovosti

Výpis B.8: Klient pro měření ztrátovosti

```
$ ping 10.0.0.100 -f -i 0.002 -c 100000
```

B.6 Měření duplikace

Výpis B.9: Klient pro měření duplikace

```
$ ping 10.0.0.100 -f -i 0.002 -c 500000
```

C Obsah elektronické přílohy

Obsahem elektronické přílohy jsou zdrojový kód emulátoru, skript pro automatizované sestavení projektu a skripty použité při měření přenosových parametrů.

- `/` — kořenový adresář přiloženého archivu
  - `emulator` — kořenový adresář projektu
    - `gradle` — konfigurační soubory pro sestavení
    - `src` — zdrojový kód
      - `main`
        - `java`
          - `cz`
            - `vutbr`
              - `networkemulator`
                - `action` — akce
                - `controller` — kontrolér
                - `gui` — grafické rozhraní
                - `model` — datová reprezentace
                - `tc` — ovládání nástroje `tc`
                - `utils` — pomocné nástroje
                - `verification` — verifikátory
                - `EmulatorTestElement.java` — reprezentace emulátoru
        - `resources`
          - `cz`
            - `vutbr`
              - `networkemulator`
                - `images` — ikony a obrázky
                - `messages.properties` — textové řetězce
    - `LICENSE` — licenční soubor projektu
    - `build.gradle` — konfigurace sestavení
    - `gradlew` — spouštěč sestavení
    - `settings.gradle` — nastavení projektové struktury
  - `mereni` — skripty pro měření a testování
    - `corruption_server.sh`
    - `delay_client.py`
    - `delay_server.py`
    - `setup.sh`