VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Martin Štainer

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 TESTER
NETWORK TESTER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Štainer**
AUTHOR

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

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Ústav telekomunikací

Student: Bc. Martin Štainer ***ID:*** 165695

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Síťový tester

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je návrh a realizace systému pro měření přenosových parametrů datových sítí. Jednotlivé přenosové parametry specifikujte a popište metodiky využívané pro jejich měření. Na základě rozboru navrhněte řešení, které umožní komplexní měření datových sítí z hlediska přenosových parametrů. Výstupem práce bude softwarový tester, který bude zakomponován do již existujícího systému pro testování sítí, jehož základem je Apache JMeter.

DOPORUČENÁ LITERATURA:

[1] BRANDER, S. RFC 2544 - Benchmarking Methodology for Network Interconnect Devices, IETF, 1999.

[2] ERINLE, Bayo. Performance Testing with JMeter 2.9. Packt Publishing Ltd, 2013.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

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

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Práce se zaobírá problematikou měření parametrů datových sítí. Byly navrhnuty metodiky měření a koncepce testeru. Podle vytvořené koncepce testeru bylo implementováno rozšíření pro nástroj Apache JMeter. Podle navržené metodiky měření byly provedeny dva experimentální testy s cílem zjistit výkonnost protokolu QUIC v porovnání s TCP.

KLÍČOVÁ SLOVA

síťový tester, QUIC, TCP, propustnost, zpoždění

ABSTRACT

The objective of this thesis is the issue of network parameters measuring. Measurement methodology and tester concept were designed. Based on the designed tester concept a plugin for Apache JMeter was implemented. Two experimental tests were run based on the methodology designed, with their objectives set to explore the difference in performance between QUIC and TCP protocols.

KEYWORDS

network tester, TCP, QUIC, throughput, delay

ŠTAINER, Martin. *Síťový tester*. Brno, Rok, 75 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Síťový tester“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících 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

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

Seznam obrázků

Seznam tabulek

Úvod

Z důvodu neustále se zvyšujícímu vlivu IT na rozličné obory, roste množství přenášených dat pomocí počítačových sítí. Spolehlivost připojení může ovlivnit fungování firem, institucí a mnoho dalších. Z tohoto důvodu je potřeba testovat sítě a síťové prvky.

Tato práce se věnuje oblasti měření parametrů datových sítí, jejíž realizací je síťový tester pro Java aplikaci Apache JMeter. Výstup práce bude zakomponován již do existujícího systému pro testování sítí. V teoretické části jsou popsány síťové parametry spolu s jejich metodikami měření.

V rámci první kapitoly jsou vysvětleny jednotlivé parametry, kterými jsou zpoždění, propustnost a ztrátovost. Druhá kapitola se věnuje již standardizovaným metodikám měření jako jsou IETF RFC2544, ITU Y.1564 nebo IETF RFC6349. První dvě zmíněné metodiky jsou určeny pro testování ethernetových sítí, přičemž RFC2544 bylo vytvořeno pro testování jednotlivých síťových rozhraní v laboratorních podmínkách, zatímco ITU-T Y.1564 nabízí komplexní testování sítě v jediném testu. Výsledek testu umožňuje poskytovatelům a zákazníkům definovat jasné chování v SLA smlouvách. Poslední zmíněný standard RFC6349 je soubor doporučení při měření TCP propustnosti. Součástí kapitoly je také návrh vlastní metodiky pro měření datových parametrů.

Další kapitola se věnuje transportnímu protokolu QUIC. Tento nový, bezpečný a spolehlivý protokol je postavený nad UDP s cílem snížit zpoždění v porovnání s TCP. QUIC kombinuje funkce HTTP protokolu (multiplexování a řízení toku), TLS protokolu (bezpečnost) a také řadě TCP protokolu (řízení toku, spolehlivost a uzavírání spojení). QUIC přináší nové zajímavé funkce jako migrace připojení, 0-RTT uzavření spojení nebo multiplexování. QUIC funguje na aplikační vrstvě. Díky této vlastnosti existují desítky implementací a je zaručeno rychlé nasazení do produkce bez nutnosti zasahovat do stávajícího softwaru či hardwaru.

Pátá kapitola je věnována popisu implementace, kompilaci rozšíření a problémům řešených při programování. Byla vytvořena tři rozšíření pro nástroj JMeter. První rozšíření, které spouští nástroj iPerf a druhé, které spouští nástroj tPerf. Pro vizualizaci výsledků bylo napsáno rozšíření třetí, které umožňuje jednotlivé protokoly porovnat v čase.

Experimentálním testům je věnována poslední kapitola, ve které se autor snaží porovnat aktuálně používaný protokol TCP a jeho možného budoucího nástupce QUIC. Při prvním experimentu je využita implementace QUIC od firmy Facebook – MVFST. Tento experiment se soustředí na měření propustnosti s emulací zhoršených síťových podmínek. Druhý experiment je orientován na porovnání balíčku protokolů HTTP, TLS, TCP oproti HTTP, QUIC. Měření v druhém experimentu

jsou provedena pomocí nástroje Lighthouse, který dokáže změřit čas potřebný k načtení webové stránky pomocí několika metrik.

1 Přenosové parametry

V současné době jsou bezpečnost a výkon dva klíčové faktory při evaluaci telekomunikačních sítí [2]. Tato kapitola popisuje základní parametry jako zpoždění, ztrátovost paketů nebo propustnost, měřených při výkonnostních testech. Metodiky jejich měření jsou popsány v následující kapitole.

1.1 Propustnost

Standard IETF RFC 1242 definuje propustnost[1] jako maximální bitovou rychlost, při které žádný z přijímaných rámců není zahozen. Vzorec 1.1 vyjadřuje propustnost p jako počet odeslaných bitů d za daný čas t. Propustnost je typicky udávána v bitech za sekundu [b/s]. Zdroj [10] uvádí pojem goodput, který vyjadřuje propustnost bez řídících dat, tj. pouze užitečné informace.

$$p\ [b/s] = \frac{d\ [b]}{t\ [s]} \tag{1.1}$$

Hodnota propustnosti pomáhá výrobcům udávat výkonnost zařízení pomocí jedné hodnoty. Ztráta jednoho rámce může způsobit velké zpoždění z důvodu zapouzdření a identifikace ztráty na vyšších vrstvách. V závislosti na tom je dobré vědět, kolik provozu dokáže zařízení odbavit za daný čas. Hodnota propustnosti by měla být spočítána pro každou velikost rámce zvlášť.[5]

1.2 Ztrátovost

Standard IETF RFC 1242 definuje ztrátovost[2] jako množství rámců, které měly být přeposlány, ale nebyly z důvodu nedostatečného množství zdrojů. Tato jednotka je udávána v procentech a může být ukazatelem chování zařízení v nepříznivých podmínkách, např. při přetížení sítě [5]. Ztrátovost z lze zapsat vzorcem viz. 1.2, kde o je počet odeslaných rámců a p počet přijatých rámců.

$$z\ [\%] = \frac{o\ [-]}{p\ [-]} \cdot 100\ [\%] \tag{1.2}$$

[1] anglicky throughput
[2] anglicky frame error rate

1.3 Zpoždění

Standard RFC 1242 definuje zpoždění[3] jako interval mezi odesláním posledního a přijmutí prvního bitu rámce [5]. Jiný zdroj uvádí zpoždění jako čas mezi odesláním a přijetím packetu od zdroje k cíli [3].

Zpoždění je jednotka času, nejčastěji uváděná v milisekundách. Některé síťové služby jsou časově závislé a i malé hodnoty zpoždění mohou znatelně ovlivnit kvalitu služeb[5].

Složení zpoždění dle místa vzniku

- **Zpoždění signálu** je čas potřebný pro propagaci signálu v médiu
- **Přenosové zpoždění** definuje čas uzlu potřebný k vložení datové jednotky na síťové médium
- **Zpoždění zpracování** uvádí čas potřebný ke kontrole bitů, k určení cílové destinace, apod.
- **Zpoždění front** popisuje čas strávený ve frontách

Celkové zpoždění je vyjádřeno následujícím vzorcem:

$$z\ [s] = z_{zs}\ [s] + z_{pz}\ [s] + z_{zz}\ [s] + z_{zf}\ [s] \tag{1.3}$$

, kde index *zs* vyjadřuje zpoždění způsobené propagací signálu, *zp* je přenosové zpoždění, *zz* definuje zpoždění způsobené zpracováním a *zf* je zpoždění způsobené frontami.

RTT

Zpoždění může být měřeno pouze od zdroje k cíli (jednosměrně) nebo od zdroje k cíli a zpět (obousměrně). RFC 6349 definuje obousměrné zpoždění jako RTT (Round time trip). Jedná se o uplynulý čas mezi zasláním prvního bitu TCP segmentu a přijetí posledního bitu odpovídajícího TCP potvrzení. Výhodou obousměrného měření je jednodušší zpracování časové známky na iniciátorem testu. [6]

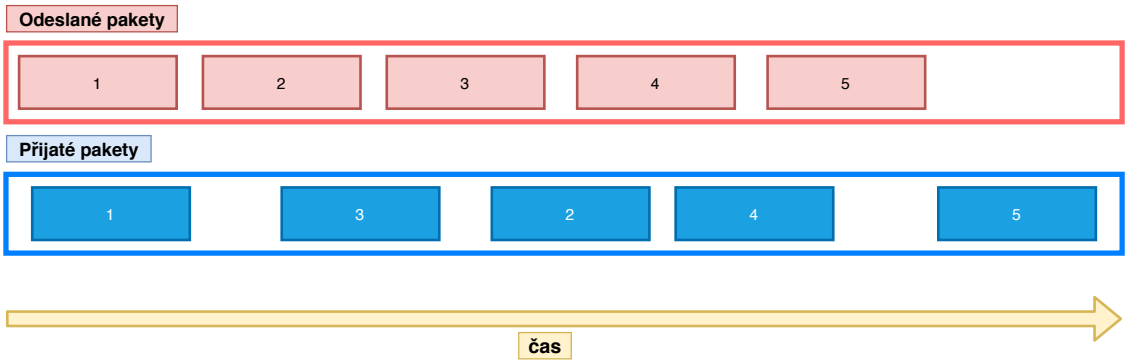
Jitter

Jitter[4] je kolísání zpoždění přijímaných paketů v paketově založených sítích. Na straně odesílatele jsou pakety odeslány s rovnoměrným rozestupem. Kvůli přetížení sítě, nevhodným frontám nebo nesprávné konfiguraci může tento konstantní rozestup kolísat. Zdroj [8] uvádí Jitter jako rozdíl mezi očekávaným přijetím paketu a reálným

[3] anglicky delay nebo latency

[4] v češtině je jitter zavedený pojem

přijetím paketu. Zdroj [9] uvádí Jitter jako standardní nebo maximální odchylku z průměrného zpoždění paketu.



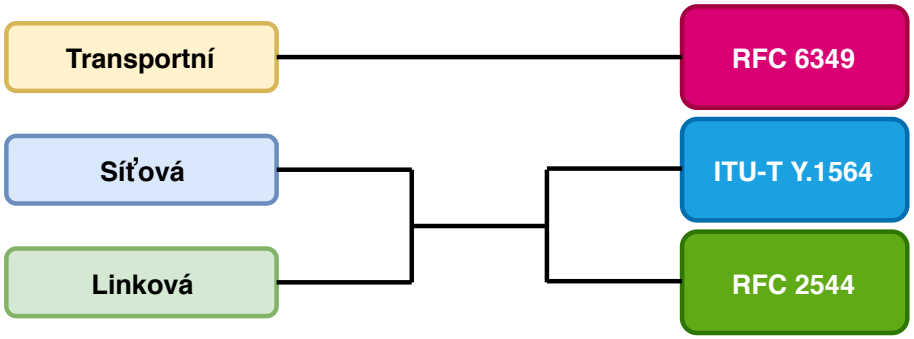
Obr. 1.1: Princip kolísání zpoždění [1]

Z důvodu nezaručeného pořadí fragmentů zprávy, jsou jednotlivé fragmenty ukládány do vyrovnávací paměti. Z paměti jsou uvolněny až v logickém celku. Tento jev může nejdříve způsobovat zpoždění a následně také ztrátu dat. Služby, které jsou závislé na přenášení dat v reálném čase jsou velmi citlivé na jitter. [8]

2 Metodiky měření přenosových parametrů datových sítí

Ať už je potřeba otestovat nový hardwarový prostředek, nový kód před uvedením do produkce nebo nalézt síťový problém, možnost generování síťového toku je nezbytností. Tato kapitola popisuje již standardizované metodiky měření a vysvětluje důležité faktory, na které musí brát síťový tester ohled. Základním faktorem při měření je vhodný výběr metodiky, sekvence a zařízení. Každá metodika je vhodná pro určitý typ testu, ale nic nebrání využití jejich kombinací. Při testování síťových parametrů je vhodné měřit obousměrný provoz, jinak mohou být výsledky zkresleny. [2]

V této kapitoly jsou shrnuty metodiky jako IETF RFC2544, ITU Y.1564 a RFC6349. Dále je vysvětlen rozdíl v postupu testování mezi jednotlivými metodikami. Obrázek 3.1 ukazuje princip testování podle orientace protokolů v ISO/OSI modelu. V poslední části je navrhnuta vlastní metodika měření.



Obr. 2.1: Metodiky z pohledu ISO/OSI modelu

2.1 IETF RFC2544

Metodika z roku 1999 definována v standardu IETF RFC2544 řešící testování jednotlivých síťových zařízení obsahuje několik testů pro rámce o standardní velikosti (64 B, 128 B, 256 B, 512 B, 1024 B, 1280 B a 1518 B). Nepodporuje měření více protokolů a služeb. Ideálním způsobem implementace testu je zapojení síťového testeru s vysílacími a přijímacími porty. Připojení jsou realizována z vysílače přes DUT[1]. [4]

Před testováním musí být nastavena uživatelská konfigurace, která by se neměla v průběhu testu měnit. Pokud existuje více konfigurací, je potřeba provést testy pro

[1] testované zařízení

každou z nich. Měření popsané v RFC2544 nebylo vytvořeno se záměrem použití mimo laboratoř. [4]

Výsledky testů pravděpodobně nepředstavují reálný výkon v síti kvůli způsobu jejich provedení a to z důvodu rozdílu ve vzorcích mezi hypotetickou topologií sítě a topologií o pevné velikosti rámců a zatížení, které jsou konstantní při měření v laboratoři, ale proměnné v produkční síti. Absence QoS, neznalost topologie při testu a jednoduchý síťový tok nejsou dostatečné pro celkové testování sítí v dnešní době [2].

2.1.1 Způsoby zapojení

Obrázek 6.2.7 ukazuje různé způsoby zapojení. V prvním případě je použit jeden síťový tester, jenž obsahuje vysílač i přijímač. Protože tester vysílá a zároveň přijímají síťový provoz, který prochází testovaným síťovým zařízením, lze jednoduše určit a ověřit, zda rámce byly v pořádku doručeny. Druhý způsob zapojení využívá stejný mechanismus pro testování s tím rozdílem, že jsou oddělena vysílací a přijímací zařízení. Pokud nejsou tyto zařízení vzdáleně kontrolována jedním strojem, který simuluje jednotný tester, mohou být některé testy (hlavně test propustnosti) omezené. Třetí způsob zapojení je jeden tester fungující jako vysílač i přijímač. V mnoha případech je tento test přesnější, protože může simulovat reálné fungování sítě.[4]

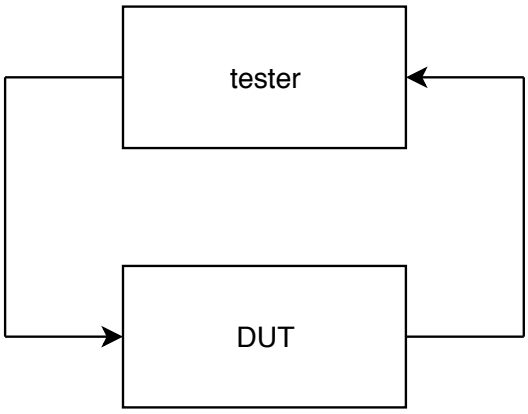
2.1.2 Měření propustnosti

Měření propustnosti probíhá cyklicky. Z vysílače jsou posílány rámce definovaného množství a určené rychlosti, které putují skrz testované zařízení. Pokud přijímač přijme všechny vyslané vzorky, zvýší se rychlost a množství pro další iteraci. Jakmile přijímač nepřijme všechny vyslané pakety, propustnost sítě je určena z předchozí iterace.[4]

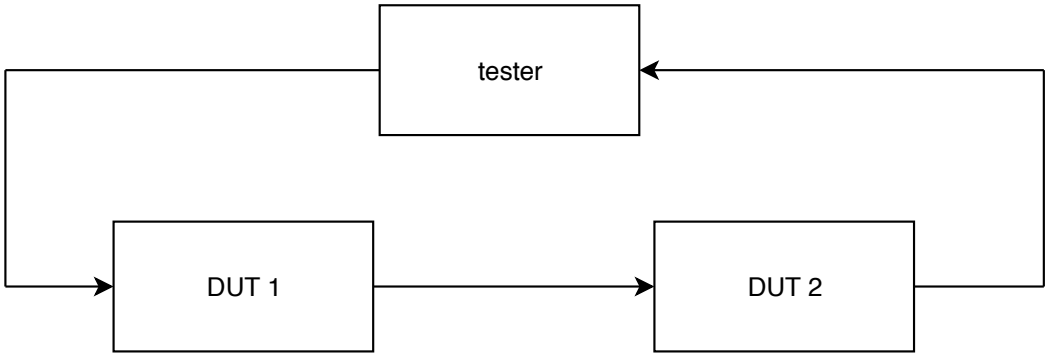
2.1.3 Měření ztrátovosti paketů

Test je založen na jednoduchém principu, kdy ztracené pakety jsou vypočítány jako rozdíl odeslaných paketů a přijatých paketů. Pro přehlednost je tento rozdíl převeden na procenta.

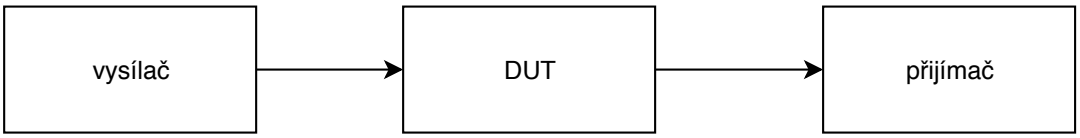
Dále se postup měření opakuje s tím rozdílem, že je snižována maximální rychlost přenosu o ne více než 10% dokud není změřena dvakrát po sobě nulová ztrátovost paketů. Před měřením ztrátovosti paketů by měl být proveden test propustnosti. V první iteraci by měla být použita maximální rychlost, která byla zjištěna v testu



Obr. 2.2: První způsob zapojení. Tester vysílá a zároveň přijímá rámce, které prochází DUT.[4]



Obr. 2.3: Druhý způsob zapojení. Jedno zařízení vysílá a druhé přijímá rámce, které prochází DUT.[4]



Obr. 2.4: Druhý způsob zapojení. Jedno zařízení vysílá a zároveň přijímá rámce, které prochází více DUT.[4]

propustnosti. Tato rychlost se v dalších iteracích zmenšuje, ale maximálně o 10 %. Test končí pokud dvě po sobě jdoucí iterace změří 0% ztrátovost.[4]

2.1.4 Měření zpoždění

Pro provedení tohoto testu je potřeba nejdříve provést test propustnosti pro každou velikost rámce DUT. Tok dat by měl trvat nejméně 120 s. Po dobu 60 s by měl být rámec identifikován razítkem. V okamžiku kdy je rámec plně odeslán, je stanovena časová známka odeslání. Přijímač rozezná identifikační rámec a zapíše časovou

známku přijetí. Měření by mělo proběhnout minimálně dvacetkrát. Odpovídající hodnota zpoždění by měla odpovídat průměru změřených vzorků. [4]

2.1.5 Test zatížitelnosti

Test se snaží zjistit maximální počet rámců poslaných s minimálním časovým intervalem povoleném na daném médiu. Test je prováděn cyklicky. První iterace začíná posláním shluku back-to-back rámců na DUT. Pokud je počet odeslaných rámců roven počtu přijatých, je v další iteraci počet rámců zvýšen. V případě, že počet přijatých rámců je menší než počet odeslaných rámců, je naopak počet odeslaných rámců zmenšen. Každá iterace tohoto testu by měla trvat nejméně dvě sekundy. Výsledek by měl být vypočítán jako průměr naměřených hodnot o minimálně 50 vzorcích. [4]

2.1.6 Test obnovy systému po přetížení

Cílem testu je změřit čas, který systém vyžaduje k navrácení se do funkčního stavu po přetížení. Prvním krokem testu je změření propustnosti. Následně je generován tok 110%ní rychlostí po minimálně jednu minutu. Časová známka A je zaznamenána ve chvíli, kdy je tok snížen na 50% a časová známka B je čas, kdy byl ztracen paket. Výsledný čas je rozdíl časové známky B a A.[4]

2.1.7 Test obnovy systému po restartu

Test měří čas, který zařízení potřebuje pro návrat do funkčního stavu po restartování zařízení. Test je prováděn vysláním toku rámců a v určitý moment je na zařízení proveden restart. Následně je změřen čas mezi přijetím posledního rámce před restartem a přijetím prvního rámce po restartu. [4]

2.2 ITU-T Y.1564

Doporučení ITU-T Y.1564 definuje metodiku testování, která může být použita při testování ethernetové sítě a jejích služeb. Toto doporučení s celým názvem Ethernet Service Activation Test methodology bylo vytvořeno s účelem nabídnout poskytovatelům standardizovaný způsob měření výkonu ethernetových služeb v jediném testu. Metoda měření se soustředí na parametry jako propustnost, zatížitelnost spoje, ztrátovost, zpoždění, dostupnost služby a dobu přepnutí na záložní zdroj. Parametry se

nazývají KPI (key parameter indicator). KPI parametry poskytují přehled o výkonosti spoje, které jsou využívané v SLA[2] smlouvách.

2.2.1 Profily datové propustnosti

Metro Ethernet Forum vydalo standard MEF 10.2 Phase 2 pro klasifikaci toku. Různé datové toky jsou rozděleny podle barev, které udávají prioritu toku. Metodika Y.1564 používá tento standard. Datový profil definuje následující čtyři parametry toku:

- **CIR** (committed information rate)
- **CBS** (committed burst size)
- **EIR** (excess information rate)
- **EBS** (excess burst size)

CIR definuje hranici garantovaného pásma pro vybranou službu. EIR udává hranici pásma, při které nejsou zaručeny kvalitativní parametry přenosu. CBS vyjadřuje maximální kapacitu datového toku podle SLA. EBS určuje kapacitu datového toku, při němž není zaručena SLA.

Pokud pakety vyhovují profilu definovanému v SLA, jsou obarveny zeleně. Zelené pakety nemohou být zahozeny a jsou pro ně garantované parametry KPI. Žlutě jsou obarveny ty pakety, které nejsou zahazovány, ale nelze pro ně garantovat doručení, v angličtine nazvané jako *best effort* pásmo. Červeně označené pakety nevyhovují datovému profilu a jsou zahazovány. [11] Rozdělení pásma je vyzobrazeno na obrázku 2.2.1.

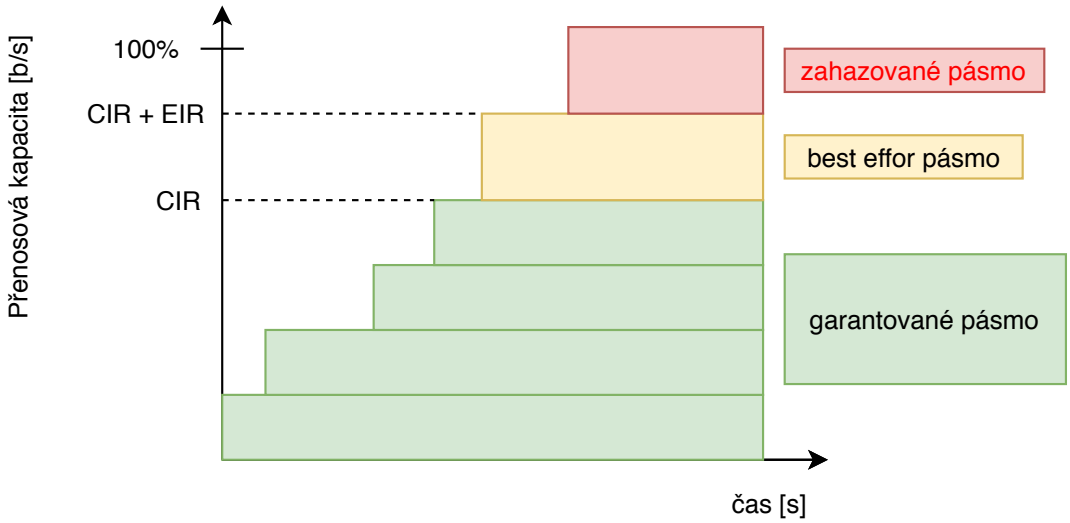
2.2.2 Metodika testování

Metodika si stanovuje dva cíle. Prvním z nichž je validace, že každá služba, jenž je založená na ethernetu, je správně nakonfigurovaná. Druhým cílem je ověření kvality služeb, které jsou doručeny zákazníkovi. Kompletní postup při testování je zobrazen na obrázku 2.2.2

V první fázi testu je zjišťovány hodnoty CIR a EIR pro danou síť. Tyto hodnoty jsou měřeny za plného provozu a musí být provedeny pro každou službu zvlášť. Test probíhá postupným navyšováním přenosové rychlosti po dobu šedesáti sekund. Při každém zvýšení jsou spočítány KPI parametry. Výsledky testu by měly být shrnuty do přehledné tabulky.

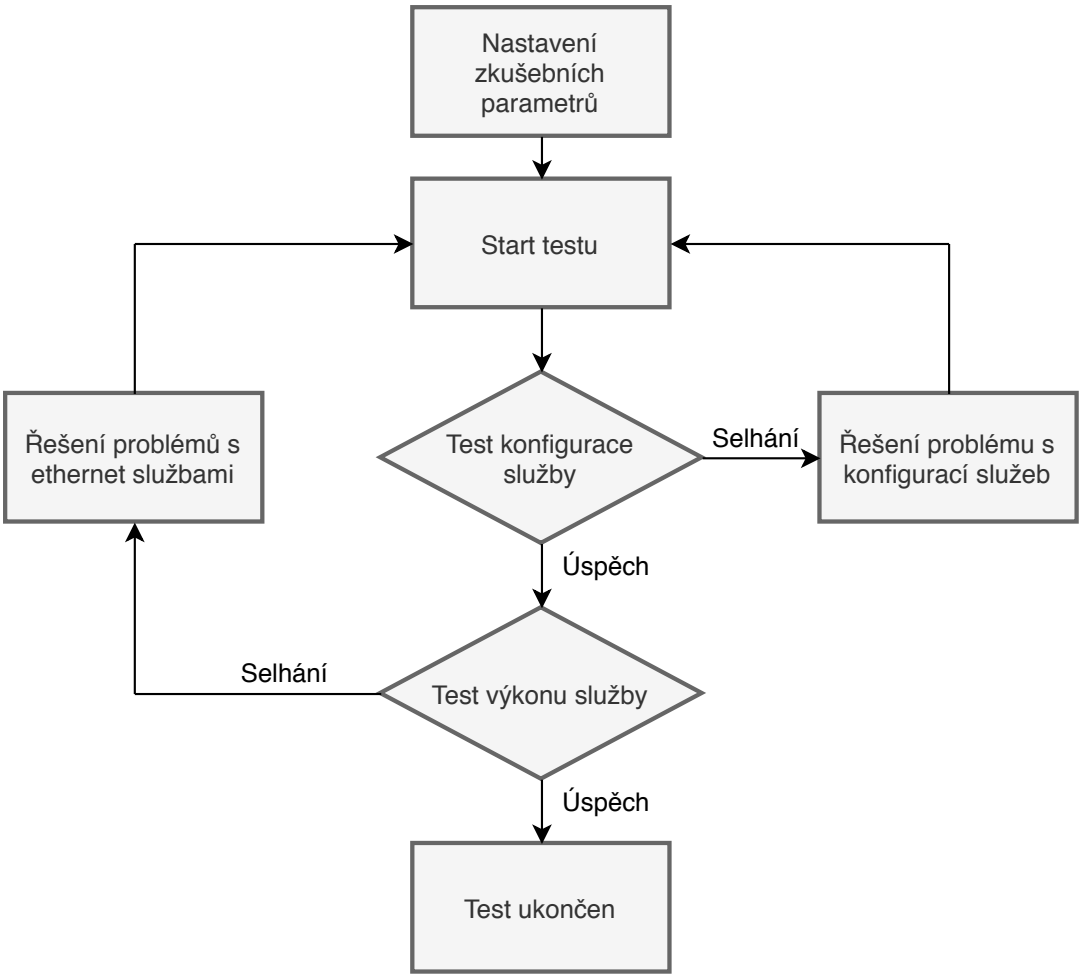
Druhá fáze testuje výkon služeb a jejím cílem je ověřit, zda nedochází k upřednostňování nebo omezení určitých datových toků. V této fázi se testují parametry

[2]SLA, celým názvem service level agreement, je smlouva definující jasná pravidla fungování a garance služby nebo produktu



Obr. 2.5: Rozdělení pásma ITU-T Y.1564 [11]

KPI pro jeden datový tok, který obsahuje všechny služby. Opět by výsledkem testů měla být přehledná tabulka s KPI parametry a velikostí rámce. [11]



Obr. 2.6: Stavový diagram metodiky ITU Y.1564 [11]

2.3 RFC6349

Z hlediska uživatelské spokojenosti nemusí být testování na linkové nebo síťové vrstvě dostačující [2]. Standard RFC6349, který je vhodný pro měření časově nezávislých služeb, popisuje metodiku měření datových parametrů na transportní vrstvě s využitím TCP protokolu. Výsledky testů tohoto frameworku se přibližují reálné spokojenosti koncového zákazníka. Nicméně testování na nižší vrstvě je potřebné k ověření integrity sítě. Bez ověření integrity sítě není možné provádět další testy. [6]

2.3.1 Metodika měření

Aby bylo měření smysluplné, je nejdříve nutné ověřit funkčnost sítě. RFC 6349 uvádí dva klíčové parametry: ztrátovost a zpoždění. Zpoždění 150 ms nebo 5% ztrátovost můžou být považovány za příliš vysoké pro přesné testování, avšak přesná hranice není uvedena. Testování probíhá podle následujícího pořadí:

1. Identifikace Path MTU z důvodu vyhnutí se fragmentaci paketů.
2. Měření obousměrného zpoždění a Bottleneck Bandwith. Tyto parametry jsou použity k nastavení TCP RWND[3] a velikosti Socket Buffer[4].
3. TCP testy propustnosti

Identifikace Path MTU

Identifikace Path MTU[5] by měla využít techniky Path MTU Discovery (PMTUD). PTMUD spoléhá na ICMP 'need to frag' zprávy ke zjištění Path MTU. Pokud zařízení odešle paket, který v sobě obsahuje atribut bez fragmentace (Don't Fragment) a paket je zároveň větší než MTU dalšího skoku, je paket zahozen. Zařízení zahazující paket v takovémto případě posílá autorovi paketu ICMP 'Need to frag' zprávu. Zpráva obsahuje MTU dalšího skoku. Tuto informaci PTMUD využívá k přizpůsobení velikosti odesílaného paketu. Z důvodu časté blokace ICMP protokolu na straně poskytovatelů může být tato metoda identifikace nespolehlivá. Fragmentace v průběhu testu může silně ovlivnit výsledky celého testování. [6]

Zjištění Bottleneck Bandwidth

Před testováním TCP propustnosti je nutné změřit Bottleneck Bandwidth[6]. Testování by mělo probíhat obousměrně, hlavně v sítích s technologií ADSL. Měření Bottleneck Bandwith lze pomocí standardu RFC5136 nebo RFC2544. I navzdory faktu, že RFC 2544 bylo vytvořeno pro laboratorní měření, v praxi se tyto výsledky používají pro určení šířky pásma. RFC5136 je orientováno na měření v reálných podmínkách, avšak ve standardu není definován jasný postup, ale pouze matematické výpočty a pojmy. [6]

Měření obousměrného zpoždění

Měření RTT by mělo probíhat mimo dobu, ve které je velmi vytížená síť. V opačném případě může být výsledek testu zkreslen vyrovnávací pamětí na spoji. V měřícím intervalu by mělo být změřeno více vzorků. Následně by měl být vybrán vzorek s nejmenší hodnotou. Měření RTT může probíhat následujícími způsoby:

- Využití síťových testerů na každém konci sítě takovým způsobem, že je možné změřit tok paketů z jednoho konce na druhý.

[3] TCP Receive Window (rwnd)
[4] paměť pro otevřené sockety
[5] česky maximální přenosová jednotka cesty
[6] česky úzké hrdlo

- Zachycení testovacích TCP relací a následná analýza. Pro určení RTT by neměl být použit three-way-handshake z důvodu možného zpoždění kontroly firewallu.
- Získání RTT z MIB (SNMP Protokol) statistik
- ICMP Ping. [6]

Konfigurace parametrů

Před testováním TCP propustnosti je nutné změřit oboustranné zpoždění a Bottleneck Bandwith. Tyto hodnoty jsou použity pro výpočet parametrů TCP RWND a Socket Buffer odesílatele. Pro optimální výsledky je potřeba nastavit velikost Socket Buffer na obou uzlech stejné. Testované zařízení musí povolit nastavení Send Socket Buffer a velikosti Receive Window. Při menším nastavení než BDP bude výkon TCP limitován. Výpočet BDP je definován vzorcem 2.3.1, kde RTT je oboustranné zpoždění a BB je bottleneck bandwith. [6]

$$bdp\ [b] = rtt\ [s]\ *\ bb\ [b/s] \tag{2.1}$$

Pro výpočet minimální velikosti TCP okna je použit vzorec:

$$tcp\ rwnd\ [B] = \frac{bdp\ [b]}{8} \tag{2.2}$$

V případě konfigurace BS a TCP RWND na vysokou hodnotu může u sítí s nízkým Bottleneck Bandwith dojít k přetížení vyrovnávací paměti síťového prvku, jehož směrem tester vygeneruje velké množství segmentů. Toto množství nezvládne síťový prvek zpracovat, a proto dojde ke zbytečnému zahazování paketů vlivem velikosti Socket Buffer. [6]

Měření TCP propustnosti

Před samotným spuštěním je doporučeno provést následující kroky:

- Ověření nastavených parametrů např. pomocí programu Wireshark.
- Je doporučeno vyzkoušet měření vůči více měřícím serverům z důvodu komparace a zjištění, zda nedochází k prioritizaci na základě IP adresy.
- Ověření síťové neutrality, tj. ověření zda nedochází k prioritizaci služeb. [6]

$$tcp\ propustnost\ [b/s] = tcp\ rwnd\ [b] \cdot\ 8rtt[s] \tag{2.3}$$

2.3.2 Výpočet TCP metrik

RFC6349 definuje tři metriky dovolující komparaci výsledků v různých podmínkách sítě a při různém nastavení měřících stran. Z tohoto důvodu by měly být všechny metriky měřeny v každém testu. Účelem těchto metrik je lepší pochopení a interpretace získaných výsledků. [6]

Transfer Time Ratio

První metrika udává poměr mezi reálným časem a teoretickým (ideálním) časem přenosu. Reálný čas přenosu je definován jako čas strávený přenosem bloku dat. Ideální čas je odhad, jak dlouho by měl přenos trvat s ohledem na BB testované sítě. Je definován vzorec 2.3.2, kde TTa je reálný čas přenosu a TTi teoretický čas přenosu. [6]

$$TTR\ [-] = \frac{TTa\ [s]}{TTi\ [s]} \tag{2.4}$$

TCP efektivita

Druhá metrika definuje procento dat, které nemuselo být přeposláno. Metrika je definována vzorcem 2.3.2, kde TB je celkový počet přenesených bajtů a TB je počet bajtů, který musel být přeposlán. [6]

$$TE\ [\%] = \frac{TB\ [B] - RB\ [B]}{TB\ [B]} \cdot 100\ [\%] \tag{2.5}$$

Buffer Delay

Tato metrika definuje vztah mezi nárůstem RTT během průtoku dat a ideálním RTT. Ideální RTT je ideální zpoždění sítě, tj. bez zahlcení sítě. Průměrné zpoždění $ARTT$ je definováno vzorcem 2.3.2, kde $TRTT$ vyjadřuje celkové zpoždění během přenosu a TD celkové trvání přenosu. [6]

$$ARTT\ [s] = \frac{TRTT\ [s]}{TD\ [s]}\ [s] \tag{2.6}$$

Výpočet Buffer Delay BD je definován vzorcem 2.3.2, kde $BRTT$ je ideální zpoždění.

$$BD\ [\%] = \frac{ARTT\ [s] - BRTT\ [s]}{BRTT\ [s]} \cdot 100\ [\%] \tag{2.7}$$

2.4 Návrh metodiky měření

Návrh metodiky vychází ze standardu RFC6349 a je doplněna o doporučení z metodického postupu Českého telekomunikačního úřadu pro měření datových parametrů sítí pomocí TCP protokolu, které mimo RFC 6349 vychází i z jiných standardů.

Před testováním je nutné ověřit funkčnost sítě. Jak již bylo zmíněno v předešlé kapitole, RFC6349 uvádí 5% ztrátovost nebo 150ms zpoždění jako vyšší hodnoty, které mohou být brány jako mimořádný stav sítě a výsledky mohou být zkresleny. Test by také neměl být prováděn pokud hodnoty zpoždění a ztrátovosti kolísají v čase. Test by měl brát v potaz Traffic Shaping[7] a Traffic Poling[8]. Test by měl být vyzkoušen na různých portech (službách), aby se ověřilo, zda nejsou nějaké služby upřednostňovány. Fragmentace ovlivňuje výsledky testů. Pro měření TCP bude metodika využívat doporučení z RFC6349. Viz 2.3.1.

Sekvence měření

Z důvodu agregace internetového připojení, které může způsobovat výkonnostní výkyvy, by mělo měření probíhat minimálně ve dvou časových interval. Jedno měření by mělo probíhat v datové špičce a druhé měření by naopak mělo probíhat za co nejmenšího provozu. [12]

IPv4 a IPv6

Protože protokoly transportní vrstvy nemusí být za použití současných technologií zapouzdřeny pouze do IPv4, ale i do IPv6, musí být testy provedeny pro všechny dostupné protokoly.[12]

Fyzické a technologické parametry měření

Testování bude probíhat podle architektury klient-server. Serverová část by měla být umístěna na páteřním uzlu datového připojení tak, aby nezkreslovala výsledky

[7] Traffic Shaping je forma omezování rychlosti sítě

[8] Traffic Poling forma omezování provozu

měření. Na klientské straně by měl být zhodnocen hardware a lokální síť. Testování by mělo probíhat bez dalších uživatelů lokální sítě.[12]

Počet spojení

V případě vysoké hodnoty BDP zmíněné v kapitole 2.3.1 by mělo být měření rozděleno do více TCP spojení a to z důvodu, aby došlo k věrohodnému pokrytí celého pásma. Počet spojení je definován vzorcem 2.4. V tabulce 2.4 jsou doporučené hodnoty pro počet spojení, který musí být uzavřeny pro každou sekvenci měření.[6]

$$N\ [-] = \frac{BDP\ [b]}{RWND\ [B]} \tag{2.8}$$

TCP RWND	Počet spojení
16 kB	20
32 kB	10
64 kB	5
128 kB	3

2.4.1 Průběh měření

Cílem měření je zjistit datové parametry pro různé aplikační služby. Pro testování by měl být použit nástroj, který umožní generovat TCP i UDP datový tok. Pro časově nezávislé služby, které obecně využívají protokol TCP by měl umožnit nastavení parametrů TCP RNWD a Socket Buffer. Pro služby závislé na reálném čase by měl nástroj podporovat multicast. Postup by měl být podle následujícího pořadí:

- **Analýza sítě**
  Před samotným měřením musí být zkontrolován stav sítě a zjištěno, zda jsou podmínky v internetovém připojení v normě. Pro analýzu můžou být využity nástroje jako ping, Wireshark, iPerf, apod.
- **Měření RTT a Bottleneck Bandwith**
  Před samotným meřením musí být změřeno obousměrné zpoždění a bottleneck bandwith. Tyto parametry slouží k následné konfiguraci, aby TCP protokol dosahoval co nejlepších výsledků.
- **Konfigurace TCP RWND a socket buffer**
  Konfigurace musí být provedena na straně klienta i serveru.

- **Nastavení emulačních parametrů v internetovém připojení**
  V případě emulace zhoršených podmínek v síti musí být provedeno nastavení. Po dokončení nastavení, musí být ověřeno, že nastavení bylo v pořádku provedeno.
- **Měření datových parametrů**
  Měření datových parametrů pomocí nástrojů iPerf a tPerf, které musí být zaznamenány do přehledné tabulky.
- Analýza a zpracování výsledků s ohledem na další možné faktory ovlivňující měření.

2.4.2 Vyhodnocení výsledků

Výsledkem měření by mělo být minimálně 10 hodnot pro tyto kategorie:

- Tok generovaný klientem ve špičce
- Tok generovaný na straně serveru ve špičce
- Tok generovaný klientem mimo špičce
- Tok generovaný na straně serveru mimo špičce

Tyto hodnoty by měly být zpracovány do přehledné tabulky pro každou výše zmíněnou kategorii a službu. Jednotlivé výsledky by měly také obsahovat veškeré nastavené parametry. Měly by být změřeny nejvíce používané a veřejné známé služby.

2.5 Návrh metodiky měření pomocí nástroje Lighthouse

Měření pomocí nástroje Ligthouse bude probíhat následujícím způsobem:

- **Analýza sítě**
  Před samotným měřením musí být zkontrolován stav sítě a zjištěno, zda jsou podmínky v internetovém připojení v normě. Pro analýzu můžou být využity nástroje jako ping, Wireshark, iperf, apod.
- **Nastavení emulačních parametrů v internetovém připojení**
  V případě emulace zhoršených podmínek v síti musí být provedeno nastavení. Po dokončení nastavení, musí být ověřeno, že nastavení bylo v pořádku provedeno.
- **Měření pomocí Lighthouse**
  Měření pomocí nástroje Ligthouse a uložení výsledků.
- **Analýza a zpracování výsledků**
  Výsledek musí být mediánem z 21 změřených hodnot a vyhodnocení musí brát ohled na další možné faktory ovlivňující výsledky.

3 Koncepce testeru

Jak již bylo zmíněno v úvodu, tester je implementován v programovacím jazyce Java pro aplikaci JMeter. Pro měření je využita architektura klient-server. Pro měření propustnosti jsou zvoleny nástroje iPerf3 a tPerf, pro které jsou vytvořena grafická rozhraní umožňující konfiguraci.

3.1 Aplikace JMeter

Aplikace JMeter, spravována organizací Apache, je open-source software navržený především pro výkonnostní testy. JMeter původně testoval webové aplikace, ale během svého vývoje se rozšířil i na další klient-server aplikace (protokoly). Pomocí výše zmíněného nástroje je v současné době možné testovat webové protokoly HTTP/HTTPS, SOAP/Rest služby, FTP, Database, LDAP, e-mailové protokoly SMTP(S), POP3(S) a IMAP(S), TCP, Java objekty a nebo spouštět nativní příkazy a shell skripty. JMeter je implementován v programovacím jazyce Java, díky kterému je možné jej označovat jako multiplatformní. Aplikace také poskytuje konzolové i grafické rozhraní pro spouštění testů. V grafickém rozhraní lze jednoduchým způsobem definovat průběh testů. [13]

3.1.1 Testovací plán

Průběh testu je definován v testovacím plánu, který může být uložen ve formátu jmx pro pozdější úpravy nebo spuštění. Test se může skládat z těchto základních prvků:

- **Thread group** - Jedná se o prvek testovacího plánu, ve kterém běží paralelně zpracované vlákno. Jedno vlákno odpovídá jednomu uživateli. V tomto prvku lze také nastavit délku trvání a počet opakování celého vlákna.
- **Sampler** – Slouží ke konstrukci a konfiguraci konkrétních požadavků. Konfigurace požadavků může být rozšířena pomocí ovládacího prvku *Configuration Element*.
- **Logic controller** – poskytuje možnost ovlivnit pořadí požadavků
- **Config element** – definuje proměnné, které jsou sdíleny pro všechny požadavky. Je vhodné do nich ukládat například autentizační údaje, protože jakákoliv změna se provede na jednom místě.
- **Timer** – umožnuje testu pracovat s časem, například zastavení nebo zvýšení intenzity v určitý čas. Časovač je vhodné vkládat mezi dva požadavky běžící v jednom vlákně.
- **Pre processor** – poskytuje možnost k úpravě sampler před jeho spuštěním v dané části testu.

- **Post processor** – umožňuje pracovat s odpovědmi požadavků. Jeho použití může být např. uložení requestu, apod.
- **Assertions** – slouží k validaci odpovědí. JMeter nemusí fungovat jen jako zátěžový test, ale díky assercím může fungovat i jako jednoduchý tester.
- **Listener** – poskytuje rozhraní pro zobrazení výsledků v podobě grafů, tabulek, apod.[13]

Pro každý prvek v seznamu je možné rozšířit nebo upravit chování. Z hlediska kódu se jedná o třídy, které je možné přetěžovat.

3.1.2 Metriky JMeteru

- Elapsed time – vyjadřuje délku trvání jednoho požadavku
- Latency – vyjadřuje zpoždění při vykonávání požadavku
- Connect time – délka připojení
- Median – prostřední čas v řadě výsledků
- 90% percentil – 90 % všech výsledků trvalo déle než tento čas
- Throughput – počet požadavků za celkový čas
- Max – nejdelší čas jednoho požadavku
- Min – nejkratší čas jednoho požadavku[13]

3.2 Možné měřicí nástroje

Pro měření síťových parametrů existuje několik programů. Autor se při výběru zaměřil na obě verze nástroje iPerf a nuttcp. Programy se liší svými funkcemi, které jsou srovnány v tabulce 3.1.

3.2.1 iPerf

iPerf[1] je nástroj pro měření propustnosti, ztrátovosti, zpoždění a dalších parametrů pomocí transportních protokolů TCP nebo UDP. Využívá architekturu klient-server. Nástroj umožňuje nastavovat jak velikost Socket Buffer, tak i velikost TCP RNWD. Tyto dva parametry je možné konfigurovat na straně klienta i serveru. Dosažitelná propustnost může být změřena i obousměrně. [14]

iPerf existuje ve dvou major verzích, konkrétně verze 2.* (iperf) a 3.* (iperf3). Obě verze jsou samostatně vyvíjeny jinými týmy a jedná se o různé programy. Liší se svými funkcemi, které jsou srovnány v tabulce 3.1.

[1] více informací o nástroji a stažení nástroje na https://iperf.fr/

3.2.2 nuttcp

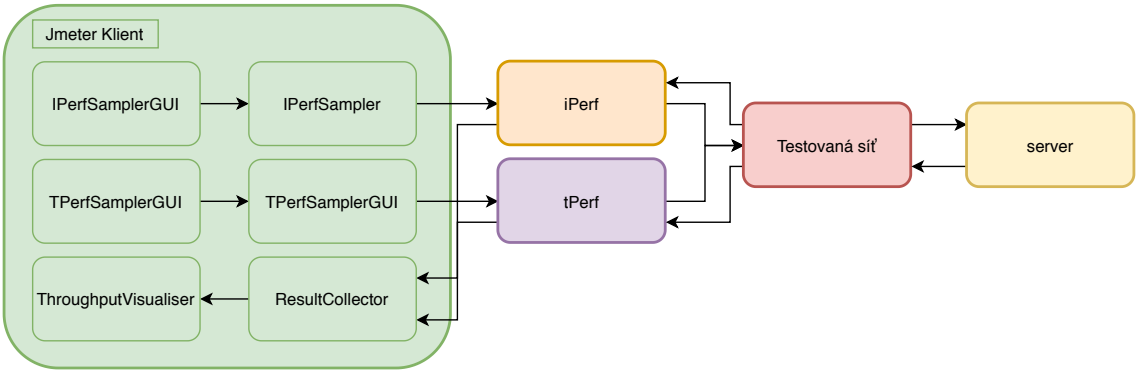
nuttcp je síťový nástroj pro měření výkonnosti. Měří hrubý TCP nebo UDP propustnost na síťové vrstvě zasíláním dat ze zdrojového systému přes připojenou síť do cílového systému. Dokáže posílat určité množství dat nebo data po určitý čas. Nuttcp poskytuje nejen hodnotu propustnosti v Mb/s, ale také další informace spojené s přenosem dat, jako uživatel, systém, čas, využití procesoru odesílatele a přijímače a v případě UDP ještě ztrátovost paketů.[15]

Vlastnost	**iperf**	**iperf3**	**nuttcp**
Podpora OS Windows	ano	ne	ne
JSON výstup	-	–json	-
Podpora multicast	–ttl	-	-m
Obousměrný test	–dualtest	–bidir	-
hlášení CWND a přeposílání	–e	v základu	-br / -bc
vyhnutí se TCP slowstart	-	–omit	-
nastavení algoritmu řízení toku	-Z	–congestion	-

Tab. 3.1: Rozdílné vlastnosti programů pro měření síťových parametrů..

3.3 Struktura testeru

Pro rozšíření je vytvořen sampler s grafickým rozhraním pro nastavení klienta i serveru. Sampler podle typu spouští měřící nástroj iPerf nebo tPerf. Konfigurace spuštění nástroje iPerf a tPerf je možná pomocí grafického rozhraní, ve kterém je možné zadat parametry, a to jak pro klienta, tak pro server. Měřící nástroj dále provádí měření vůči testovacímu serveru skrze testovanou síť. Výsledky testu se zobrazí v grafickém rozhraní aplikace JMeter.



Obr. 3.1: Princip fungování síťového testeru. Skrz grafické rozhraní jsou nastaveny parametry pro nástroj iPerf. Po spuštění testu v aplikaci JMeter je nástroj spuštěn. Výsledky měření se dají zobrazit v aplikaci.

4 QUIC

QUIC[1] je nový přenosový a šifrovaný transportní protokol postavený nad UDP, který je navržen tak, aby poskytoval bezpečné a spolehlivé připojení s cílem snížit zpoždění v porovnání s TCP. QUIC klient je dostupný v Google Chrome od verze 28 a výš a je možné jej využít pro komunikaci se servery, které tento protokol podporují. QUIC kombinuje funkce HTTP2 protokolu - multiplexování a řízení toku a TLS – bezpečnost a TCP – řízení toku, spolehlivost a uzavření spojení. Hlavní myšlenkou nového protokolu je využít UDP jako transportního protokolu a implementovat nové funkce v aplikační části.

TCP se ukázalo jako velmi spolehlivý protokol pro HTTP. TLS protokol naplnil požadavky pro bezpečný přenos dat. Kombinace těchto dvou protokolů se stala nezbytnou pro spolehlivý a bezpečný přenos dat. TCP třícestný handshake zvyšuje latenci sestavení připojení, která je znatelná v aplikacích závislých na čase. V HTTP/1.1 je vytvářeno spojení pro každou URL. Protože protokol TCP není multiplexovaný, vytváří více spojení mezi serverem a klientem. Vytváření spojení vede k využívání více soketů a často dochází k přenášení redundantních informací. Protokol HTTP/2 přes TCP má problém s HOL blokací, která díky řízení toku způsobuje snížení propustnosti. Všechny tyto důvody vedly k tomu, aby vznikl nový transportní protokol – QUIC.

V této kapitole jsou popsány cíle návrhu protokolu, hlavní funkce a vlastnosti, hlavičky a typy paketů. Kapitola je zakončena popisem připojení, řízení datového toku a funkce multiplexování.

4.1 Cíle návrhu protokolu

Při návrhu protokolu byly definovány klíčové cíle:

- Minimalizace času potřebného k sestavení připojení
- Redukce HOL
- Možnost rychlého a rozsáhlého uvedení do produkce bez nutnosti upravovat stávající infrastrukturu a operační systémy
- Redukce zpoždění způsobeného přeposíláním zpráv, které je zapříčiněno ztrátou paketů
- Detekce přetížení
- Bezpečnost ekvivalentní s TLS
- Spolehlivé a bezpečné sdílení prostředků jak na straně klienta, tak na straně serveru [18]

[1] Quick UDP Internet Connections

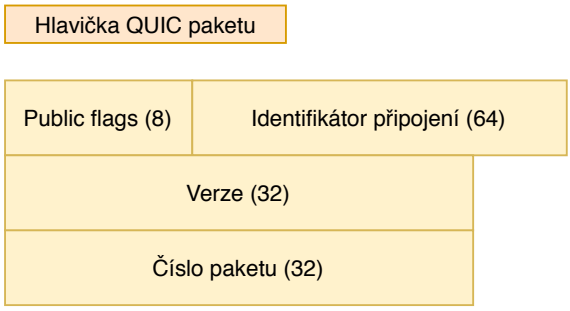
4.2 Hlavní funkce a vlastnosti protokolu

Hlavními funkcemi a vlastnostmi protokolu jsou:

- **Snížení odezvy při sestavování připojení**
  QUIC kombinuje transportní a šifrovací handshake, s cílem snížit obousměrné zpoždění. Pro známé servery QUIC využívá 0-RTT, které dovoluje odeslat data bez nutnosti čekat na odpověď serveru.
- **Migrace připojení**
  QUIC připojení používá 64 bitový identifikátor na aplikační vrstvě. TCP využívá čtveřici: zdrojová IP adresa a port, cílová IP adresa a port. Z tohoto důvodu může připojení migrovat i přes změnu IP adresy nebo portu. Při migraci klient používá stejný kryptografický klíč v další komunikaci.
- **Multiplexování**
  QUIC využívá více toků pro přenost dat. Tyto toky jsou na sobě nezávislé a jsou identifikovatelné podle jejich *Stream ID*. Data ve streamech jsou posílány v rámcích. Ztráta jednoho rámce ovlivní pouze jeden tok [18].

4.3 Hlavička QUIC paketu

Všechny QUIC pakety na lince začínají hlavičkou zobrazenou na 4.1. Veřejné značky hlavičky obsahují následující informace:



Obr. 4.1: Hlavička QUIC paketu.

- **0x01 = PUBLIC_FLAG_VERSION** – interpretace tohoto příznaku záleží na tom, jestli je paket odeslán ze strany serveru nebo klienta. Pokud je paket poslán od klienta, nastavení indikuje, že paket obsahuje verzi QUIC. Tento bit musí být nastaven ve všech paketech až do potvrzení serveru. Server verzi QUIC potvrdí tím, že v odpovědi vynechá nastavení tohoto bitu. Pokud server v odpovědi nastaví tento bit, dochází k vyjednávání verze.
- **0x02 = PUBLIC_RESET_FLAG** – nastavení indikuje, zda je paket typu *Public Reset*

- **0x0C dva bity indikují velikost identifikátoru připojení:**
  - 0x0C definuje 8 bajtový identifikátor připojení
  - 0x08 definuje 4 bajtový identifikátor připojení
  - 0x04 definuje 1 bajtový identifikátor připojení
  - 0x00 indikuje, že identifikátor připojení je vynechán
- **0x30 dva bity indikují velikost čísla paketu:**
  - 0x30 definuje 6 bajtové číslo paketu
  - 0x30 definuje 4 bajtové číslo paketu
  - 0x30 definuje 2 bajtové číslo paketu
  - 0x30 definuje 1 bajtové číslo paketu
- **0x40** je rezervováno pro vícecestné použití
- **0x80** je momentálně nepoužívané, musí být nastaveno na 0 [18]

4.4 Typy QUIC paketu

Všechny QUIC pakety začínají hlavičkou, která je doplněna o políčka podle typu paketu. Typ paketu je určen podle příznaků v *Public Flags*. QUIC pakety mohou být rozděleny do dvou kategorií, kterými jsou speciální pakety a běžné pakety.

Speciální pakety

- **Version Negotiation Packet** – tento paket je poslán serverem. Začíná políčkem public flags, který je následován identifikátorem připojení. Zbytek dat obsahuje *version negotiation paket* s 32 bitovým listem verzí, který server podporuje.
- **Public Reset Packet** – tento paket uačíná políčkem Public Flags, který je následován identifikátorem připojení. Zbytek je *Version Negotiation Paket* je kroptografický handshake se značkami [17].

Běžné pakety

- **Forward Error Correction Paket** – Bežný paket s nastaveným příznakem FEC
- **Frame Paket** – obsahuje aplikační data. [17]

4.5 Typy rámců

Všechny aplikační data jsou zapouzdřeny do rámců v QUIC paketech. Rámcové pakety jsou identifikované typovým polem, kterému předchází hlavička podle nasta-

veného typu.

Speciální typ rámce	**Běžný typ rámce**
STREAM	PADDING
ACK	RST_STREAM
CONGESTION_FEEDBACK	CONNECTION_CLOSE
	GOAWAY
	WINDOW_UPDATE
	BLOCKED
	STOP_WAITING
	PING

Tab. 4.1: Tabulka typu rámců. [18]

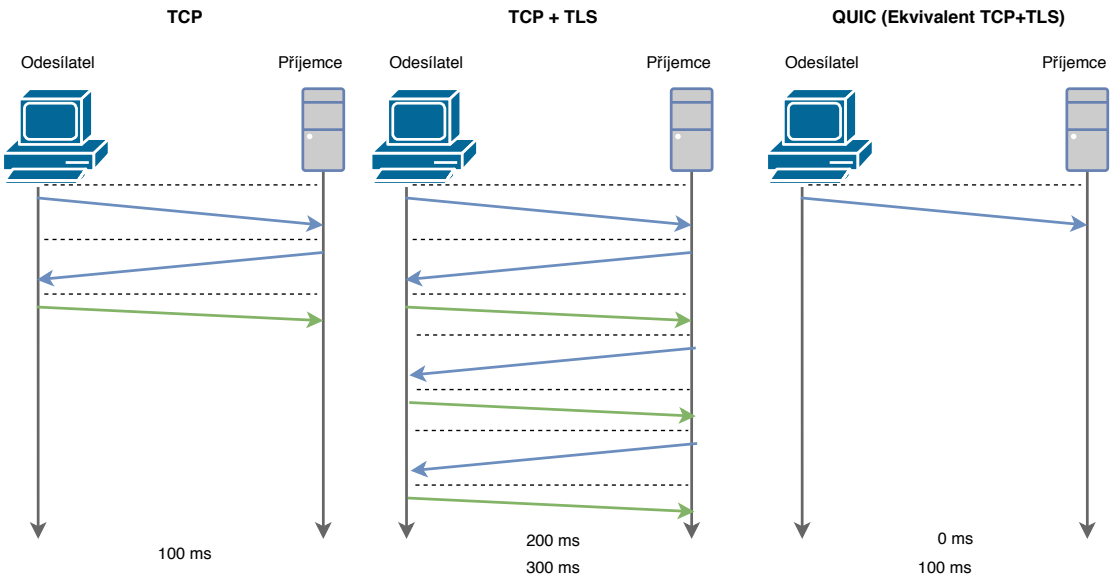
- **STREAM**
  STREAM je použit pro inicializaci nového toku nebo k poslání dat do existujícího toku. Rámec obsahuje 8-bitové pole pro *STREAM ID.*
- **ACK**
  Jedná se o typ rámce, který má za úkol indikovat, zda byl paket doručen nebo je postrádán. Rámec obsahuje pole potvrzení od 1 do 256.
- **WINDOW_UPDATE**
  Tento rámec je použit pro indikaci druhému uzlu, že byla zvýšena velikost okna.
- **STOP_WAITING**
  STOP_WAITING indikuje, že uzel, který přijímá pakety, již dále nečeká na pakety nižší než specifikovaná hodnota.
- **BLOCKED**
  BLOCKED je rámec, který se používá pro zjišťování chyb. odesílatel toho rámce indikuje druhé straně, že je připraven na odeslání dat, ale je zablokován řízením toku.
- **RST_STREAM**
  Rámec typu RST_STREAM FRAME slouží k ukončení toku odesílatelem i příjemcem.
- **CONGESTION_FEEDBACK**
  Experimentální rámec, který se momentálně nepoužívá.
- **PING**
  Rámec, který se používá se pro zjištění, zda je příjemce stále aktivní. Příjemce na tento paket musí odpovědět.
- **CONNECTION_CLOSE**

Tento rámec je vhodný pro běžné uzavření spojení a jednotlivé toky jsou uzavřeny se spojením.

- **GOAWAY**
  Rámec indikuje, že uzel bude uzavírat spojení. V ideálním případě by měl být odeslán chvíli před odesláním CONNECTION_CLOSE [17].

4.6 Připojení

Sestavení připojení začíná kombinovanou kryptografickou a transportní handshake fází, během které klient a server sestaví sdílený klíč pomocí jejich kryptografického protokolu (QUIC-TLS) a domluví se na aplikačním protokolu. Tento handshake potvrzuje, že obě strany chtějí komunikovat a sestaví parametry pro komunikaci. QUIC může omezeným způsobem fungovat i bez uzavření spojení. 0-RTT umožňuje klientovi odesílat zprávy bez přijetí potvrzení spojení ze serveru a naopak. 0-RTT spojení není bezproblémové a přináší rizika, o kterých by uživatelé měli vědět před použitím. Data odeslána při 0-RTT mohou být kompromitována, protože není předem provedena výměna kryptografických klíčů. Po provedení 0-RTT handshake jsou data v bezpečí, protože dojde k výměně klíčů. [17]



Obr. 4.2: Sestavení připojení QUIC, TCP a TCP + TLS. V případě QUIC a TCP+TLS sestavování připojení s již známým serverem je čas potřebný úspěšnému sestavení menší.

Připojení mezi klientem a serverem je stavová interakce za účelem výměny dat mezi aplikacemi. Sestavené připojení má svůj identifikátor CID pouze na aplikační

vrstvě a může migrovat na jinou IP adresu nebo port. QUIC by díky identifikátorům na aplikační vrstvě mohl podporovat vícecestné spojení. Odesílatel nebo přijímač mohou spojení libovolně ukončit. Ukončení spojení může nastat v následujících scénářích:

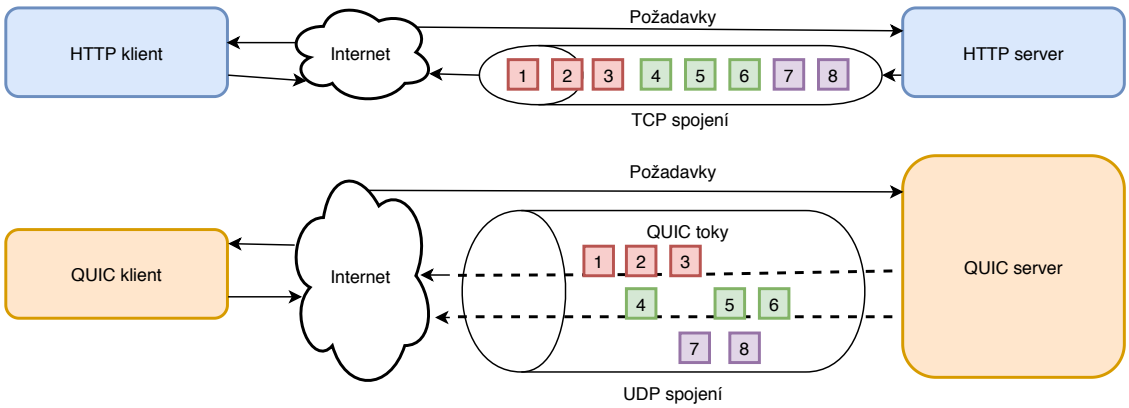
- **Vynucené ukončení**
  jeden z uzlů pošle *CONNECTION_CLOSE* rámec druhému uzlu. Tento rámec indikuje uzavření spojení. Uzel může odeslat *GOAWAY* rámec před odesláním *CONNECTION_CLOSE*, které indikuje, že uzel nebude vytvářet ani přijímat nové toky v tomto spojení.
- **Vypršení spojení**
  základní čas spojení je nastaven na 30 sekund, během handshake fáze může být tato hodnota nastavena až na 10 minut. Po uplynutí nečinnosti je odeslán rámec *CONNECTION_CLOSE* a spojení je ukončeno. [17]

4.7 Multiplexování

Stream je v protokolu QUIC jednoduchý a seřazený tok bytů poskytující aplikacím abstrakci. Toky mohou být jednosměrné nebo obousměrné. Toky mohou být tvořeny daty, ale také procesy, které pomáhají řídit tok. Toky jsou navrženy tak, aby vytvářeli minimální režijní náklady. Tato abstrakce může přenášet data po celou dobu připojení. QUIC multiplexing byl navržen tak, aby poskytoval 1) prioritizaci toku 2) sjednocení toku sdílené přes jedno UDP spojení 3) komprese http hlavičky, která je sdílená přes jedno připojení [17].

Používáním UDP je QUIC schopný eleminovat HOL[2]. Například pokud je ma obrázku 6.1 ztracen paket číslo jedna, ale ostatní jsou doručeny, všechny pakety musí čekat, než je ztracený paket doručen, aby byly doručeny aplikaci. Při použití UDP mohou být všechny pakety doručeny aplikaci bez čekání na přeposlání jediného ztraceného paketu [17].

[2]headline of blocking



Obr. 4.3: Porovnání HTTP a QUIC architektury.

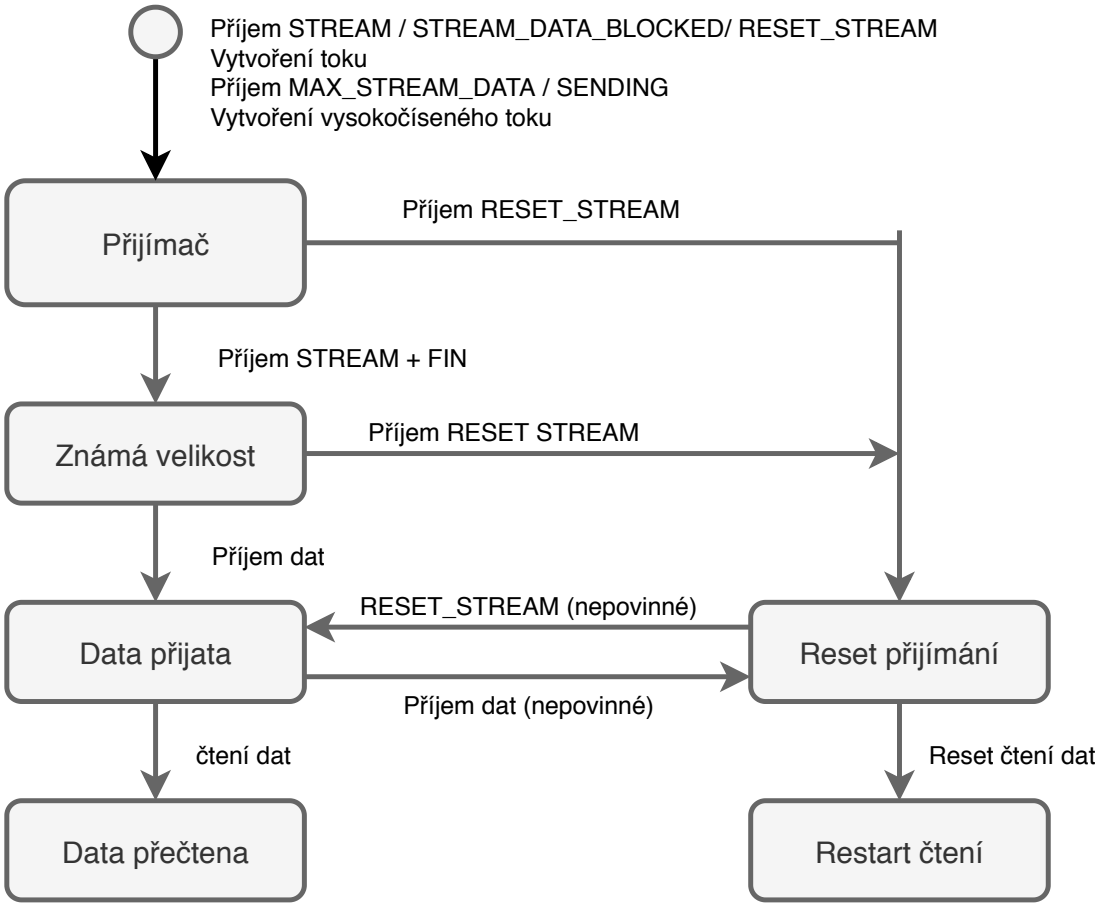
4.8 Řízení datového toku

QUIC používá schéma řízení toku podobné schématu v HTTP/2, kde příjemce inzeruje počet bytů, které je připraven přijímat v daném proudu a pro celé připojení. To vede k následujícím typům řízení toku:

- **řízení toku proudu**, které zabraňuje obsazení celé vyrovnávací paměti, to jest omezení dat, které lze v proudu odeslat
- **řízení toku připojení**, které zabraňuje odesílatelům překročit vyrovnávací paměť přijímače v rámci všech toků

4.9 Forward error correction

Další zajímavou funkcí nového protokolu je následná oprava chyb, která spolupracuje se z řízením ztráty paketů. Jedná se o korekci chyb, kdy vysílač odešle redundantní data a přijímač přijme jen část. Tento segment je ovšem díky redundanci schopen přijímač opravit. Společnost Google tuto funkci testovala, ale testy nepřinesly žádné relevantní výsledky. Na webovém serveru YouTube se zvýšili odezva i počet obnovení vyrovnávací paměti. Momentálně není funkce v návrhu IETF Quic. [16]



Obr. 4.4: Diagram stavů přijímaných částí toku.[17]

5 Implementace rozšíření

V této kapitole autor popisuje způsob implementace rozšíření, zajímavé zdrojové kódy a problémy, se kterými se setkal při programování. Při práci autor používá nástroje jako Git, Java IDE IntelliJ IDEA Educational, které jsou v dnešní době nezbytné pro rychlý a kvalitní vývoj. Pro kompilaci Java kódu autor používá nástroj Gradle, který je v základním nastavení repozitáře aplikace JMeter.

5.1 Nastavení prostředí a kompilace

Zdrojové kódy nástroje JMeter jsou kompilovány v nástroji Gradle[1]. Autor pro vývoj rozšíření zvolil stejný nástroj oficiálního repozitáře. Pro vývoj rozšíření byla stáhnutá poslední verze JMeteru. Byla vytvořena složka pro umístění zdrojových kódů rozšíření. Vytvořenou složku bylo nutné přidat pro kompilaci, kteráje popsána ve zdrojovém kódu nacházející se v souboru *build.gradle.kts*. Pro práci s celým projektem je přiložen jednoduchý skript, který umožňuje následující operace:

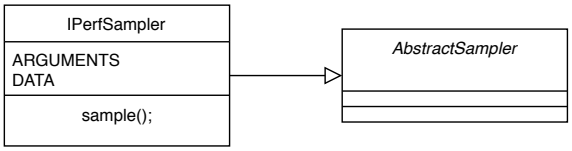
- kompilace projektu pomocí příkazu *gradlew build*
- pustit jednotkové testy *gradlew test*
- vygenerování dokumentace *gradlew doc*
- spuštení grafického rozhraní pro vývoj *gradlew runGui*
- další procesy, které jsou určeny k publikaci, vygenerování poměru pokrytí kódu testy, atd.

5.2 Implementace iPerf Sampler

Při implementaci sampler bylo nutné realizovat interní volání programu. Díky implementaci v jazyce Java, byla využita třída *ProcessBuilder*, která slouží k vytváření procesů. Při spuštění testovacího plánu je v rozšíření vytvořen proces, který čeká na odpověď interního programu iPerf a následnou asercí návratového kódu vyhodnocuje, zda volání programu proběhlo v pořádku či nikoliv. Návratový kód je zpracován ve všech přehledech a informuje uživatele o stavu požadavku.

Výstupem volání požadavku jsou základní metriky popsané v sekci 3.1.2. Při kladném vyhodnocení správného návratového kódu musí sampler sestavit objekt pro výsledné hodnoty. Nástroj iPerf měří propustnost, ale pro nástroj JMeter je metrika počet volání požadavků za sekundu. Čas trvání nesmí být nastaven z doby trvání procesu, ale musí být nastaven z času měření programu iPerf. Při úspěšném

[1] https://gradle.com



Obr. 5.1: Aby JMeter nový sampler správně zaregistroval, je nutné novou třídu dědit z třídy *AbstractSampler*

provedení měření a zpracování výsledků z programu iPerf je instanciován objekt sampleResults typu *SampleResult* a jsou nastaveny následující hodnoty:

- *responseCode* - návratový kód
- *sentBytes* - počet odeslaných bytů
- *successful* - bool hodnota, zda požadavek proběhl v pořádku
- *responseMessage* - textový výstup z volání programu iPerf
- *responseStartTime* - časová známka začátku volání nástroje
- *responseEndTime* - časová známka konce volání nástroje

Tyto výsledné hodnoty jsou dál zpracovávány nástrojem JMeter. Pokud nástroj programu není nainstalován, uživatel se to dozví v textové odpovědi výpisu.

```
/**
 * {@inheritDoc}
 */
@Override
public SampleResult sample(Entry e) {
    trace("sample()");

    SampleResult sampleResults = new SampleResult();
    sampleResults.setContentType("iperf");
    sampleResults.setSampleLabel(getTitle());
    cmds.add(command);
    SystemCommand nativeCommand = null;
    File directory = new File(FileServer.getDefaultBase());

    Arguments args = getArguments();

    for (int i=0; i < args.getArgumentCount(); i++) {
        Argument arg = args.getArgument(i);
        cmds.add(arg.getPropertyAsString(Argument.VALUE));
    }
}
```

Výpis 5.1: Ukázka vytvoření procesu pro spuštění nástroje iPerf

5.3 Implementace tPerf Sampler

Z důvodu fungování protokolu na aplikační vrstvě s využitím UDP lze naleznout mnoho knihoven pro QUIC klienta i server. V tabulce 5.3 je přehled implementací, se kterými se autor setkal při výběru. Pro implementaci rozšíření byla využita implementace MVFST a to především z důvodu přehledné dokumentace a dostupných informací. Autoři tvrdí, že tato implementace je využívána v produkci na některých produktech firmy Facebook. Implementace KWIK v jazyce Java nebyla vhodná pro výběr, protože není doporučené nasazení do produkce. Výhodou je také jednodušší kompilace a menší velikost. Implementace QUIC v Chromium má složitou kompilaci a nezkompilovaný kód zabírá zhruba 20 Gb. Implementace MVFST je využitá v knihovně Proxygen[2], ve které je možné pustit server a klienta v módu HTTP přes protokol QUIC. Existují i knihovny pro Javu, které ale postrádají informace o tom, zda jsou vhodné pro výkonnostní testy, proto při výběru nebyly brány v úvahu. Více knihoven je možné naleznout v příloze F.

Název projektu	**Jazyk**	**Odkaz**
Libquic	C++	https://github.com/devsisters/libquic
MVSFT	C++	https://github.com/facebookincubator/mvfst
Chromium QUIC	C++	https://www.chromium.org/quic
KWIK	Java	https://bitbucket.org/pjtr/kwik/src

Pro spuštění je využívána třída *ProcessBuilder*, které je při vytváření objektu předána složka se zkompilovaným kódem. Při spuštění měření sampler čeká na návratový kód programu a při úspěšném provedení měření jsou nastavy metriky stejně jako v případě iPerf.

[2]https://github.com/facebook/proxygen

5.4 Implementace Visualizéru

Z důvodu rozdílných metrik JMeteru bylo nutné implementovat vlastní grafické rozhraní pro zobrazení grafu propustnosti. Aby JMeter toto rozšíření zaregistroval, musí dědit od abstraktní třídy *AbstractVisualiser*. Při spuštění testů jsou výsledky testů předávány do instance třídy *ThroughputChartVisualiser* pomocí metody zobrazené v kódu 5.2, který je v reálném čase vykresluje.

```
@Override
@SuppressWarnings("SynchronizeOnNonFinalField")
public void add(SampleResult sample) {
    if (sample.getResponseCode() != "0") {
        for (SampleResult sr : sample.getSubResults()) {
            throughputChart.addValueToChartData(
                    (float) (sr.getStartTime() - sr.getEndTime()),
                    (float) sr.getSentBytes(),
                    sample.getContentType()
            );
        }
        throughputChart.addValueToChartData(
                (float) (sample.getStartTime() - sample.getEndTime
    ()),
                (float) sample.getSentBytes(),
                sample.getContentType()
        );
    }
}
```

Výpis 5.2: Ukázka předání dat pro vykreslení hodnot v reálném čase do grafu

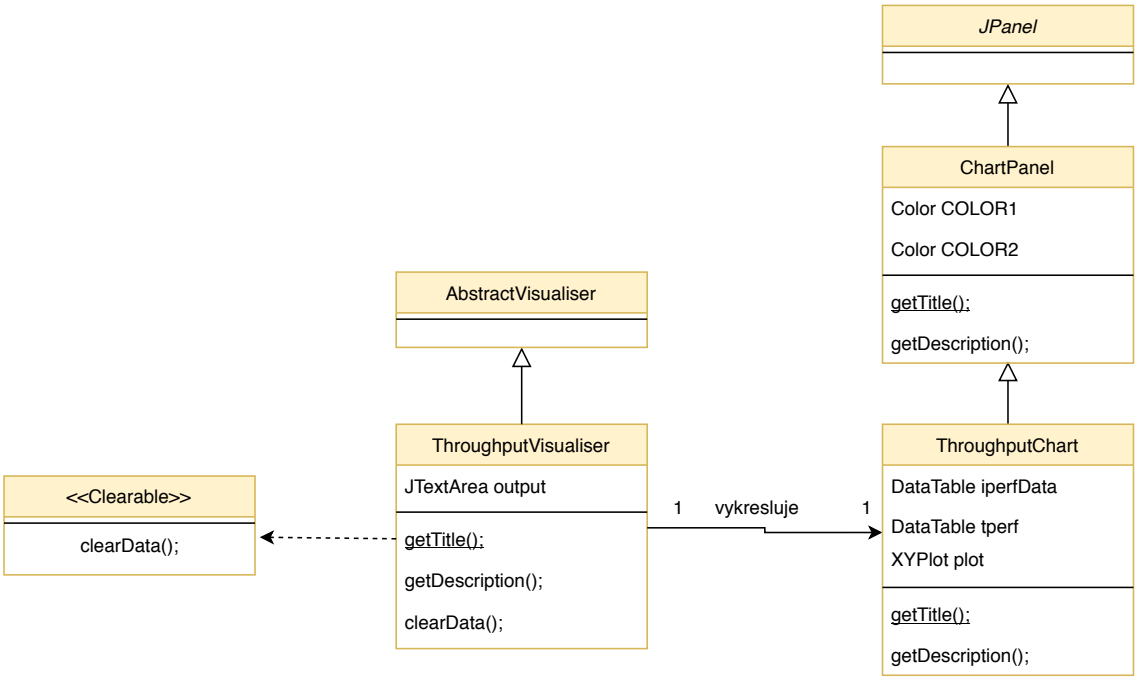
Knihoven, které vykreslují grafy existuje několik. Pro zobrazení grafů byla vybrána knihovna Gral. Tato knihovna byla zvolena [3] pro svoji přehlednou dokumentaci a vzhled. Při implementaci grafů bylo nutné vyřešit především zobrazení os s adaptací na různé jednotky dat. Výška grafu na ose je vždy o 10 % větší než je nejvyšší hodnota v datech a graf začíná vždy v nule.

5.5 Návrh grafického rozhraní

Nástroj JMeter používá pro své grafické prvky framework Swing[4]. Rozšíření má předem daný vzhled a před implementací byly vytvořeny jednoduché drátové modely. Autor se snažil při implementaci grafického rozhraní o dodržení stejné konvence,

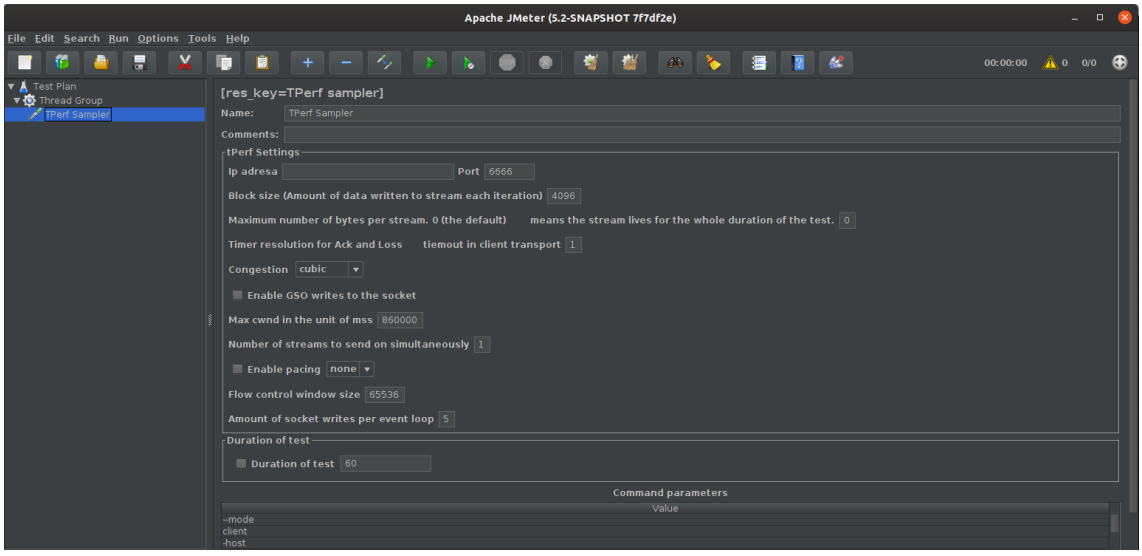
[3] http://github.com/gral

[4] https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html



Obr. 5.2: Diagram tříd pro zobrazení návrhu zobrazování dat na grafu.

která je napříč celou aplikací. Při návrhu bylo dbáno na to, aby jednotlivé parametry odpovídaly grafickým prvkům, tedy booleanovské hodnoty jako *Checkbox*, výběr z více možností jako *SelectBox* a podobně. Zpětná vazba rozhraní je založena na zobrazení výstupu z volaných požadavků a návratovém kódu.



Obr. 5.3: Ukázka konfigurace pro měření QUIC propustnosti. Jednotlivé parametry jsou reprezentovány odpovídajícím grafickým prvkem.

Každý sampler musí mít své grafické rozhraní, které umožňuje jeho konfiguraci.

Toto grafické rozhraní musí dědit od *AbstractSamplerGui*. Třída grafického rozhraní vytváří instanci sampler a umožňuje předat konfiguraci z uživatelských vstupů. Zdrojové kódy vytvoření instance sampler a jeho konfigurace lze naleznout v kódech níže.

```
/**
 * {@inheritDoc}
 */
@Override
public TestElement createTestElement() {
    IPerfSampler sampler = new IPerfSampler();
    modifyTestElement(sampler);
    return sampler;
}
```

Výpis 5.3: Vytvoření instance sampler, které je předána konfigurace z uživatelských vstupů

```
/**
 * {@inheritDoc}
 */
@Override
public void modifyTestElement(TestElement te) {
    super.configureTestElement(te);
    IPerfSampler iPerfSampler = (IPerfSampler) te;
     // dekorace objektu
    iPerfSampler.setProperty("congestion", "tcp");

    Arguments args = new Arguments();
    args.addArgument(new Argument("Client mode", "--client"));
    args.addArgument(new Argument("IP Address", this.
   serverIpAddress.getText()));
    // další konfigurace ...

    iPerfSampler.setArguments(args);
}
```

Výpis 5.4: Ukázka konfigurace sampler pomocí předání pole argumentů a dekorace objektu za běhu o nové vlastnosti

6 Testování a interpretace výsledků

Cílem této kapitoly je provést experimentální testy pomocí navržené metodiky a demonstrovat výsledky výkonnostních testů protokolů TCP a QUIC. Testy jsou prováděny ze symetrické linky od poskytovatele O2 a přes síť LTE od poskytovatele Vodafone.

6.1 Použitý hardware

Při testování byl na straně klienta použit osobní počítač s operačním systémem UBUNTU 18.04 s následujícími parametry:

- Procesor : Intel i5, 5400k, 2.4 Ghz
- Ram : 8 Gb ram
- Hdd : 256 Gb SSD
- Připojení : O2 VDSL (20/2 Mb/s), 22 ms RTT, běžně 0% ztrátovost paketů

Server je hostován u WEDOS Internet, a.s. s IPv6 i IPv4 adresou s následujícími parametry:

- 120 GB HDD SSD RAID 10
- 16 GB RAM
- 4 procesorove jednotky
- linka připojená k páteřní síti o rychlosti 100 Mb/s, 5 ms RTT
- IPv4 adresa: 46.28.110.37
- IPv6 adresa: 2a02:2b88:2:1::6dfe:1/64

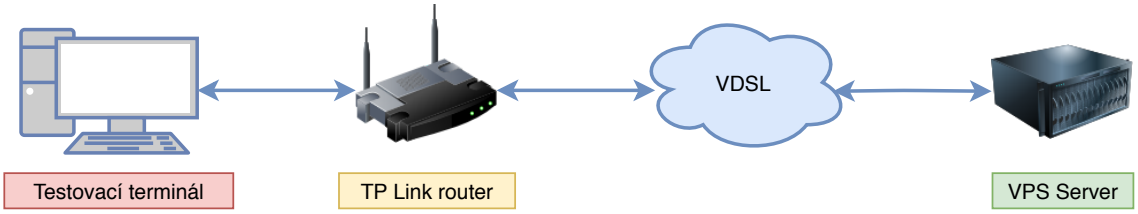
6.2 Testy propustnosti TCP a QUIC

Cílem této kapitoly je popsat experimentální měření výkonnosti protokolů QUIC oproti TCP. Pro měření je využit nástroj JMeter a rozšíření pro měření propustnosti protokolů QUIC a TCP založených na nástrojích iPerf a nástroji tPerf implementace MVSFT QUIC.

6.2.1 Analýza sítě

Prvním krokem provedené analýzy bylo ověření, zda není na lokální síti jiný provoz než na testovacím terminálu. Grafické rozhraní zařízení TP Link ukazuje počet připojených klientů. Tímto jednoduchým způsobem bylo ověřeno, zda nejsou výsledky zkresleny jiným provozem. Na straně serveru byla zjišťována omezení zkreslující výkonnost. Pomocí nástroje iPerf bylo provedeno měření, kde virtuální server byl

využit jako klient a jako server byl použit veřejný server z oficiálních stránek nástroje iPerf.



Obr. 6.1: Obrázek popisující zapojení síťě pro měření.

6.2.2 Maximální přenosová jednotka cesty a obousměrné zpoždění

Pro výpočet maximální přenosové jednotky cesty je podle doporučení RFC6349 využit ICMP ping. Pakety ICMP protokolu jsou nastaveny na maximální možnou velikost 65 535 bajtů. Pro test je spuštěn příkaz 6.1, ve kterém je nastavena IP adresa serveru. Pro adresy serveru IPv4 i IPv6 byla naměřena hodnota **1500B**. Obousměrné zpoždění bylo změřeno pomocí nástroje ping. Pro IPv6 i IPv4 adresy bylo naměřeno zpoždění **22ms**.

```
1 ping ipAdresa -M do - s 65535 -c 1
```

Výpis 6.1: Příkaz ping pro zjištění maximální přenosové jednotky cesty.

```
x@martinstainer-DESKTOP:~$ ping 46.28.110.37 -M do -s 65507 -c 1
PING 46.28.110.37 (46.28.110.37) 65507(65535) bytes of data.
ping: local error: message too long, mtu=1500

--- 46.28.110.37 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

x@martinstainer-DESKTOP:~$ ping 2a02:2b88:2:1::6dfe:1 -M do -s 65535 -c 1
PING 2a02:2b88:2:1::6dfe:1(2a02:2b88:2:1::6dfe:1) 65535 data bytes
ping: local error: message too long, mtu: 1500

--- 2a02:2b88:2:1::6dfe:1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Obr. 6.2: Testování maximální přenosové jednotky cesty.

6.2.3 Konfigurace TCP

Pro získání optimálních výsledků je nutné změřit oboustranné zpoždění a Bottleneck Bandwith. Tyto hodnoty slouží k výpočtům parametrů TCP RWND a velikosti vyrovnávací paměti. Vypočítané hodnoty musí být nastaveny na straně serveru i klienta. Při špatně provedeném nastavení parametrů může být výkon TCP limitován. Špatné nastavení velikosti vyrovnávací paměti může způsobovat zahazování paketů.

Následující výpočty jsou pro tok generovaný ze strany serveru:

Výpočet parametru BDP

$$bdp\ [b] = rtt\ [s]\ *\ bb\ [b/s] \tag{6.1}$$

$$bdp\ [b] = 0.022\ [s]\ \cdot\ 2 \cdot 1024 \cdot 1024\ [b/s] \tag{6.2}$$

$$bdp\ = 461373\ b \tag{6.3}$$

Výpočet parametru TCP RWND

$$tcp\ rwnd\ \ [B] = \frac{461373\ [b]}{8} \tag{6.4}$$

$$tcp\ rwnd\ = 56,31\ kB \tag{6.5}$$

Počet spojení

$$N\ \ [-] = \frac{bdp\ [b]}{tcp\ rwnd\ [B]} \tag{6.6}$$

$$N\ \ [-] = \frac{461373\ [b]}{57671\ [B]} \tag{6.7}$$

$$N\ = 8\ [-] \tag{6.8}$$

Následující výpočty jsou pro tok generovaný ze strany klienta:

Výpočet parametru BDP

$$bdp\ [b] = rtt\ [s]\ *\ bb\ [b/s] \tag{6.9}$$

$$bdp\ [bit] = 0.022\ [s]\ \cdot\ 2 \cdot 1024 \cdot 1024\ [b/s] \tag{6.10}$$

$$bdp\ = 46137\ \ [b] \tag{6.11}$$

Výpočet parametru TCP RWND

$$tcp\ rwnd\ \ [B] = \frac{46137\ [b]}{8} \tag{6.12}$$

$$tcp\ rwnd\ \ = 5,63\ kB \tag{6.13}$$

Počet spojení

$$N\ [-] = \frac{bdp\ [b]}{tcp\ rwnd\ [B]} \tag{6.14}$$

$$N\ \ = \frac{46137\ [b]}{5767\ [B]} \tag{6.15}$$

$$N\ \ = 8\ [-] \tag{6.16}$$

6.2.4 Konfigurace QUIC

I přes to, že neexistuje žádné doporučení jak testovat propustnost protokolu QUIC, bylo nutné protokol nakonfigurovat. Implementace QUIC protokolu MVFST je v základním nastavení a její výchozí hodnoty nejsou dostatečné pro změření celkového výkonu. V prvotním měření byla hodnota propustnosti 5 Mb/s. V repozitáři na stránce github je založený úkol, díky kterému by měl vzniknout dokument, jak správně implementaci nastavit. Nástroj tPerf umožňuje nastavit počet toků. Pro měření byl nastaven stejný počet jako v případě TCP, tedy osm toků. Lepších výsledků se dosáhne také se zvětšením parametru *Window Size*. Ověřením konfigurace bylo několik testů s různými kombinacemi. Nejlepších výsledků bylo dosáhnuto spuštěním nástroje tPerf s následující mi parametry:

```
1 tperf -gso true -pacing true -window 262144 -num_streams 8 -
     max_cwnd_mss 860000
```

Výpis 6.2: Nejlepších výsledků bylo dosáhnuto při spuštění nástroje tPerf s následujícími parametry.

6.2.5 Měření propustnosti

Pro měření propustnosti protokolů QUIC a TCP byly použity nástroje iPerf3 a tPerf. Pro jejich spuštění byly napsány rozšíření do nástroje JMeter, které umožňují automatické testování s reprezentací výsledků v grafu. Před začátkem samotného měření bylo vyzkoušeno několik testových měření přes IPv4 a IPv6, ale nebyl zjištěn

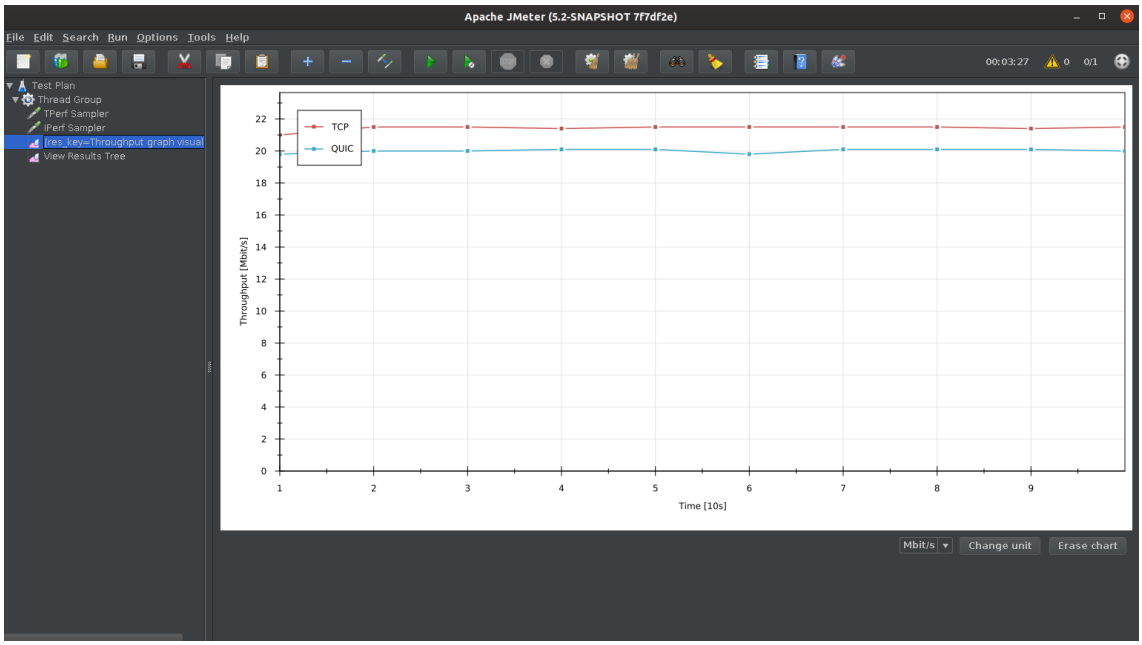
žádný výrazný rozdíl, proto další měření probíhaly s použitím IPv6 adresy. Před dalším měřením byly vyzkoušeny různé porty, ale nebylo zjištěno upřednostňování žádné služby. V tabulce 6.2 bylo naměřeno deset vzorků. Doba trvání měření jednoho vzorku byla nastavena na 15s. Výsledný průměr ukazuje že TCP vykazuje vyšší propustnost než QUIC.

Číslo měření	**TCP [Mb/s]**	**QUIC [Mb/s]**
1	21	19.9
2	21.5	20.9
3	21.4	20.3
4	21.5	20.2
5	21.5	20.9
6	21.4	20.2
7	21.5	19.8
8	21.4	20.1
9	21.5	20.1
10	21.4	20.6
Průměr	**21.43**	**20.25**

Tab. 6.1: Naměřené hodnoty propustnosti pro tok generovaný ze strany serveru.

Číslo měření	**TCP [Kb/s]**	**QUIC [Kb/s]**
1	2181	2048
2	2262	2073
3	2358	2146
4	2164	1986
5	2237	1929
6	2193	2045
7	2298	2131
8	2315	2098
9	2283	2084
10	2331	2096
Průměr	**2262**	**2063**

Tab. 6.2: Naměřené hodnoty propustnosti pro tok generovaný ze strany klienta.



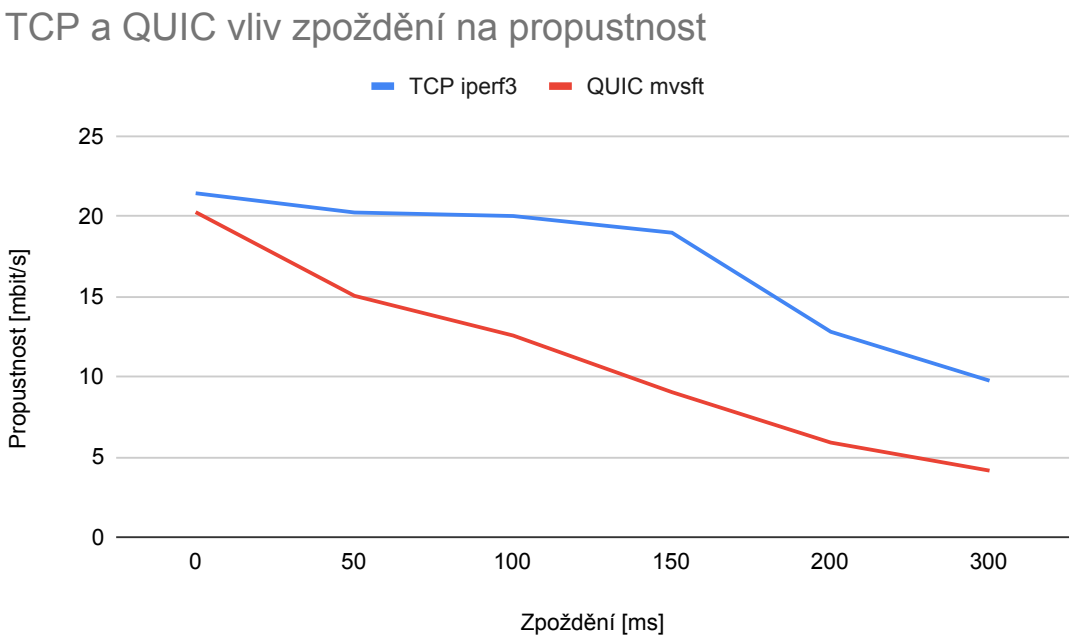
Obr. 6.3: Nástroj JMeter s rozšířeními pro měření propustnosti QUIC a TCP.

6.2.6 Vliv zpoždění na propustnost

Pro zjištění vlivu zpoždění na propustnost bylo provedeno měření o deseti vzorcích pro oba testované protokoly. Po provedení deseti vzorků bylo emulováno vyšší zpoždění. Funkčnost tohoto nastavení byla ověřena nástrojem ping a byl proveden další test. Pro testování byl využit implementovaný nástroj v nástroji JMeter, který pro měření TCP a QUIC propustnosti používá nástroj iPerf3 a tPerf.

Z výsledků viditelných na grafu 6.5 lze usuzovat, že zpoždění způsobuje ztrátu propustnosti obou technologií. Zpoždění má mnohem větší vliv na propustnost u protokolu QUIC než TCP. Pro obě technologie byl nastaven stejný parametr pro řízení toku. Podobné měření provedli uživatelé MVFST v simulované síti Mininet a dosáhli obdobných výsledků.[1]

[1] https://github.com/facebookincubator/mvfst/issues/119



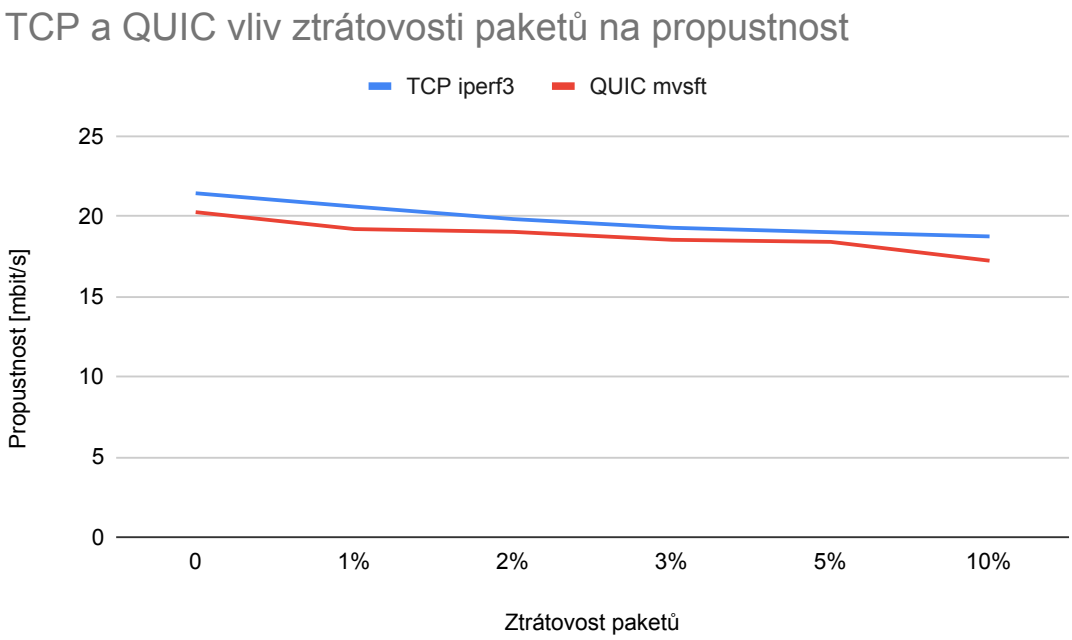
Obr. 6.4: Vliv zpoždění na propustnost porovnávaných technologií.

6.2.7 Vliv ztrátovosti paketů na propustnost

Pro zjištění vlivu zpoždění na propustnost bylo provedeno měření o deseti vzorcích pro obě testované technologie. Výsledné hodnoty na grafu jsou průměrné hodnoty z těchto vzorků. Po provedení testů bylo nastaveno zpoždění. Funkčnost toho nastavení byla ověřena nástrojem ping a poté byl proveden další test. Pro testování byl využit implementovaný nástroj v nástroji JMeter, který pro měření TCP a QUIC propustnosti používá nástroj iPerf3 a tPerf.

Z výsledků viditelných na grafu 6.5 lze usuzovat, že ztrátovost paketů způsobuje ztrátu propustnosti obou protokolů. Pro obě technologie byl nastaven stejný parametr pro řízení toku. Podobné měření provedli uživatelé MVFST v emulované síti a dosáhli obdobných výsledků.[2]

[2]https://github.com/facebookincubator/mvfst/issues/119



Obr. 6.5: Vliv ztrátovosti paketů na propustnost porovnávaných technologií.

6.3 Experimentální test rychlosti načítání webových stránek

V této podkapitole se autor zaměřuje na výkonnostní rozdíly mezi aktuálně používaným balíčkem protokolů TCP, TLS, HTTP2 a jeho alternativou HTTP2 přes QUIC. Transportní protokol QUIC byl primárně vytvořen, aby snížil odezvu webových stránek. Z tohoto důvodu jsou testy prováděny pomocí nástroje Lighthouse, který dokáže měřit celkovou výkonnost webu a průběh rychlosti načítání stránek. Výkon webových stránek je popsán pomocí několika metrik, které vcelku určují celkové skóre výkonnosti.

6.3.1 Lighthouse

Pro srovnání balíčků TCP, TLS a HTTP2 a jeho alternativy HTTP2 přes QUIC si autor vybral nástroj Lighthouse a to z důvodu podpory obou zmíněných technologií. Nástroj Lighthouse komplexně hodnotí webové stránky a v následujících testech je uvažována pouze kategorie *výkon*. Nástroj Lighthouse používá následující výkonnostní metriky:

- **First Contentful Paint** – tato metrika ukazuje čas, kdy se vykreslil první text nebo obrázek.
- **Speed Index** – vyjadřuje čas, kdy byl dostupný obsah.

- **Time to Interactive** – čas potřebný k tomu, aby webová stránka byla pro uživatele interaktivní.
- **First Meaningful Paint** – vyjadřuje čas, který je potřeba k tomu, aby se načetl hlavní obsah.
- **First CPU Idle** – označuje čas, ve kterém je procesor dostatečně výkonný, aby zvládl vykreslit obsah.
- **Max Potential First Input Delay** – maximální zpoždění, které můžou uživatelé zažít při načítání.

Nástroj Lighthouse v základním nastavení simuluje CPU throttling a network throttling[3] a měří i další aspekty webových stránek. Všechny tyto možnosti emulace v nástroji byly při spouštění nástroje vypnuty a emulace síťě byla prováděna pomocí nástroje NetEm.

6.3.2 Měřicí skript

Měření bylo zautomatizováno pomocí skriptu v NodeJS. Na pátém řádku je možné nastavit adresu webového serveru. Před spuštením je nutné ověřit, zda server podporuje QUIC. Při nedostupnosti QUIC nástroj Lighthouse přejde na komunikaci pomocí TCP. Na sedmém řádku je definován seznam s nastavením pro emulaci sítě. Na desátém řádku je definován seznam složek, do kterých se ukládají výstupy testů.

```
1  #!/usr/bin/env node
2
3  const execSync = require('child_process').execSync;
4  const { exec } = require('child_process');
5  let url = "https://www.youtube.com";
6  let runLimit = 10;
7  var tests = [
8       'tc qdisc replace dev enp0s31f6 root netem delay 0ms',
9  ];
10 var testsFolders = [
11     '0-ms-0-delay',
12 ];
```

Výpis 6.3: Nastavení parametrů pro emulaci a složek, do kterých se ukládají výsledky.

```
1  function doTest(command, folder) {
2      let runs = 0;
3      do {
4          console.log(`Starting performance test ${runs + 1}`);
```

[3]throttling – emulace omezení výkonu

```
        const file = "${folder}-${runs + 1}";
        try {
            execSync(`lighthouse ${url} --disable-setuid-sandbox --no-sandbox --only-categories=performance --emulated-form-factor=none --throttling-method=provided --output=json --output-path=./${folder}/${runs + 1}-report.json --chrome-flags="--headless --enable-quic --origin-to-force-quic-on=${url}:433"`);
        } catch
            (err) {
            console.log(`Performance test ${runs + 1} failed`);
            exit;
        }
        console.log(`Finished running performance test ${runs + 1}`);
        runs++;
    }
    while (runs < runLimit);
}
for(let i = 0; i < tests.length; i++) {
    doTest(tests[i], testsFolders[i]);
}
```

Výpis 6.4: Testovací funkce bez výpisů a nastavení emulace sítě.

```
Sun, 31 May 2020 18:27:42 GMT Printer json output written to ./21-05-2020-quic-0
-loss-0-delay/2-report.json
Sun, 31 May 2020 18:27:42 GMT ChromeLauncher Killing Chrome instance 16282
Finished running performance test 2
Starting performance test 3
```

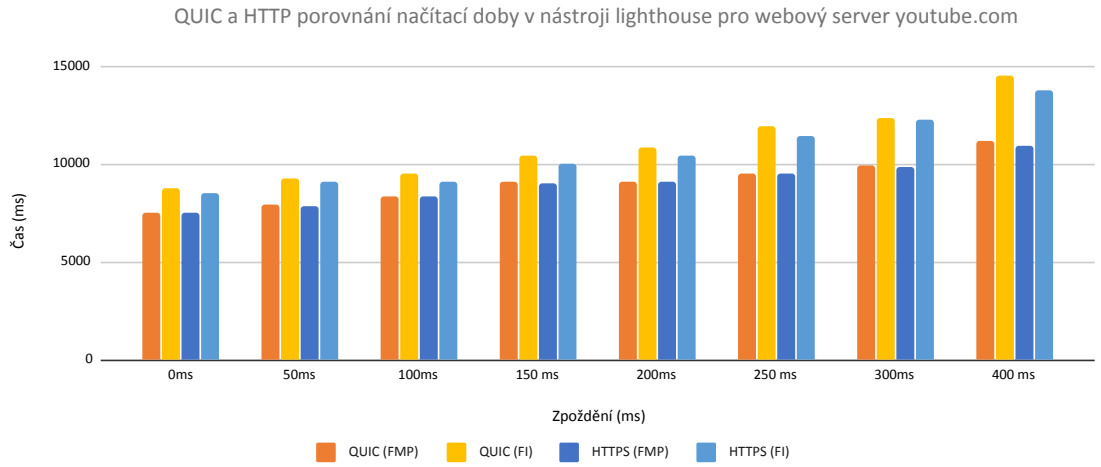
Obr. 6.6: Výpis aktuálního stavu měření je vypsán na standardní výstup.

6.3.3 Měření

Měření pomocí nástroje Lighthouse bylo provedeno pomocí skriptu v NodeJS, který nastavuje parametry emulace síťového rozhraní, spouští testy a jednotlivé zprávy ukládá do složek podle nastavených parametrů. Emulace podmínek v internetovém připojení byla provedena pomocí nástroje NetEm. Jednotlivá nastavení emulátoru sítě bylo ověřeno pomocí nástroje ping, iPerf a Wireshark. Srovnávané hodnoty v grafu jsou mediánem z 21 vzorků. Pro výpočet mediánu byl implementován skript, který prochází zprávy ve formátu JSON a zpracovává je. Jednotlivé vzorky byly měřeny po sobě. Hodnoty konkrétních měření se nacházejí v příloze E.

Vliv zpoždění na rychlost načítání webové stránky

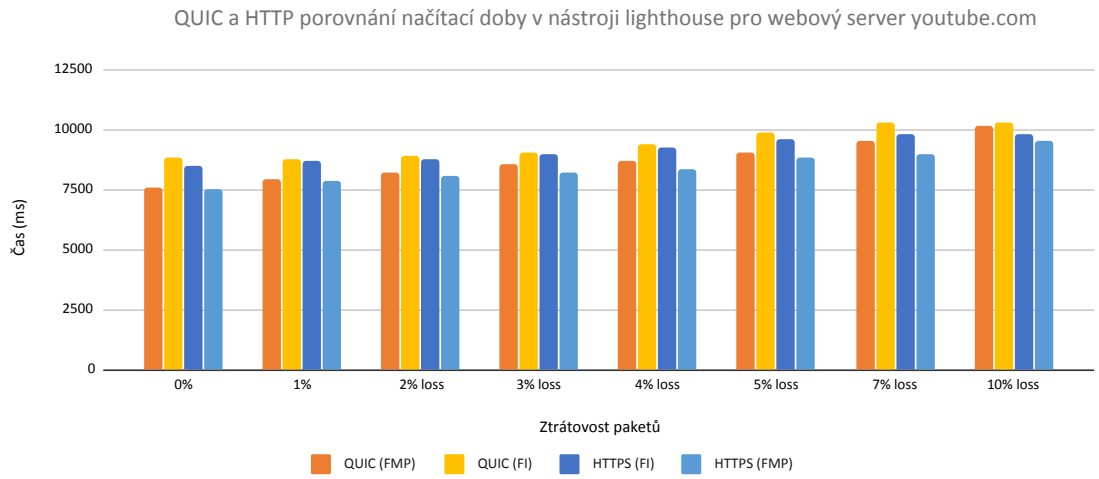
Grafu 6.7 ukazuje naměřené metriky *First Meaningful Paint* a *Time to Interactive*. Měřené výsledky ukazují, že při zhoršených podmínkách v síti klesá výkon – stoupá doba potřebná k načtené webové stránky. Výkon alternativy HTTP přes QUIC je více ovlivněn zpožděním, než běžný balíček HTTP, TLS a TCP.



Obr. 6.7: Graf vlivu zpoždění na rychlost načítání webové stránky www.youtube.com protokolů HTTP2,TLS a TCP a HTTP2 a QUIC.

Vliv ztrátovosti paketů na rychlost načítání webové stránky přes VDSL

Grafu 6.7 ukazuje naměřené metriky *First Meaningful Paint* a *Time to Interactive*. Měřené výsledky ukazují, že při větší ztrátovosti paketů v síti klesá výkon. Výkon alternativy HTTP přes QUIC je více ovlivněn ztrátovostí paketů, než běžný balíček HTTP, TLS a TCP.

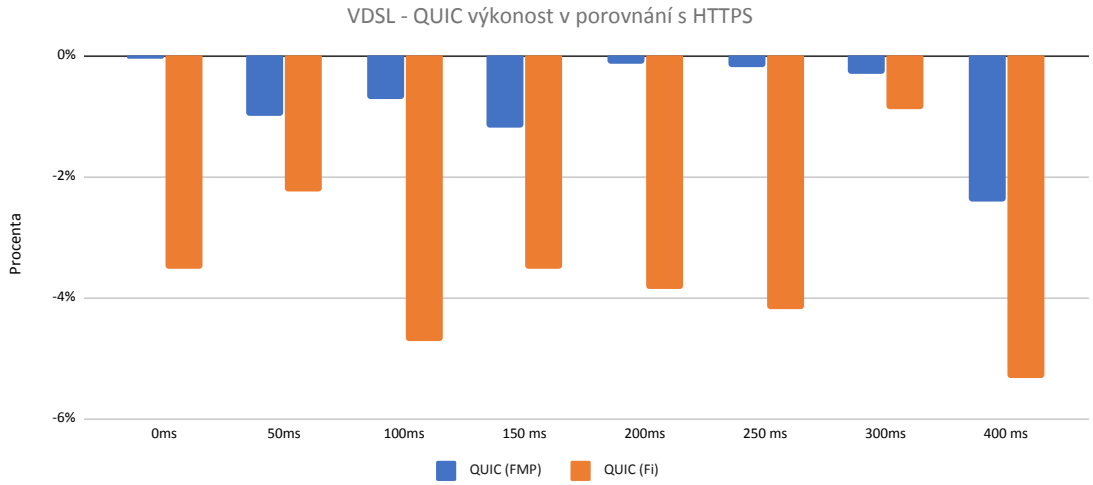


Obr. 6.8: Rozdíly v rychlosti načítání webové stránky www.youtube.com.

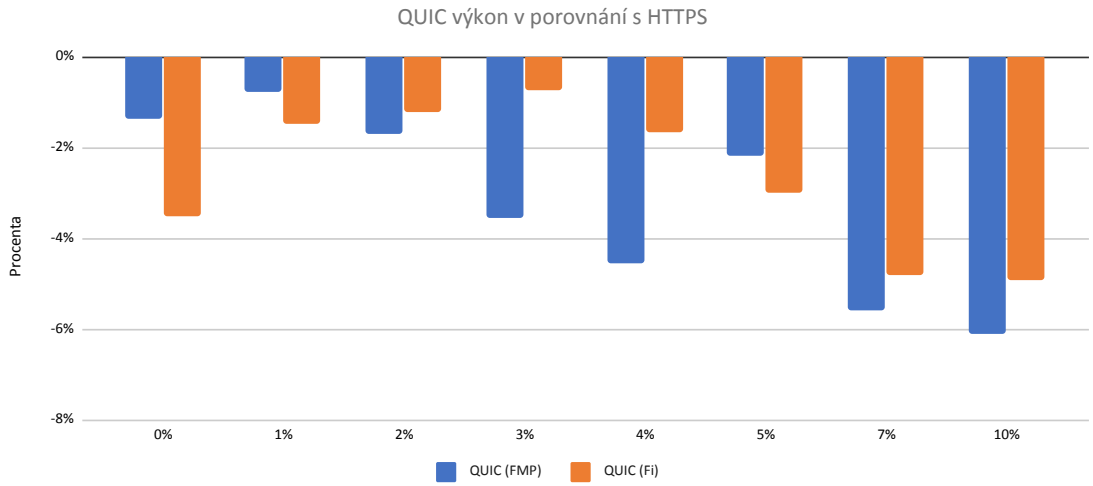
6.3.4 Shrnutí

QUIC je transportní protokol, který funguje na aplikační vrstvě a využívá již zavedeného UDP protokolu. V internetovém prohlížeči Google Chrome (od verze 28 a výše) lze pro navštívení webových stránek využívat nový protokol QUIC. Z hodnot naměřených v nástroji Lighthouse lze usuzovat, že obě technologie se pro různé podmínky v síti liší v jednotkách procent. Výsledné hodnoty na grafech jsou mediánem z 21 vzorků. Načítání webových stránek přes transportní protokol QUIC lze na základě výsledků považovat za pomalejší o několik jednotek procent. Vyhodnocení testů může být ovlivněno následujícími faktory:

- Použití Lighthouse metrik
- Rozdíly mezi GQUIC a QUIC protokolem. Jeden z důvodu pomalejšího načítání by mohlo být použití algoritmu BBR.
- Neznalost nastavení serveru: server může být určen pouze pro integrační testy
- Použití TLSv1.3, které je velmi zrychlené oproti TLSv1.2



Obr. 6.9: Rozdíly v rychlosti načítání webové stránky www.youtube.com



Obr. 6.10: Graf vlivu ztrátovosti paketů na rychlost načítání webové stránky www.youtube.com protokolů HTTP2,TLS a TCP a HTTP2 a QUIC.

6.4 Vyhodnocení experimentálních testů

Byla provedena dvě experimentální měření dvou rozdílných implementací protokolu QUIC. Jedno měření bylo provedeno pomocí JMeteru s rozšířeními pro iPerf a tPerf. Druhé měření bylo provedeno pomocí skriptu, který spouštěl nástroj Lighthouse a nastavoval emulaci podmínek v síti.

Při prvním měření propustnosti bez nastavení emulace podmínek v síti byl rozdíl v průměrné propustnosti zhruba 1 Mb/s. Při emulaci ztrátovosti paketů a zpoždění se ukázal protokol TCP jako mnohem odolnější vůči zhoršeným podmínkám v síti.

S ohledem na fakt, že TCP protokol, který prošel mnohaletým vývoj, stejně jako několika optimalizacemi a je podporován jak softwarově tak i hardwarově, nepovažuje autor práce výsledky nového protokolu QUIC za neuspokojivé. Je nutné také brát v potaz, že QUIC i testována implementace MVFST jsou stále ve fázi vývoje.

Při druhém měření se autor zaměřil na přednosti QUIC, jakožto protokolu, je nový přenosový protokol, který vznikl s primárním účelem zrychlit načítání webových stránek. Nástroj Lighthouse měří výkonnost, podle doby potřebné k načtení webové stránky pomocí několika metrik. Autor se zaměřil na dobu potřebnou k načtení hlavního obsahu a dobu, která je potřeba k tomu, aby internetová stránka byla pro uživatele interaktivní. Při měření pomocí nástroje Lighthouse je spouštěn prohlížeč Chrome, který využívá pro komunikaci přes QUIC implementaci Chromium. Bylo naměřeno 21 vzorků pro dvě metriky. Výsledné srovnávané časy byly hodnoty medián z naměřených vzorků. Z výsledků vyplynulo, že při použití TCP, TLS a HTTPv2 se webové stránky načetly o jednotky procent rychleji.

Pro testování protokolu QUIC lze vybrat jednu z mnoho implementací. Liší se především návrhem, ze kterého vycházejí a rolemi, které podporují. QUIC se ukázal jako rovnocenný soupeř s mnoho novými funkcemi, především migrace připojení a bezpečný přenos jako základ protokolu. Další vývoj tohoto nového protokolu, jestli nahradí TCP i v jiných protokolech než HTTP.

Závěr

Cílem práce bylo navrhnout a realizovat systém pro měření přenosových parametrů datových sítí. V rámci diplomové práce autor definoval přenosové parametry datových sítí, popsal metodiky měření parametrů datových sítí, popsal nový transportní protokol QUIC a navrhl vlastní metodiku měření. Pro vytvořenou metodiku navrhl koncepci síťového testeru, podle které implementoval rozšíření do nástroje Apache JMeter. S pomocí navržené metodiky a implementovaného testeru autor provedl dva druhy experimentálních testů. Autor se ve své práci zaměřil na experimentální testy nového protokolu QUIC v porovnání s TCP.

Momentálně existuje několik desítek implementací protokolu QUIC. Implementace se liší převážně tím, v jakém jazyce byly implementovány a dále také, z jakého návrhu IETF QUIC vychází. Autor pro své testy vybral implementace Chromium QUIC od firmy Google a MVFST od firmy Facebook. Obě tyto implementace jsou populární a běží v produkčním režimu na některých produktech obou společností. Prvně zmíněná implementace je v internetovém prohlížeči Chrome od verze 28 a dá se používat pro navštívení webových stránek podporujících QUIC. Webový server s podporou QUIC Proxygen je založený na implementaci MVFST.

Autor se v prvním testu zaměřil na test propustnosti. V testu se podařilo změřit, že TCP má zhruba o 1 Mb/s vyšší průměrnou hodnotu propustnosti než protokol QUIC. Při testu bylo emulováno zpoždění a ztrátovost paketů v síti. Při emulaci zpoždění a ztrátovosti paketů se ukázalo, že TCP protokol je odolnější proti zhoršujícím se podmínkám v síti.

Protože protokol QUIC vznikl, aby snížil zpoždění při načítání webových stránek, byl druhý test zaměřen na měření výkonu balíčků HTTP, TLS a TCP a jeho alternativě HTTP přes QUIC. Pro test byl napsán skript, který automatizuje měření pomocí nástroje Lighthouse. Na základě testů lze usuzovat, že se webové stránky při použití balíčku HTTP, TLS a TCP načítají o několik jednotek procent rychleji.

QUIC protokol funguje na aplikační vrstvě a díky této vlastnosti je možné nasazení protokolu do produkce bez nutnosti upravovat stávající hardware či software. Protokol QUIC se ukázal jako silný soupeř aktuálně používanému protokolu TCP i přesto, že je stále ve fázi vývoje. TCP má za sebou roky vývoje, optimalizací a hardwarovou podporu. QUIC pro svůj přenos používá protokol UDP. Nové vlastnosti jako migrace připojení, bezpečný přenos dat, 0-RTT uzavření spojení, atd. dělají z QUIC velice zajímavou alternativu, která by nemusela nahradit TCP jen v HTTP protokolu.

Literatura

[1] *Understanding Jitter in Packet Voice Networks* [online]. 2006 [cit. 2000-04-01]. Dostupné z URL: <https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/18902-jitter-packet-voice.html>

[2] CHAPMAN, Chris. *Network performance and security*. Syngress, 2016. ISBN 9780128036013.

[3] GRIGORIK, Ilya. *High Performance Browser Networking*. O'Reilly Media, 2013. ISBN 9781449344764.

[4] *Benchmarking Methodology for Network Interconnect Devices* [online]. Internet Engineering Task Force, 1999 [cit. 2020-02-01]. Dostupné z URL: <https://tools.ietf.org/html/rfc2544>

[5] *Benchmarking Terminology for Network Interconnection Devices* [online]. Internet Engineering Task Force, 1991 [cit. 2020-02-03]. Dostupné z URL: <https://tools.ietf.org/html/rfc1242>

[6] *Framework for TCP Throughput Testing* [online]. Internet Engineering Task Force, 2011 [cit. 2020-02-12]. Dostupné z URL: <https://tools.ietf.org/html/rfc6349>

[7] *Requirements for Internet Hosts – Communication Layers* [online]. Internet Engineering Task Force, 1989 [cit. 2020-02-13]. Dostupné z URL: <https://tools.ietf.org/html/rfc1122>

[8] ADEEL, Ahmed. *VoIP Performance Management and Optimization*. Cisco Press, 2010. ISBN 9781587055287.

[9] EPSTEIN, Joseph. *Scalable VoIP Mobility*. Newnes, 2009. ISBN 9780080949512.

[10] BOCK, Lisa. *Learn Wireshark*. Pack publishing, 2019. ISBN 9781789134506.

[11] *Ethernet service activation test methodology* [online]. International Telecommunication Union, 2011 [cit. 2020-02-14]. Dostupné z URL: <https://www.itu.int/rec/T-REC-Y.1564-201602-I/en>

[12] *Měření datových parametrů sítí pomocí TCP protokolu* [online]. Český telekomunikační úřad, 2014[cit. 2020-03-14]. Dostupné z URL: <https://www.ctu.cz/sites/default/files/obsah/stranky/937/soubory/merenidatovychparametrusitipomocitcpprotokolu.pdf>

[13] *JMeter* [online]. Apache Foundation, [cit. 2020-03-18]. Dostupné z URL: <ttps://jmeter.apache.org/>

[14] *iPerf* [online]. [cit. 2020-03-18]. Dostupné z URL: <https://iperf.fr/>

[15] *nuttcp* [online]. [cit. 2020-03-18]. Dostupné z URL: <http://nuttcp.net/WelcomePage.htm>

[16] SWETT, Ian. *QUIC FECv1* [online]. Google, 2014 [cit. 2020-05-14]. Dostupné z URL: <https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit#heading=h.xgjl2srtytjt>

[17] *QUIC A UDP-Based Multiplexed and Secure Transport* [online]. Internet Engineering Task Force, 2020 [cit. 2020-05-01]. Dostupné z URL: <https://tools.ietf.org/html/draft-ietf-quic-transport-27>

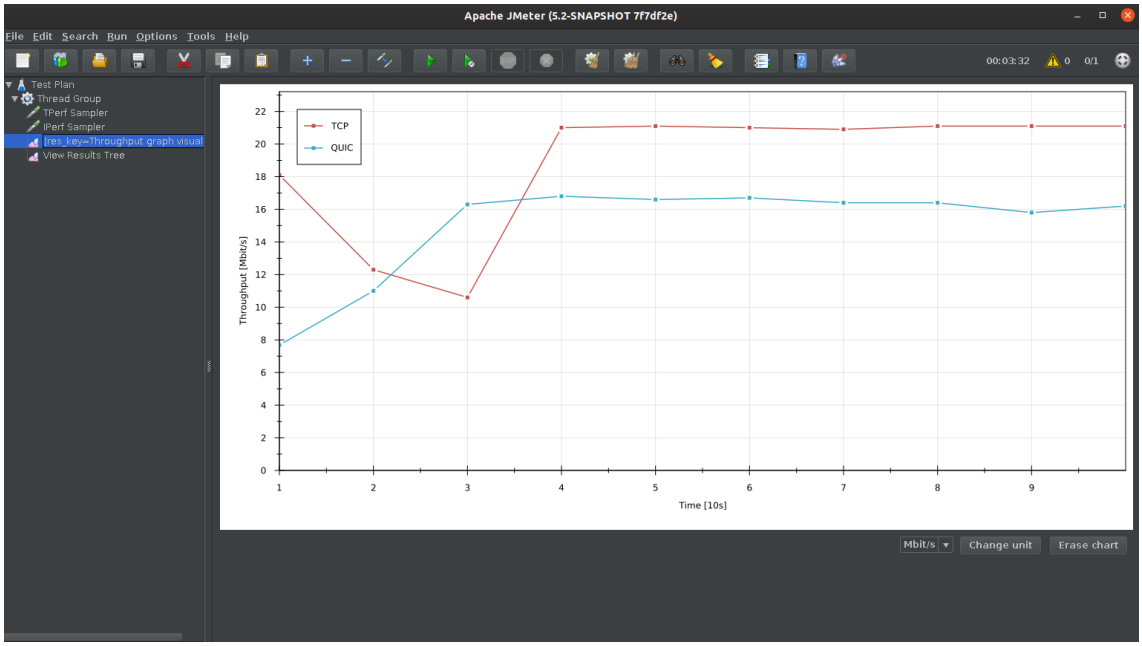
[18] *QUIC* [online]. Internet Engineering Task Force, 2020 [cit. 2020-04-13]. Dostupné z URL: <https://datatracker.ietf.org/doc/charter-ietf-quic/>

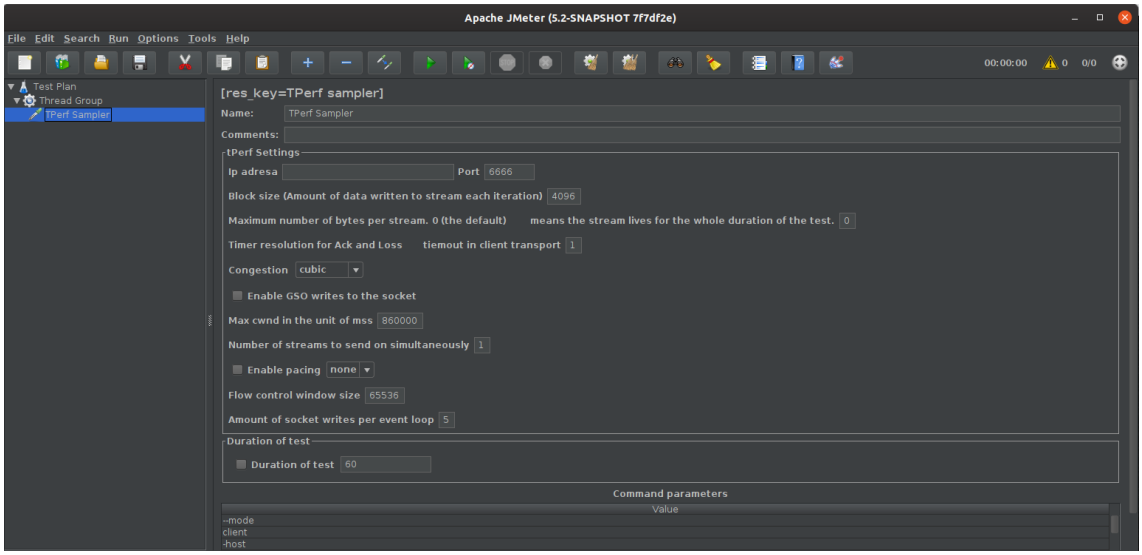
Seznam příloh

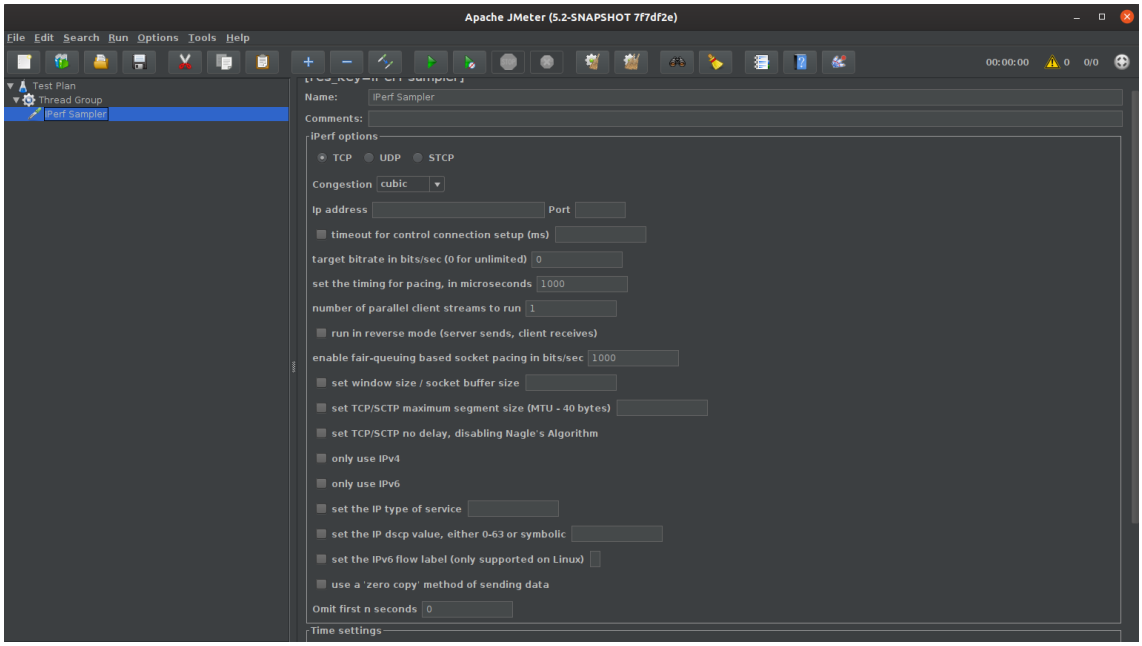
A Obsah přiloženého média

- `/` ..... kořenový adresář přiloženého média
  - `src` ..... zdrojové kódy
  - `snimky` ..... snímky aplikace
  - `pdf` ..... pdf
    - `diplomova-prace.pdf`
  - `text` ..... zdrojové textové soubory
    - `literatura.tex`
    - `prilohy.tex`
    - `reseni.tex`
    - `uvod.tex`
    - `vysledky.tex`
    - `zaver.tex`
    - `zkratky.tex`
  - `readme.txt` ..... soubor s popisem obsahu CD a návodem
  - `tester` ..... zkompilovaný Java kód rozšíření pro JMeter
  - `ligthouse-tester.js` ..... NodeJS zdrojový kód testeru Lighthouse
  - `ligthouse-reporty` ..... složka s reporty ve formátu JSON

B Snímky aplikace






Apache JMeter (5.2-SNAPSHOT 7f7df2e)
File Edit Search Run Options Tools Help
00:00:00 0 0/0
Test Plan
Thread Group
IPerf Sampler
Name: IPerf Sampler
Comments:
iPerf options
TCP UDP STCP
Congestion cubic
Ip address Port
timeout for control connection setup (ms)
target bitrate in bits/sec (0 for unlimited) 0
set the timing for pacing, in microseconds 1000
number of parallel client streams to run 1
run in reverse mode (server sends, client receives)
enable fair-queuing based socket pacing in bits/sec 1000
set window size / socket buffer size
set TCP/SCTP maximum segment size (MTU - 40 bytes)
set TCP/SCTP no delay, disabling Nagle's Algorithm
only use IPv4
only use IPv6
set the IP type of service
set the IP dscp value, either 0-63 or symbolic
set the IPv6 flow label (only supported on Linux)
use a 'zero copy' method of sending data
Omit first n seconds 0
Time settings

C Návod na spuštění Lighthouse skriptu

První řadě je potřeba nainstalovat NodeJS. Následující kód je pro systémy Ubuntu a Debian

```
sudo apt update
sudo apt -y install curl dirmngr apt-transport-https lsb-release ca
   -certificates
curl -sL https://deb.nodesource.com/setup_10.x | sudo bash
```

V druhém kroku je potřeba nainstalovat Lighthouse.

```
npm install -g lighthouse
```

V poslední řadě je nutné nastavit práva pro spuštění.

```
chmod +xa lighthouseTester.js
```

D Instalace rozšíření do JMeteru

Rozšíření bylo vyvíjeno a pro JMeter verze 5.2.1. Nejdříve je nutné provést instalaci použitých nástrojů.

Instalace nástroje iPerf

```
apt-get install -y iperf3
```

Kompilace knihovny pro QUIC

```
git clone https://github.com/facebookincubator/mvfst.git
./build_helper.sh
```

Vložení zkompilovaného rozšíření do JMeter

K úspěšnému spuštění rozšíření je potřeba vložit zkompilovaný Java kód do složky lib/ext v instalační složce JMeter.

E Naměřené hodnoty

Jednotlivé hodnoty lze nalézt v přiloženém médiu ve složce *lighthouse-reporty*.

	1%	2% loss	3% loss	4% loss	5% loss	7% loss	10% loss
QUIC (FMP)	7940	8214	8530	8730	9028	9535	10170
QUIC (FI)	8792	8885	9018	9427	9891	10270	10307
HTTPS (FMP)	7878	8074	8229	8333	8830	9004	9550
HTTPS (FI)	8664	8777	8953	9272	9596	9778	9800

Tab. E.1: Lighthouse: medián hodnoty při emulované ztrátovosti paketů

	50ms	100ms	150 ms	200ms	250 ms	300ms	400 ms
QUIC (FMP)	7948	8391	9109	9104	9561	9913	11215
QUIC (FI)	9274	9533	10403	10876	11961	12380	14508
HTTPS (FMP)	7868	8331	9001	9091	9542	9883	10945
HTTPS (FI)	9066	9084	10037	10458	11459	12271	13736

Tab. E.2: Lighthouse: medián hodnoty při emulovaném zpoždění

F Seznam implementací protokolu QUIC

F.1 aioquic

- jazyk: **Python**
- verze: **Draft 28**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**
- protokol ID: **0xff00001c**

F.2 AppleQUIC

- jazyk: **C, Objective-C**
- verze: **Draft 27**
- role: **klient, server**
- handshake: **TLS 1.3 RFC**
- protokol ID: **0xff00001b**

F.3 ats

- jazyk: **C++**
- verze: **Draft 27**
- role: **klient, server**
- handshake: **TLS 1.3 RFC**
- protokol ID: **0xff00001b**

F.4 Chromium

- jazyk: **C, C++**
- verze: **Draft 27**
- role: **klient, server, knihovna**
- handshake: **QUIC Crypto, TLS**
- protokol ID: **0xff00019, 0xff00001b**

F.5 f5

- jazyk: **C**
- verze: **Draft24-25**
- role: **klient, server**
- handshake: **RC 8446**

- protokol ID: **0xff00018, 0xff000019**

F.6 Haskell quic

- jazyk: **haskell**
- verze: **Draft27**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**
- protokol ID: **0xff0001b**

F.7 Kwik

- jazyk: **Java**
- verze: **Draft28**
- role: **klient**
- handshake: **TLS 1.3**
- protokol ID: **0xff0001c**

F.8 lsquic

- jazyk: **C**
- verze: **Draft 27-28**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**
- protokol ID: **0xff0001c, 0xff00001b**

F.9 msquic

- jazyk: **C**
- verze: **Draft 27-28**
- role: **klient, server**
- handshake: **TLS 1.3 RFC**
- protokol ID: **0xff00019**

F.10 mvfst

- jazyk: **C++**
- verze: **Draft 27**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**

- protokol ID: **0xff00018**

F.11 Neqo

- jazyk: **Rust**
- verze: **Draft 23**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**
- protokol ID: **0xff00017**

F.12 ngtcp2

- jazyk: **C**
- verze: **Draft 28**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3 RFC 8446**
- protokol ID: **0xff0001c**

F.13 ngx_quic

- jazyk: **C**
- verze: **Draft 24-27**
- role: **server**
- handshake: **TLS 1.3 RFC 8446**
- protokol ID: **0xff00017, 0xff00018, 0xff00019, 0xff0001b**

F.14 Node.js QUIC

- jazyk: **C++ a JavaScript**
- verze: **Draft 25**
- role: **klient, server**
- handshake: **TLS 1.3**
- protokol ID: **0xff00019**

F.15 Pandora

- jazyk: **C**
- verze: **Draft 23**
- role: **klient, server**
- handshake: **TLS 1.3**

- protokol ID: **0xff00017**

F.16 picoquic

- jazyk: **C**
- verze: **Draft 27/28**
- role: **klient, server**
- handshake: **TLS 1.3 (picotls)**
- protokol ID: **0xff00017**

F.17 quant

- jazyk: **C11**
- verze: **Draft 28**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3**
- protokol ID: **0xff0001c**

F.18 quiche

- jazyk: **Rust**
- verze: **Draft 23-27**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3 RFC8446**
- protokol ID: **0xff00017, 0xff00018, 0xff00019, 0xff0001b**

F.19 QUICker

- jazyk: **TypeScript**
- verze: **Draft 20**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3 RFC8446**
- protokol ID: **0xff00014**

F.20 Quicly

- jazyk: **C**
- verze: **Draft 27**
- role: **klient, server**
- handshake: **TLS 1.3 RFC8446**

- protokol ID: **0xff00014**

F.21 quiche

- jazyk: **Rust**
- verze: **Draft 23-27**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3 RFC8446**
- protokol ID: **0xff00017, 0xff00018, 0xff00019, 0xff0001b**

F.22 quic-go

- jazyk: **Go**
- verze: **Draft 22**
- role: **klient, server, knihovna**
- handshake: **TLS 1.3 RFC8446**