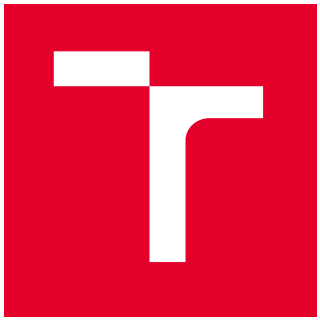


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

ZÁTĚŽOVÉ TESTOVÁNÍ INTERNETOVÉ TELEFONIE

STRESS TESTING OF INTERNET TELEPHONY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Šípek**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Číka, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Martin Šípek ***ID:*** 193937

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

NÁZEV TÉMATU:

Zátěžové testování internetové telefonie

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s protokolem SIP a audio kodeky využívané při internetové telefonii a popište je. Nakonfigurujte IVR v softwarové ústředně Asterisk a připravte scénář zátěžového testování služeb VoIP. Navrhněte a naprogramujte software využívající program JMeter k zátěžovému testování ústředen VoIP. Vyvinutým softwarem realizujte sadu zátěžových testů, výsledky zpracujte do přehledných grafů.

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

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

Vedoucí práce: doc. Ing. Petr Číka, Ph.D.

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

ABSTRAKT

Diplomová práce se zabývá vývojem modulu pro program JMeter, jenž přidává funkcionalitu zátěžového testování protokolu SIP, která doposud v programu chyběla. Teoretická část práce se zabývá internetovou telefonií a kodeky využívající se v telefonii. Je podrobně rozebrán protokol SIP a závěr této části se věnuje zátěžovému testování. Praktická část se zaměřuje především na vývoj navrženého modulu, kdy je dopodrobna popsán kód modulu, ale také se věnuje instalaci a konfiguraci telefonní ústředny Asterisk, na které je navržený modul testován. Výsledky tohoto testování jsou analyzovány na samotném konci této práce.

KLÍČOVÁ SLOVA

Zátěžové testování, JMeter, SIP, VoIP, Java, Asterisk, IVR

ABSTRACT

The thesis deals with the development of a module for JMeter that adds SIP stress testing functionality that was previously missing in the program. The theoretical part of the thesis deals with Internet telephony and codecs used in telephony. The SIP protocol is discussed in detail and the end of this part is devoted to stress testing. The practical part focuses mainly on the development of the proposed module, where the code of the module is described in detail, but it also deals with the installation and configuration of the Asterisk PBX on which the proposed module is tested. The results of this testing are analyzed at the very end of this thesis.

KEYWORDS

Stress Testing, JMeter, SIP, VoIP, Java, Asterisk, IVR

ŠÍPEK, Martin. *Zátěžové testování internetové telefonie*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 68 s. Diplomová práce. Vedoucí práce: doc. Ing. Petr Číka, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Martin Šípek
VUT ID autora:	193937
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Zátěžové testování internetové telefonie

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. Petru Číkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci, panu Ing. Martinu Štůskovi za ochotu projevenou při konzultacích, panu Mgr. Ing. Pavlovi Šedovi za propůjčení školního ICT testeru a v neposlední řade paní Bc. Lence Buchtové za neutuchající podporu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Internetová telefonie (Voice over Internet Protocol – VoIP) je metoda sloužící k uskutečňování a přijímání telefonních hovorů využívající Internet, nikoliv běžnou síť pevných linek. Jedná se o převod analogových zvukových signálů na digitální data. Jinými slovy, zvukové vlny, typicky lidský hlas, jsou převedeny na digitální data, což umožňuje používat Internet jako komunikační metodu pro přenos telefonních hovorů. VoIP telefony, ačkoliv většinou vypadají podobně jako telefony pevné linky, fungují zcela odlišně. Systém pevných linek funguje na základě přepojování okruhů, kdežto VoIP využívá přepojování paketů [1, 2].

Přepojování okruhů je spolehlivá, ale neefektivní metoda spojování hovorů. Jedná se o koncept používaný v telefonních sítích více než sto let. Během uskutečnění hovoru mezi dvěma stranami je spojení udržováno po celou dobu trvání hovoru. Jelikož musí být spojeny dva body v obou směrech, spojení se nazývá okruh. To je základem veřejné komutované telefonní sítě, anglicky Public Switched Telephone Network (PSTN) [1].

Datové sítě nepoužívají přepojování okruhů, protože by internetové připojení bylo mnohem pomalejší za předpokladu, kdy by se muselo udržovat konstantní spojení například s webovou stránkou, která je v daný moment prohlížena. Místo toho se informace odesílají a přijímají podle potřeby v daném okamžiku po malých blocích dat s maximální délkou, zvaných pakety, a místo používání vyhrazené linky data putují chaotickou sítí. Přepojování paketů je v současnosti nejčastější způsob přenosu v datových sítích. Jedná se o velice efektivní způsob přenosu informace, jelikož umožňuje směrovat data v síti po nejméně zatížených a takzvaně nejlevnějších linkách a zároveň neomezuje dva komunikující body od souběžné komunikace s dalšími informačními body [1].

Nástup technologie VoIP přinesl koncovým uživatelům i poskytovatelům řadu výhod, ale zároveň přinesl bezpečnostní hrozby, zranitelnosti a útoky, které se dříve v sítích s uzavřenou architekturou, jako je veřejná telefonní síť, nevyskytovaly. Pokud jsou zvuková data přenášena přes veřejný Internet, neexistuje žádná záruka konstantního toku dat nebo spolehlivého přenosu. Veřejně přístupné koncové body SIP (Session Initiation Protocol) jsou také vystaveny útokům typu DDoS (Distributed Denial of Service), které mohou způsobit vážné zhoršení kvality hlasu a úspěšnosti sestavení hovoru. Dokonce i v uzavřeném ethernetovém prostředí je kvalita zvuku internetové telefonie stále zranitelná zpožděním při zpracování toku RTP (Real-Time Transfer Protocol) způsobeným klientem, serverem nebo sítí. Tato zpoždění mohou být způsobena přetížením procesoru nebo síťového zásobníku. Pro správné fungování všech zařízení i celé sítě je nezbytné testovat, analyzovat a vylepšovat jednotlivé prvky, jež se na síti nacházejí. K takovému testování se využívá zátěžové

testování, které dokáže například identifikovat úzká hrdla v testovaných systémech nebo určit maximální provoz, který dokáže dané zařízení obsloužit bez překročení akceptovatelného zpoždění [3].

Hlavním cílem této diplomové práce je navrhnout a implementovat software využívající program JMeter k zátěžovému testování ústředen VoIP a pomocí tohoto softwaru realizovat sadu zátěžových testů. Výsledky testování následně zpracovat do přehledných grafů. Dalšími cíli je seznámení se s protokolem SIP a audio kodeky využívanými v internetové telefonii a nakonfigurování IVR systému, neboli Interactive Voice Response systém, což lze přeložit jako systém interaktivní hlasové odezvy, v softwarové ústředně Asterisk.

1 Internetová telefonie

Jak již bylo zmíněno v úvodu, technologie VoIP slouží k uskutečnění telefonního hovoru využívající Internet. Souběžně s tím funguje i technologie SIP.

SIP (Session Initiation Protocol) je signalizační protokol, který se používá k vytvoření relace mezi dvěma nebo více účastníky, k úpravě této relace a k jejímu případnému ukončení. Jeho hlavní využití se nachází v IP telefonii. Skutečnost, že SIP je otevřený standard, vyvolala obrovský zájem na telefonním trhu [6].

Jedná se o protokol aplikační vrstvy, který obsahuje mnoho prvků protokolu HTTP (Hypertext Transfer Protocol) a protokolu SMTP (Simple Mail Transfer Protocol). Zprávy jsou textově založené a mechanismus požadavek-odpověď usnadňuje řešení problémů. Vlastní přenos dat probíhá pomocí protokolu TCP (Transmission Control Protocol), nebo protokolu UDP (User Datagram Protocol) na čtvrté vrstvě modelu ISO/OSI. Který z těchto protokolů se použije, určuje protokol SDP (Session Description Protocol) [6, 7].

SIP zprávy popisují identitu účastníků hovoru a způsob, jakým mohou být účastníci zastiženi prostřednictvím IP sítě. Uvnitř SIP zpráv lze někdy vidět také deklaraci SDP. Protokol SDP definuje typ mediálních kanálů, které budou pro relaci vytvořeny. Typicky deklaruje, které kodeky jsou k dispozici a jak se mediální zařízení mohou navzájem spojit prostřednictvím IP sítě [6].

Po dokončení této výměny zpráv o nastavení se multimediální data vyměňují pomocí dalšího protokolu, typicky RTP (Real-Time Transmission Protocol) [6].

Protokol SIP byl vyvinut organizací IETF a publikován jako RFC 3261. Díky své flexibilitě téměř úplně nahradil protokol H.323 ve světě VoIP [6].

1.1 Funkce SIP

Protokol SIP má větší rozsah než jen zpracovávání nastavení hovoru. Níže uvedená tabulka 1.1 ukazuje pět hlavních funkcí v rámci SIP z hlediska VoIP [8].

1.2 SIP prvky

Pro úspěšnou výměnu informací mezi dvěma koncovými body musí být nejdříve vytvořeno spojení. Aby bylo možné tohoto dosáhnout, je nezbytné vytvoření SIP sítě, která se může skládat pouze ze dvou koncových bodů, jež by byly přímo propojeny, nebo se může jednat o komplexní síť, skládající se z velkého množství různých SIP prvků. Každý takový prvek je identifikován pomocí SIP URI (Uniform Resource Identifier), což je unikátní adresa v síti [7].

Tab. 1.1: Popis hlavních funkcí protokolu SIP.

Funkce	Popis
Umístění a registrace uživatele	Koncové zařízení se registrují, identifikují a oznamují SIP serverům svou polohu.
Dostupnost uživatele	Použití v každém koncovém zařízení ke zjištění, zda-li je účastník dostupný.
Možnosti uživatele	Používá se k vyjednání o možnostech média, například k dohodě o vzájemně podporovaném hlasovém kodeku.
Nastavení relace	Volanému koncovému bodu je sděleno, že by zařízení mělo zvonit a jsou stanoveny atributy relace pro každý zúčastněný koncový bod.
Správa relace	Používá se ke správě relace, včetně ukončení hovoru, přidání účastníka během probíhajícího hovoru nebo přepojení probíhajícího nebo nepřijatého hovoru.

1.2.1 User agent

User agent, což se dá přeložit jako uživatelský agent, je koncový bod a zároveň jeden z nejdůležitějších síťových prvků SIP sítě. Dokáže zahajovat, měnit nebo ukončovat relace. Může se jednat o softwarový, nebo hardwarový telefon. SIP je založen na architektuře klient-server, proto lze uživatelského agenta dále rozdělit na dvě části.

- **User agent klient** - prvek, který odesílá požadavky a přijímá odpovědi.
- **User agent server** - prvek, který odesílá odpovědi a přijímá požadavky.

Telefon volajícího funguje jako klient, jenž iniciuje volání, a telefon volaného funguje jako server odpovídající na volání [7].

1.2.2 Proxy server

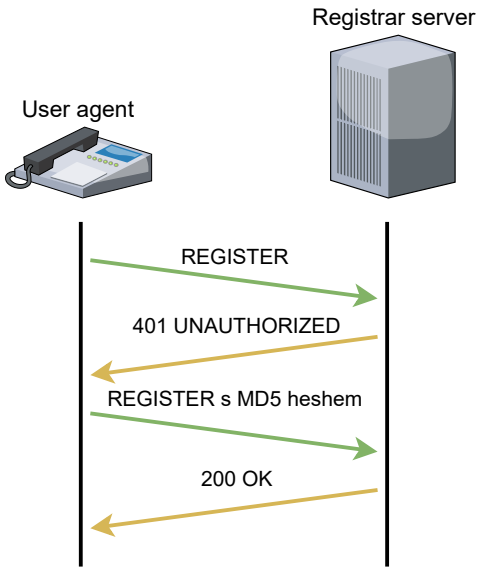
Proxy server je síťový prvek, jenž přijímá požadavky od klientů a přeposílá je jiným uživatelům, nachází se tudíž vždy mezi dvěma koncovými body. V klasické IP síti se dá takový server přirovnat ke směrovači, jelikož směruje provoz. Mimo to také může zajišťovat autentizaci nebo autorizaci uživatelů. Dá se rozdělit na dva různé typy [7, 9].

- **Bezstavový proxy server** - jedná se o základní typ tohoto druhu serveru. Jeho funkce je přijímání a předávání informací potřebných k výkonu své práce, ale neuchovává o těchto úkonech žádné záznamy. Díky této jednoduchosti je snazší takový server škálovat a upgradovat. Na rozdíl od stavového proxy serveru je také rychlejší, jelikož se musí starat pouze o omezený počet funkcí, ale tento rozdíl v rychlosti se dá považovat za zanedbatelný [9].

- **Stavový proxy server** - jedná se o sofistikovanější typ SIP serveru, jenž informace nejen přenáší, ale zároveň ukládá záznamy o těchto transakcích, aby k nim mohl později přistupovat. To umožňuje například neúspěšný požadavek od nějakého klienta směrovat jinou částí sítě, aby úspěšně dorazil k příjemci [9].

1.2.3 Registrar server

Registrar server, lze přeložit jako registrační server, přijímá požadavky na registraci od uživatelských agentů. Registrace je jednoduše přiřazení uživatelského jména ke konkrétní síťové adrese a následné uložení této kombinace do databáze. Na obrázku 1.1 je zobrazena komunikace mezi koncovým bodem a registrar serverem. Nejdříve je odeslána zpráva REGISTER od uživatelského agenta směrem na registrar server, která zatím neobsahuje žádné informace o uživatelově hesle. Na tuto zprávu odpovídá server zprávou 401 UNAUTHORIZED, pokud je nutné, aby se uživatel autentizoval heslem. V opačném případě by server již odeslal zprávu 200 OK a potvrdil by tím registraci uživatele. V zobrazeném případě je ovšem vyžadována další interakce od uživatelského agenta, který musí vložit hlavičku do odesílané zprávy REGISTER s MD5 heshem svého hesla. Pokud je toto heslo správné, server odpovídá zprávou 200 OK, a tím je registrace úspěšně ukončena [7].



Obr. 1.1: Průběh komunikace mezi klientem a registrar serverem.

1.2.4 Redirect server

Do češtiny lze přeložit jako přesměrovávací server nebo server pro přesměrovávání. Jedná se o speciální typ user agent serveru, jenž generuje odpovědi typu 3XX, které obsahují adresu volaného. Tuto adresu získává z databáze vytvořené registrar serverem. Redirect server je využit, pokud se user agent snaží dovolat někam, kde se změnila adresa. V takovém případě dostane klient odpověď typu 3XX a obohacený aktualizovanými informacemi z přesměrovávacího serveru pak znovu požádá o volání s použitím nových informací o cíli. Tím se částečně odlehčí proxy serverům a zvýší se robustnost směrování hovorů [7, 10].

1.3 SIP Zprávy

Jak již bylo zmíněno v tomto textu, v SIP sítích existují různé prvky, které spolu komunikují. Tuto komunikaci vždy zahajuje user agent klient, jenž posílá požadavek, nebo žádost na nějaký server, který následně odesílá odpověď zpět klientovi.

1.3.1 SIP Požadavky

SIP požadavky jsou slova používané k navázání komunikace. Označují se jako metody a dělí se na základní, kterých existuje šest, a rozšiřující, jichž je osm. Mezi základní metody patří INVITE, ACK, CANCEL, OPTIONS, BYE a REGISTER. Do rozšiřujících metod spadají SUBSCRIBE, NOTIFY, PUBLISH, REFER, INFO, UPDATE, PRACK a MESSAGE [7]. V tomto textu budou podrobně popsány pouze základní metody.

Metoda INVITE

Metoda INVITE se používá k zahájení nebo modifikaci relace a může se také použít společně s protokolem SDP k výměně a dohodnutí parametrů v rámci navázané relace jako například použité kodeky, IP adresy nebo porty k příjmu následných dat pomocí protokolu RTP. V případě použití metody k modifikaci relace se metoda nazývá RE-INVITE. Příkladem takové modifikace je přidání videa do původní audio relace [7]. Ve výpise 1.1 je zobrazen příklad této metody.

Výpis 1.1: Příklad metody INVITE

```
INVITE sip:Bob@TMC.com SIP/2.0                                    1
Via: SIP/2.0/TLS client.ANC.com:5061;branch = z9hG4bK74bf9        2
Max-Forwards: 70                                                  3
From: Alice<sip:Alice@TTP.com>;tag = 1234567                      4
To: Bob<sip:Bob@TMC.com>                                          5
```

```
Call-ID: 12345601@192.168.2.1                                          6
CSeq: 1 INVITE                                                         7
Contact: <sip:Alice@client.ANC.com>                                    8
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY                9
Supported: replaces                                                    10
Content-Type: application/sdp                                          11
Content-Length: 175                                                    12
                                                                       13
v = 0                                                                  14
o = Alice 2890844526 2890844526 IN IP4 client.ANC.com                  15
s = Session SDP                                                        16
c = IN IP4 client.ANC.com                                              17
t = 3034423619 0                                                       18
m = audio 49170 RTP/AVP 0                                              19
a = rtpmap:0 PCMU/8000                                                 20
```

Metoda ACK

Metoda ACK je využíváná k potvrzování finálních odpovědí na metodu INVITE a lze ji použít jen a pouze v souvislosti s metodou INVITE. Co je to finální odpověď, bude probráno později v kapitole 1.3.2. Metoda ACK někdy může obsahovat charakteristiky dostupných prostředků a médií volajícího v podobě protokolu SDP, ale pouze tehdy, kdy tyto informace nebyly obsaženy v předchozí metodě INVITE [7].

Metoda CANCEL

Metoda CANCEL slouží ke zrušení nevyřízené žádosti. Může se jednat například o odmítnutí přicházejícího hovoru. Odeslání této metody by ovšem nemělo vliv na předchozí metody, na které již bylo zodpovězeno finální odpovědí [7].

Metoda OPTIONS

Metoda OPTIONS se používá k dotazování uživatelského agenta nebo proxy serveru na jeho možnosti a zjišťování jeho aktuální dostupnosti. V odpovědi na dotaz jsou uvedeny možnosti dotazované entity. Proxy server nikdy negeneruje požadavek OPTIONS [7].

Metoda BYE

Metoda BYE je používaná k ukončení navázané relace. Jedná se o požadavek, který může odeslat volající nebo volaný, aby ukončil relaci, nemůže být ovšem odeslán

proxy serverem. Požadavek BYE nelze odeslat na čekající INVITE nebo na nenavázanou relaci, k tomu slouží metoda CANCEL [7].

Metoda REGISTER

Tato metoda již byla zmíněna v kapitole 1.2.3 a zde budou doplněny pouze podstatné informace. Metoda slouží k registraci klienta a identifikaci jeho lokace v síti neboli adresy, to znamená, že metodu může odeslat pouze user agent klient. Může být však odeslán požadavek REGISTER nějakého klienta jménem jiného klienta. Jedná se o registraci třetí stranou. Metoda obsahuje identitu registrovaného klienta v podobě SIP URI, jež je obsažena v poli *To* hlavičky metody. Obsahuje taky lokaci klienta opět ve formě SIP URI, nacházející se v poli *Contact*. Potvrzení serverem je provedeno odpovědí typu 200 OK. Registrace je pouze dočasná a musí se v pravidelném intervalu obnovovat [7].

1.3.2 SIP odpovědi

SIP odpověď je zpráva generovaná serverem uživatelského agenta nebo SIP serverem odpovídající na požadavek generovaný klientem. Tyto odpovědi jsou rozřazeny do šesti kategorií. Odpovědi 1XX až 5XX byly převzaty z protokolu HTTP a odpovědi 6XX byly zavedeny speciálně pro protokol SIP. První kategorie odpovědí, tedy 1XX, jsou dočasné odpovědi, jež indikují průběh žádosti, nikoliv její vyřízení. To označují odpovědi finální, do kterých spadají všechny zbylé kategorie [7].

1XX - Dočasné odpovědi

V případě, kdy odpověď začíná číslicí jedna, se jedná o odpověď dočasnou. Jak již bylo řečeno, tyto odpovědi indikují průběh žádosti, to znamená, že byla žádost doručena k cílovému uživatelskému klientovi a ten aktuálně pracuje na jejím vyřízení. Nevypovídají ovšem o tom, zdali byl odeslaný požadavek úspěšně zpracován, či nikoliv [7].

Výpis dočasných odpovědí: 100 Trying, 180 Ringing, 181 Call is Being Forwarded, 182 Queued, 183 Session Progress, 199 Early Dialog Terminated [11].

2XX - Úspěšné odpovědi

Tato kategorie odpovědí začínajících na číslici dvě je určena pro indikaci, že byl požadavek úspěšně zpracován [7].

Výpis úspěšných odpovědí: 200 OK, 202 Accepted, 204 No Notification [11].

3XX - Přesměrovávací odpovědi

Odpovědi této kategorie jsou většinou odesílány serverem redirect, jenž informuje klienta o adrese, kde se nachází požadovaný server uživatelského agenta. Ten také může posílat odpovědi 3XX za předpokladu, že je na něm nastaveno přesměrovávání hovorů [7].

Výpis přesměrovávacích odpovědí: 300 Multiple Choices, 301 Moved Permanently, 302 Moved Temporarily, 305 Use Proxy, 380 Alternative Service [11].

4XX - Chyba na straně klienta

Odpovědi z této kategorie oznamují, že požadavek nemůže být splněn, jelikož se v něm nacházejí chyby, které jsou způsobené ze strany klienta. Detaily těchto chyb jsou popsány v odeslané odpovědi a na základě toho může klient požadavek upravit a znovu odeslat. Příkladem důvodu takové klientské chyby může být nepřítomnost povinné hlavičky v požadavku nebo překlep v URI [7].

V této kategorii je definováno čtyřicet osm různých odpovědí, proto zde nebudou vypsány. Čtenář se o nich může dozvědět více ze zdroje [11].

5XX - Chyba na straně serveru

Tato kategorie odpovědí se používá k označení, že požadavek nelze zpracovat z důvodu chyby na serveru. Jinými slovy, serveru se nepodařilo splnit zjevně platný požadavek. Při obdržení odpovědi z této kategorie může klient zkusit stejný požadavek odeslat na jinou lokaci v síti, protože požadavek jako takový není chybný [7].

Výpis odpovědí z kategorie chyba na straně serveru: 500 Internal Server Error, 501 Not Implemented, 502 Bad Gateway, 503 Service Unavailable, 504 Server Timeout, 505 Version Not Supported, 513 Message Too Large, 555 Push Notification Service Not Supported, 580 Precondition Failure [11].

6XX - Globální chyba

Odpovědi začínající na číslo šest odesílá server v případě, kdy je jisté, že požadavek selže všude v síti. Z tohoto důvodu by požadavek neměl být odeslán na jiné lokace [7].

Výpis odpovědí z kategorie globální chyba: 600 Busy Everywhere, 603 Decline, 604 Does Not Exist Anywhere, 606 Not Acceptable, 607 Unwanted, 608 Rejected [11].

1.4 Kodeky

Jak již bylo zmíněno v úvodu práce, pro fungování VoIP je nutné převést analogové zvukové signály na digitální data pro přenos po datových sítích. To zajišťuje PCM, tedy Pulse-Code Modulation, což lze přeložit jako pulzně kódová modulace. Po tomto převodu je nezbytné tato data ještě komprimovat. To zajišťují kodeky, což je zkratka pro kodér-dekodér. Převádí digitální signál do komprimované podoby a po přenesení po síti zpět do nekomprimovaného signálu pro přehrání. K porovnávání různých kodeků se používá metoda MOS (Mean Opinion Score). Měří kvalitu zvuku po kompresi a následné dekompresi a její hodnoty se pohybují v rozmezí od jedné do pěti, kdy jedna představuje nejhorší kompresi a pět tu nejlepší možnou, tedy bezeztrátovou. Celkově existuje velká řada kodeků, ale v tomto textu budou probrány jen kodeky, jež jsou spojeny právě s VoIP [4, 5].

1.4.1 G.711

Mezinárodní telekomunikační unie (ITU) standardizovala tento kodek v roce 1972 pro digitální telefonii, tudíž se jedná o relativně starý kodek. Má dvě varianty:

- A-law - používaný v Evropě a při mezinárodních telefonních spojích
- μ-law - používaný ve Spojených státech amerických a Japonsku

Využívá ke kódování a dekódování dat logaritmickou kompresi a pracuje v pásmu 64 Kb/s. Každý šestnácti bitový vzorek komprimuje na osmi bitový, dosahuje tedy kompresního poměru 1:2. V porovnání s ostatními kodeky nabízí lepší kvalitu hlasu, ale kvůli jeho kompresnímu poměru je vhodné ho využívat v lokálních sítích LAN z důvodu velké šířky pásma. Kodek lze volně používat bez licenčních poplatků a jeho MOS hodnota se pohybuje okolo 4,2. Jeho implementace je jednoduchá, a tudíž nevyžaduje velký procesní výkon [4, 5].

1.4.2 G.722

Jedná se o kodek vydaný v roce 1988 opět organizací ITU. Pracuje v pásmech 48, 56 nebo 64 Kb/s a ke kódování a dekódování dat používá adaptivní diferenciální pulzní kódovou modulaci (ADPCM). Data jsou vzorkována s frekvencí 16 kHz a i když se to nejeví jako příliš rychlé vzorkování, je dvakrát rychlejší než vzorkování u tradičního telefonního systému. Díky tomu dokáže produkovat zvuk ve vynikající kvalitě a čistotě. Kodek G.722 také představuje širokopásmový systém kódování zvuku, takže jej lze použít pro vysoce kvalitní řečové aplikace protokolu VoIP [5].

1.4.3 G.729

Kodek G.729 má nízké nároky na šířku pásma, pouze 8 Kb/s, a přitom nabízí dobrou kvalitu zvuku. Ten se kóduje po rámcích, kdy každý rámec má délku 10 ms a obsahuje osmdesát zvukových vzorků. Hodnota MOS se pohybuje okolo 4. Běžně využívaná varianta tohoto kodeku se nazývá G.729a, má nižší procesní nároky, a tudíž je efektivnější [4, 5].

Do nedávna byl tento kodek licencován, ale v lednu 2017 vypršela platnost patentu, takže je nyní volně dostupný [4].

1.4.4 G.723.1

Kodek G.723.1 existuje ve dvou variantách. Obě pracují se zvukovými rámci dlouhými 30 ms, to znamená, že mohou zpracovávat 240 vzorků. Algoritmy kódování se ovšem liší. První varianta má přenosovou rychlost 6,4 Kb/s spolu s MOS hodnotou 3,9. Druhá varianta má nižší přenosovou rychlost, 5,6 Kb/s s MOS hodnotou 3,7. Přestože nemá kodek nejlepší kvalitu zvuku, má nejlepší kompresní poměr proti ostatním popisovaným kodekům, a to 1:12. Z tohoto důvodu je vhodný pro použití u spojení s malou šířkou pásma [5].

2 Zátěžové testování

V současné době existuje obrovské množství systémů, které musí podporovat souběžný přístup tisíců až milionů uživatelů. Velký růst Internetu přispěl k využívání aplikací, jež musí fungovat svižně, nepřerušovaně a pokud možno co nejdéle. S tím jsou spojené problémy související s výkonem serverů, na kterých tyto aplikace běží. Náhlý přístup velkého množství uživatelů může způsobit selhání serveru, a tedy nepřístupnost aplikace a následné náklady na opravu mohou být vysoké. K předcházení takovým situacím se využívá zátěžové testování, které dokáže identifikovat úzká hrdla výkonu testované aplikace [12].

Zátěžové testování může být také využito k obraně proti útokům DoS nebo útokům DDoS (Distributed Denial of Service), kdy útok DoS je veden z jednoho místa proti útoku DDoS, který je veden z více míst najednou. Tyto typy útoků narušují nebo zcela vyřazují službu, na kterou útočí. To je obrovský problém, vzhledem k tomu, že většina podniků je závislá na síťových službách [12].

Jako příklad lze uvést společnost Google, u které se předpokládá, že během pětiminutového výpadku serverů dne 19. srpna 2013 přišla až o pět set čtyřicet pět tisíc dolarů [13]. Dalším příkladem může být firma Facebook, která podle deníku CNBC přišla během několika hodinového výpadku dne 4. října 2021 o téměř sto milionů dolarů a tržní hodnota společnosti během jediného dne spadla o více než čtyřicet sedm miliard dolarů [14].

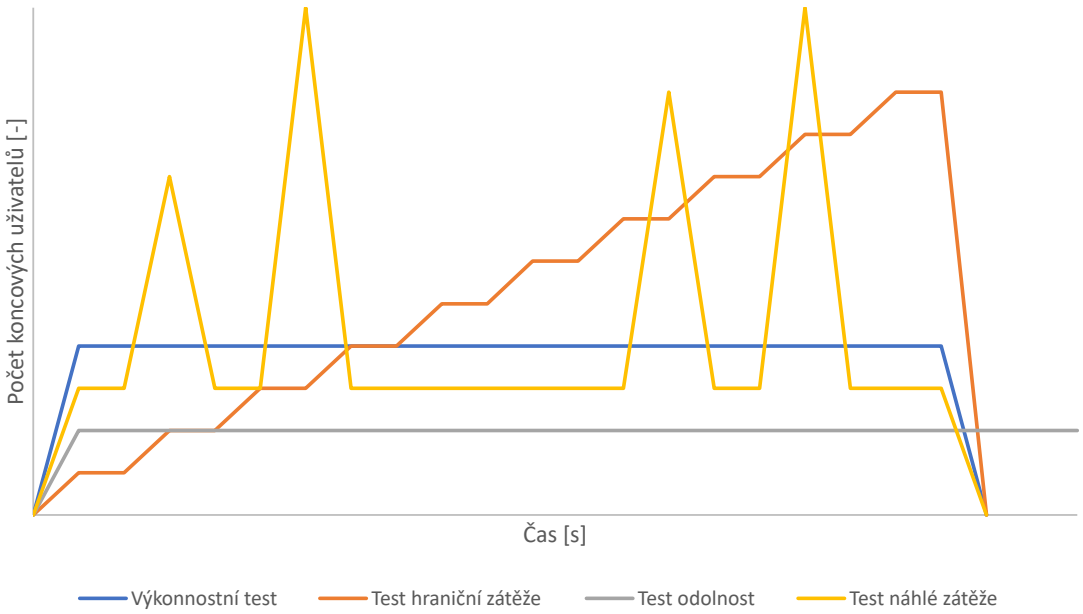
Je tedy zřejmé, že zátěžové testování je velice důležité. Lze ho rozdělit na několik typů. Každý typ se používá k zjištění různých parametrů a provádí se jinými způsoby.

2.1 Typy zátěžového testování

Existuje několik typů zátěžového testování. Všechny simulují síťový provoz, ale mají různé cíle a provoz generují s různou frekvencí a různým objemem. Na obrázku 2.1 jsou zobrazeny průběhy některých typů zátěžových testů, které budou probrány v následujících kapitolách.

2.1.1 Výkonnostní test

Výkonnostní test, anglicky označovaný jako load test, pomáhá pochopit chování systému při určité hodnotě zátěže. V procesu testování je simulován očekávaný počet souběžných uživatelů po určitou dobu, aby se ověřila předpokládaná doba odezvy a mohla se lokalizovat úzká hrdla. Tento typ testů pomáhá vývojářům určit, kolik uživatelů může aplikace současně obsloužit předtím, než bude spuštěna do produkčního prostředí [15].



Obr. 2.1: Průběhy čtyř typů zátěžových testů.

2.1.2 Test hraniční zátěže

U testu hraniční zátěže (stress test) je testovací systém vystaven vyššímu než očekávanému provoznímu zatížení, takže umožňuje zjistit, jak dobře systém funguje nad očekávanými kapacitními limity. Funguje tak, že postupně zatěžuje hardwarové zdroje s cílem určit potenciální bod zlomu aplikace. Může se jednat o zatěžovaní procesorů, paměti nebo pevných disků. Zatížení systému může také vést ke zpomalení výměny dat, nedostatku paměti, poškození dat nebo problémům se zabezpečením. Dalším faktorem, který mohou testy odhalit je, za jak dlouho se klíčové ukazatele výkonnosti vrátí na normální provozní úroveň po určité události. Testy hraniční zátěže mohou probíhat jak ve vývojovém, tak i v produkčním prostředí. Organizace často využívají tento druh zátěžového testování před důležitými událostmi, jako je například černý pátek pro internetové obchody, jež by přišli o obrovské částky peněz při výpadku během takové události [15].

2.1.3 Test odolnosti

Test odolnosti, anglicky soak test nebo endurance test, simuluje stálý počet koncových uživatelů v průběhu dlouhého časového období, aby se otestovala dlouhodobá udržitelnost systému. Během testu jsou sledovány klíčové ukazatele výkonnosti jako například využití paměti, propustnost a doba odezvy po trvalém používání a jsou porovnávány s hodnotami naměřenými na začátku testu [15].

2.1.4 Test náhlé zátěže

Test náhlé zátěže, označovaný v anglickém jazyce jako spike test, hodnotí výkon systému při náhlém a výrazném růstu počtu koncových uživatelů. Tyto testy pomáhají zjistit, zda systém zvládne náhlé a drastické zvýšení zátěže v krátkém časovém intervalu, a to opakovaně. Podobně jako u testů hraniční zátěže jsou tyto testy prováděny před nějakou očekávanou důležitou událostí, kdy je třeba zajistit spolehlivost fungování aplikace, protože je vystavena vyššímu než běžnému objemu provozu [15].

2.1.5 Test škálovatelnosti

Test škálovatelnosti, anglicky scalability test, měří výkonnost systému při zvyšování, nebo snižování počtu uživatelských požadavků. Účelem tohoto testování je zajistit, aby systém zvládl předpokládaný nárůst koncových uživatelů. Jinými slovy testuje schopnost systému vyhovět rostoucím potřebám. Měří se také, v jakém okamžiku se aplikace přestane škálovat, a je nutné určit příčinu, která za tím stojí [15].

2.1.6 Test objemu dat

Test objemu dat, anglicky nazýván capacity test, slouží k ověřování chování aplikace v momentě, kdy se zvyšuje objem dat. Je analyzována například rychlost volání určitých hodnot při narůstajícím objemu dat v databázi [15].

2.2 Metriky zátěžového testování

V předchozích kapitolách byl zmíněn pojem klíčové ukazatele výkonnosti. Jedná se o metriky testování, jež pomáhají vyhodnotit aktuální výkonnost testovaného systému. Mezi tyto metriky se například řadí:

- **Propustnost** - používá se ke kontrole počtu požadavků, které bude systém schopen zpracovat za jednotku času. Každý plán testů zpravidla obsahuje cíl propustnosti a čím realističtější je, tím přesnějšího výsledku je dosaženo.
- **Paměť** - množství fyzické paměti, které je dostupné procesoru.
- **Doba odezvy** - doba, která uplyne mezi požadavkem zadaným uživatelem a začátkem odpovědi systému na tento požadavek.
- **Šířka pásma** - objem dat za sekundu využitých síťovým rozhraním.
- **Přerušení procesoru za sekundu** - průměrný počet hardwarových přerušení, která procesor přijímá a zpracovává každou sekundu [15].

2.3 Proces zátěžového testování

Metodika zátěžového testování se může značně lišit, ale cíl testování zůstává stejný. Může se jednat o potvrzení, že systém splňuje určitá předem definovaná výkonnostní kritéria, může porovnat výkonnost dvou systémů nebo může identifikovat části systému, které snižují jeho výkonnost, takzvaná úzká hrdla. Níže je popsán obecný proces, jak zátěžové testování provádět.

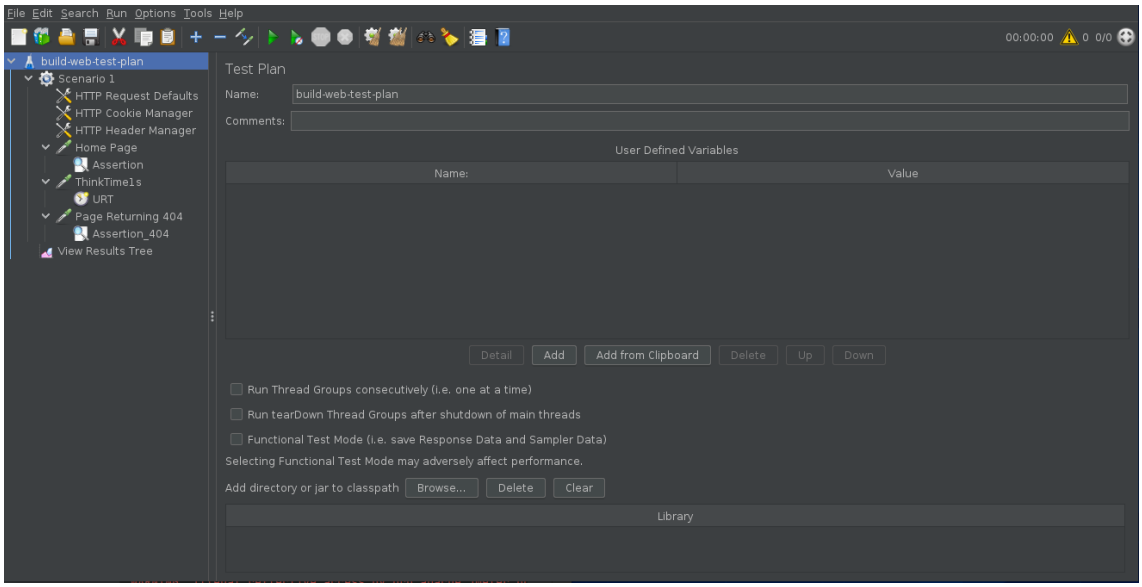
1. **Identifikace testovacího prostředí** - je nutné analyzovat testovací i produkční prostředí a seznámit se s testovacími nástroji, jež jsou k dispozici. To je nezbytné provést před zahájením samotného testování, jelikož se tyto nabyté znalosti dají využít k vytvoření efektivnějších testů nebo se mohou identifikovat problémy, na které by se mohlo narazit až při samotném testování.
2. **Ustanovení přijatelných kritérií** - před navržením zátěžového testu je nezbytné určit si cíle a omezení týkajících se metrik testu a identifikovat kritéria úspěšnosti testování. Častým způsobem stanovování těchto parametrů je nalezení podobného testovacího subjektu, který již prošel procesem zátěžového testování a inspirovat se právě odtud.
3. **Plánování a navrhování zátěžových testů** - jedná se o určení, jak se může lišit používání aplikace mezi koncovými uživateli a definování klíčových scénářů pro testování všech možných případů použití. V zátěžovém testu se simulují různí koncoví uživatelé, konkrétní data, jež uživatelé mohou posílat do aplikace a musí se nastínit, jaké metriky budou během testu shromažďovány.
4. **Konfigurace testovacího prostředí** - příprava testovacího prostředí před spuštěním samotného zátěžového testu.
5. **Implementace návrhu testu** - vytvoření zátěžového testu podle jeho návrhu z předchozího kroku.
6. **Provádění testu** - realizace a monitorování samotného testu.
7. **Analýza, ladění a opakované testování** - po provedení testu je nutná jeho analýza, aby bylo možné následné doladění a opakované provedení testu, což může odhalit zlepšení, ale také zhoršení výkonu [13].

2.4 Analýza dostupných nástrojů

Jelikož obor zátěžového testování není nová převratná technologie, ale už je na světě nějakou dobu, existuje několik dostupných nástrojů, jež se k zátěžovému testování využívají. Nejrozšířenějším typem nástrojů jsou ty, které slouží pro testování webových služeb. Pomocí nich lze testovat internetové stránky, nacházet slabiny v implementaci kódu a následně tyto slabiny opravovat [12]. V této kapitole budou rozebrány nejznámější nástroje.

2.4.1 Apache JMeter

Apache JMeter je otevřený program kompletně napsaný v programovacím jazyce Java a existuje ve verzi s grafickým rozhraním, nebo bez něj. Je určen k analýze a měření výkonnosti a funkčního chování webových aplikací, ale podporuje i řadu dalších služeb a díky tomu, že je modulární, do něj lze přidávat i vlastní naprogramované moduly, což je hlavním cílem této práce. Kromě webových služeb JMeter ve výchozím nastavení také podporuje například mailové nebo adresářové služby. Podporuje testování jak statické, tak i dynamické, což umožňuje uživateli nahrávat svoje úkony pro co nejpřesnější simulaci lidského používání testované aplikace. Program byl vyvinut Stefanem Mazzocchim ze společnosti Apache Software Foundation. Na obrázku 2.2 je zobrazeno uživatelské rozhraní programu s předpřipraveným testovacím plánem pro webovou aplikaci [16].



Obr. 2.2: Grafické rozhraní programu JMeter.

Prvky testovacího plánu

V této kapitole jsou popsány základní prvky testovacího plánu v programu JMeter. Z důvodu obtížného překladu těchto prvků do češtiny je nutné tyto překlady brát s rezervou.

V každém testovacím plánu musí být alespoň jedna skupina vláken a do této skupiny se následně vkládají další prvky jako například samplery nebo listenery. Program obsahuje prvky:

- **Test Plan** (testovací plán) - hlavní prvek, který slouží pro vkládání všech ostatních komponent.

- **Thread Group** (skupina vláken) - jedná se o reprezentaci uživatelů.
- **Sampler** (vzorkovník) - je to základní element simulující požadavek na testovaný systém v podobě určitého protokolu.
- **Logic Controller** (logický regulátor) - slouží ke stanovení v jakém pořadí budou vykonány samplery.
- **Config Element** (konfigurační prvek) - nastavuje a udržuje proměnné pro další použití v samplerech.
- **Listener** (posluchač) - ukládá výsledky testování a zobrazuje je ve vizuální podobě, například ve formě tabulky, grafu nebo stromového diagramu.
- **Timer** (časovač) - vkládá časové prodlevy mezi odesíláním požadavků daného sampleru. Bez timeru JMeter odesílá požadavky nejvyšší možnou rychlostí, což může způsobit zahlcení serveru.
- **Assertion** (tvrzení) - slouží k validaci funkčnosti testování. Porovnává skutečnou a očekávanou odpověď ze serveru, a pokud se odpovědi liší, požadavek je vyhodnocen jako neúspěšný.
- **Pre Processor** (pre zpracovatel) - umožňuje provádět operace ještě před spuštěním sampleru.
- **Post Processor** (post zpracovatel) - vykonává úkony po ukončení funguvání sampleru. Může být využit například k filtrování různých informací z odpovědí serveru [17].

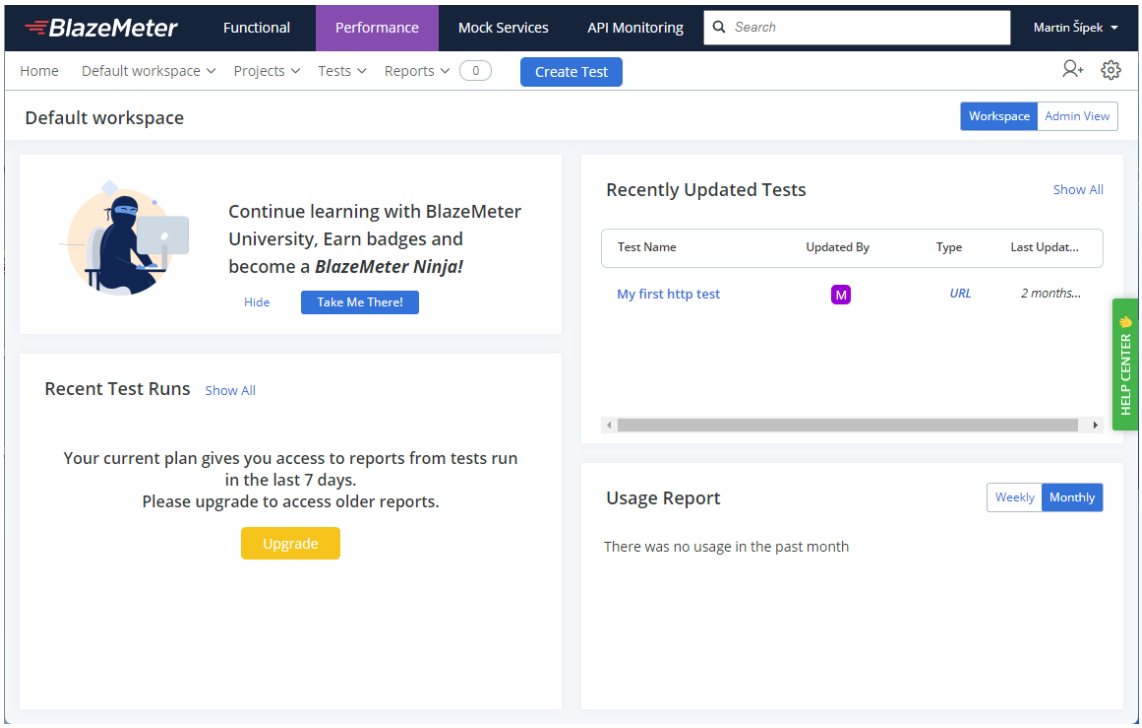
Časové provedení výše zmíněných prvků v případě, kdy by byly v testovacím plánu využity všechny, je následující [17]:

1. Config Element
2. Pre Processor
3. Timer
4. Sampler
5. Post Processor
6. Assertion
7. Listener

2.4.2 BlazeMeter

BlazeMeter je jeden z nejznámějších nástrojů pro zátěžové testování. Jedná se o cloudovou službu a je zcela kompatibilní s programem JMeter, takže se testovací plány mezi těmito systémy dají jednoduše importovat. Mimo funkcionalitu JMeteru program BlazeMeter nabízí celou řadu dalších funkcí jako například Taurus, což je určeno pro průběžné testování. Dalšími příklady mohou být URL/API test, Webdriver test nebo Multi test [12].

Jedná se o placenou službu, ale existuje i bezplatná varianta, kdy má uživatel možnost testovat až deset hodin s až padesáti různými uživateli měsíčně zcela zdarma [18]. Kolegovi Petru Křížovi se podařilo získat od podpory BlazeMeteru premium verzi na čtrnáct dní zdarma k edukativním účelům. Bohužel tento systém nepodporuje protokol SIP, který je nezbytný pro vypracování této práce, proto tento premium účet nebyl využit. Na obrázku 2.3 je zobrazena úvodní stránka BlazeMeteru po přihlášení uživatele.



Obr. 2.3: Grafické rozhraní programu BlazeMeter.

3 Vývoj vlastního modulu

Hlavním cílem diplomové práce je implementace modulu do programu JMeter, který umožní testování základních SIP metod a to konkrétně REGISTER, INVITE a MESSAGE. Tato kapitola bude věnována dané problematice a bude v ní popsáno, jakým způsobem bylo dosaženo požadovaného cíle. Před samotným vývojem a implementací modulu je potřeba program JMeter importovat do vývojového prostředí.

3.1 Import do vývojového prostředí

Jak již bylo zmíněno v kapitole 2.4.1, JMeter je vyvinut kompletně v programovacím jazyce Java, proto bylo vybráno vývojové prostředí IntelliJ IDEA od společnosti JetBrains nainstalované na operačním systému CentOS 8 běžící ve virtuálním prostředí VMware. To bylo upřednostněno před vývojovým prostředím Eclipse kvůli mnohem jednoduššímu importu programu JMeter do prostředí a kvůli osobním preferencím. Samotný program JMeter lze stáhnout ze zdroje [19]. Pro import do vývojového prostředí je potřeba stáhnout program v podobě zdrojových souborů, nikoliv binárních. V této práci je použita verze programu 5.4.3. a Java verze 18.

Pro import programu do vývojového prostředí bylo využito návodu od samotných vývojářů v souboru CONTRIBUTING.md nacházející se ve stažené složce JMeteru. Je nutné spustit vývojové prostředí IntelliJ IDEA a z horního panelu vybrat *File > Open*. Otevře se dialogové okno, v němž je nutné vybrat soubor *build.gradle.kts*, jenž se nachází také ve zdrojové složce programu JMeter. V dalším dialogovém okně se musí zvolit, že se má soubor otevřít jako projekt, tedy tlačítko *Open as Project*. Poslední dialogové okno se zeptá, zdali je tento soubor důvěryhodný a jelikož je, může se stisknout tlačítko *Trust Project*, a tím se spustí import programu do vývojového prostředí. Tento proces může trvat delší dobu, záleží na výpočetních prostředcích počítače. Pokrok lze zobrazit v pravé dolní části okna programu IntelliJ klinutím na text *Show all*, čímž se zobrazí aktuálně prováděné procesy.

Po úspěšném importu je nutné uskutečnit build programu, tedy kompilaci všech jeho částí a instalaci. To se provede následovně:

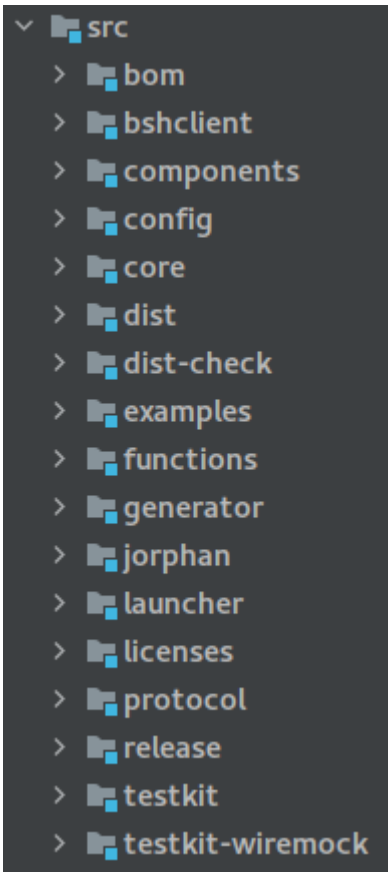
1. V pravé horní části programu kliknout na tlačítko *Add Configuration...*.
2. V nově otevřeném okně kliknout na znaménko „+“ v levé horní části, čímž se přidá nová konfigurace pro spouštění programu.
3. Po otevření nabídky konfigurací je nutné vybrat možnost *Gradle*, čímž se zobrazí různá nastavení a parametry.
4. V textovém poli *Tasks and arguments* je potřeba napsat *runGui*, což je funkce, která spouští JMeter program s grafickým rozhraním.

5. Všechny ostatní možnosti mohou být ponechány ve výchozím stavu a provedené změny lze potvrdit tlačítkem *OK*.
6. Posledním krokem je samotné spuštění kompilace stisknutím zeleného trojúhelníku vpravo od nově vytvořené konfigurace, nebo vybrat z hlavního panelu *Run* > *Run...* a v dialogovém okně zvolit nově vytvořenou konfiguraci.

Pokud vše proběhlo v pořádku, po několika sekundách by se měl spustit program JMeter i s grafickým rozhraním a import do vývojového prostředí byl úspěšný.

3.2 Struktura programu JMeter

JMeter je strukturovaný do složek, jež se dají logicky rozdělit do dvou skupin. První skupina se dá nazvat jako jádro programu, do kterého by uživatel neměl zasahovat, aby program fungoval tak, jak má, a spadají sem všechny složky vyjma *src/protocol* a *src/examples*. Tyto složky se řadí do druhé skupiny, což jsou funkce programu, jež ke své funkci využívají abstraktní třídy z jádra. Díky této organizaci je program modulární a velice jednoduše se dá přidávat další funkcionalita. Všechny složky jsou zobrazeny na obrázku 3.1.



Obr. 3.1: Struktura programu JMeter.

Ve složce *src/examples* vývojáři poskytli ukázku, jak vytvořit jednoduchý modul. Každý modul je složen minimálně ze dvou tříd, ze třídy *Sampler* a *SamplerGui*. Třída *Sampler* je potomek třídy *AbstractSampler* a stejným způsobem funguje *SamplerGui*, která dědí ze třídy *AbstractSamplerGui*. Díky tomu se lze při vývoji nových modulů zabývat pouze funkcionalitou samotného modulu a integrita celého programu je tím zajištěna.

Třída *Sampler* má za úkol obstarávat hlavní funkčnost modulu. Udržuje proměnné, objekty nebo metody, které provádějí dané úkony za pomoci abstraktních metod z jádra programu. Třída *SamplerGui* se stará o konstrukci uživatelského rozhraní modulu a předává vstupní parametry od uživatele do programu.

K vložení nového modulu do programu JMeter je nutné modul zkompilovat, čímž se vytvoří soubor s koncovkou *.jar*. Tento soubor se potom musí vložit do složky *lib/ext* a při dalším spuštění programu JMeter bude modul již dostupný k použití.

3.3 Realizace modulu

Když už je JMeter naimportovaný do vývojového prostředí a je známa struktura programu stejně tak jako vývoj nových modulů, je vhodná doba na samotnou realizaci vlastního modulu.

3.3.1 Nastavení projektu Maven

Modul je vyvinut v samostatném projektu Maven, což je doporučováno, ale nikoliv vyžadováno. Maven ulehčuje správu knihoven, jež jsou potřebné pro správný běh programu, a také lze jednoduše nastavit build proces daného programu. Definice projektu se řídí konvencí Mavenu, a je tedy definována následovně:

- groupId: cz.vut.fekt.utko.sipsampler,
- artifactId: jmeter-sip-plugin,
- version: 1.0.

Nastavení projektu Maven je definováno v souboru *pom.xml*, který se nachází ve zdrojové složce projektu. Je potřeba provést několik úprav. Program je nutné exportovat do souboru typu *.jar*, aby bylo možné následné vložení do programu JMeter. Tato úprava je zobrazena ve výpise 3.1 na řádku tři.

Výpis 3.1: Informace o projektu v souboru pom.xml.

```
<groupId>cz.vut.fekt.utko.sipsampler</groupId>                    1
<artifactId>jmeter-sip-plugin</artifactId>                        2
<packaging>jar</packaging>                                        3
<version>1.0</version>                                            4
```

Následuje přidání pluginů do projektu. Pro kompilaci projektu je zapotřebí plugin *maven-compiler-plugin*. Přidání pluginu je zobrazeno ve výpise 3.2.

Výpis 3.2: Přidání pluginů v souboru pom.xml.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

V neposlední řadě jsou definovány knihovny, které jsou použity při vývoji vlastního modulu. Samotnou kompatibilitu s programem JMeter zajišťuje knihovna *ApacheJMeter_core* ve verzi 5.4.3. Dále jsou přidány knihovny pro práci s protokolem SIP. Jedná se o knihovny *jain-sip-api*, *jain-sip-ri* a *jain-sip-sctp*. Ty ke správné funkcionalitě vyžadují knihovnu *log4j*. V projektu budou využity také knihovny *awaitility* a *hamcrest*. Přídání všech těchto knihoven je zobrazeno ve výpise 3.3.

Výpis 3.3: Přidání knihoven v souboru pom.xml.

```
<dependencies>
    <dependency>
        <groupId>org.apache.jmeter</groupId>
        <artifactId>ApacheJMeter_core</artifactId>
        <version>5.4.3</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <dependency>
        <groupId>javax.sip</groupId>
        <artifactId>jain-sip-api</artifactId>
        <version>1.2.1.4</version>
    </dependency>
```

```
    <dependency>                                                    17
        <groupId>javax.sip</groupId>                                18
        <artifactId>jain-sip-ri</artifactId>                        19
        <version>1.3.0-91</version>                                 20
    </dependency>                                                   21
    <dependency>                                                    22
        <groupId>javax.sip</groupId>                                23
        <artifactId>jain-sip-sctp</artifactId>                      24
        <version>1.2.164</version>                                  25
    </dependency>                                                   26
    <dependency>                                                    27
        <groupId>org.awaitility</groupId>                           28
        <artifactId>awaitility</artifactId>                         29
        <version>4.2.0</version>                                    30
    </dependency>                                                   31
    <dependency>                                                    32
        <groupId>org.hamcrest</groupId>                             33
        <artifactId>hamcrest-core</artifactId>                      34
        <version>2.2</version>                                      35
    </dependency>                                                   36
</dependencies>                                                     37
```

Po úpravách v souboru *pom.xml* je projekt nastaven a připraven k vývoji vlastního modulu.

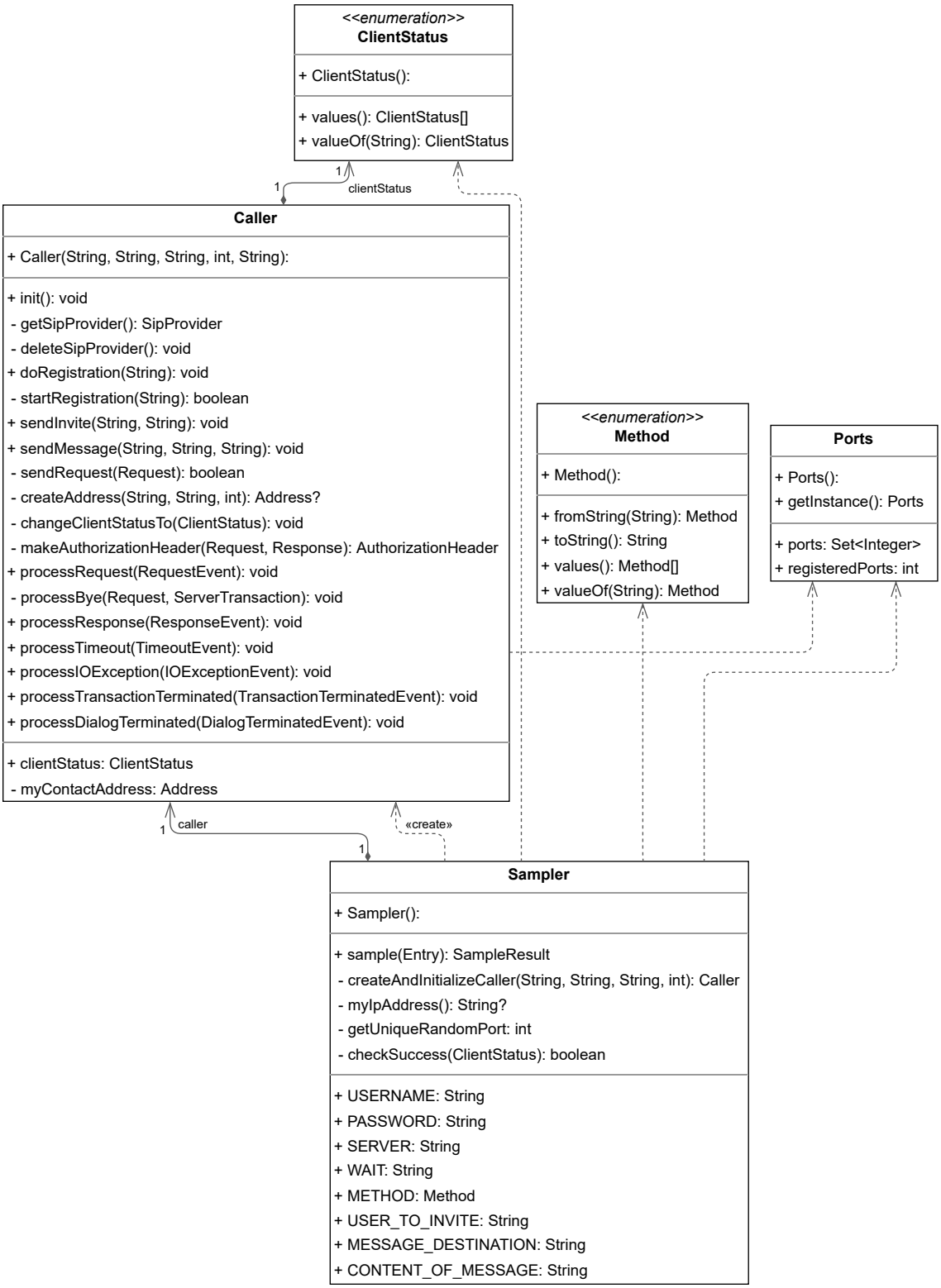
3.3.2 Struktura projektu

Projekt je rozdělen do dvou různých balíčků. Balíček *cz.vut.fekt.utko.sipsampler* se stará o logické funkce modulu a samotnou funkcionalitu. Skládá se ze tří různých tříd a dvou výčtových typů, které budou popsány později v textu. Balíček *cz.vut.fekt.utko.sipsampler.gui* obstarává grafické uživatelské rozhraní a do logiky programu nezasahuje. Obsahuje tři třídy.

Logická struktura

Jak již bylo zmíněno v předchozí sekci, logickou strukturu obstarává balíček *cz.vut.fekt.utko.sipsampler*. Obsahuje třídy *Sampler*, *Caller*, *Ports* a výčtové typy *Method* a *ClientStatus*. UML diagram popisující tento balíček je zobrazen na obrázku 3.2.

Třída **Sampler** dědí ze třídy *AbstractSampler*. Jedná se v podstatě o kontrolní centrum celého modulu. Ve třídě jsou definovány proměnné obsahující uživatelské



Obr. 3.2: UML diagram balíčku cz.vut.fekt.utko.sipsampler.

jméno a heslo, IP adresu SIP serveru nebo délku čekání na úspěšnou response zprávu. K těmto proměnným jsou vytvořeny gettery a settery k získávání a modifikaci informací obsažených v proměnných.

Metoda *sample(Entry)* slouží ke komunikaci s testovanou službou. Nejdříve je vytvořena instance *SampleResult*, což je třída, kterou JMeter používá k ukládání všech měření konkrétního vzorku a později lze zobrazit vizualizaci dat v nějakém listeneru. Poté je vytvořena instance třídy *Caller*, jenž se stará o problematiku protokolu SIP a bude probrána později v textu. Následuje *switch* blok, který rozhoduje jaká část kódu bude vykonána na základě testované metody, která byla vybrána uživatelem. Může se jednat buď o metodu REGISTER, INVITE nebo MESSAGE. Tyto metody jsou uloženy ve výčtovém typu **Method**, který také obsahuje metody *fromString(String)* a *toString()*. Tyto metody převádí objekty z typu *String* do typu *Method* a naopak. Metoda *sample(Entry)* vrací instanci *SampleResult* obsahující informace o měření.

Další definovanou metodou je *createAndInitializeCaller(String, String, String, int)*. Jak již z názvu vypovídá, metoda vytváří instanci třídy *Caller* a následovně volá metodu *init()*, jež inicializuje určité proměnné, které jsou nezbytné pro správné fungování třídy *Caller*. Vytvořená instance je nakonec vrácena.

Metoda *myIpAddress()* je také samovysvětlující. Zajišťuje získání IP adresy na rozhraní, které bude využito při komunikaci s testovaným SIP serverem. Je volána pouze při vytváření instance třídy *Caller*.

Metoda *getUniqueRandomPort()* získává unikátní UDP port, jenž je potřeba při vytváření instance *Caller*. Každý uživatel musí mít port unikátní, jelikož se pomocí něj v pozdější části kódu vytváří SIP listener, díky kterému uživatelé mohou přijímat odpovědi od SIP serveru. Proměnná obsahující již použité porty musí být sdílena mezi všemi vlákny, aby byla zaručena unikátnost. Proto existuje třída **Ports**, která tuto funkcionalitu zajišťuje. Obsahuje metodu *getInstance()*, jež vytváří instanci třídy *Ports*, ale pouze pokud je proměnná obsahující tuto instanci rovna *null*, což znamená, že instance ještě nebyla vytvořena. Pokud se *null* nerovná, je vrácena hodnota proměnné, tedy dříve vytvořená instance této třídy. Metoda je *synchronized*, což znamená, že vlákna do této metody mohou přistupovat pouze jednotlivě. To je důležité, protože pokud by byl umožněn přístup více vláknům zároveň, data by mohla být nekonzistentní a byla by narušena vlastnost unikátnosti portů. Důležitou proměnnou této třídy je *ports* typu *Set<Integer>*, ve které jsou uloženy již využité UDP porty na vytvoření SIP listenerů. Metoda *getUniqueRandomPort()* tedy získává instanci třídy *Ports* a následně je vygenerováno náhodné číslo v rozmezí od 49152 až 65535, což je rozsah dynamických UDP/TCP portů. Toto vygenerované číslo je porovnáno s hodnotami obsaženými v proměnné ports, a pokud číslo proměnná neobsahuje, číslo se nejdříve do proměnné přidá a následně je metodou

vráceno. V opačném případě je metoda rekurzivně zavolána a proces je zopakován, dokud se nenajde port, jenž se dá použít pro vytvoření SIP listeneru.

Poslední definovanou metodou ve třídě *Caller* je metoda s názvem *checkSuccess(ClientStatus, Duration)*. Jedná se o metodu, která čeká uživatelem definovanou dobu, než se změní status komunikace mezi modulem a SIP serverem na status, který byl metodě předán jako parametr. Jsou využity knihovny *awaitility* a *hamcrest*. Knihovna *awaitility* dokáže pozastavit chod vlákna na dobu, než je splněna nějaká podmínka. V tomto případě je podmínka splněna při změně statusu komunikace na požadovaný status. Knihovna také dokáže nastavit maximální dobu čekání, než je podmínka splněna. Doba je předána jako druhý parametr metody. Podmínka je vyhodnocována pomocí knihovny *hamcrest*. V případě, kdy je status komunikace změněn na požadovaný v předem stanovené době, metoda vrací *boolean true*, v opačném případě je vrácen *boolean false*. Všechny možné statusy komunikace jsou uloženy ve výčtovém typu **ClientStatus**. Konkrétně se jedná o DISCONNECTED, REGISTERING, REGISTERED, TRYING, RINGING, CONNECTED a MESSAGE_ACCEPTED. Při testování registrace uživatele na server se čeká na změnu statusu REGISTERED, při testování hovoru je očekáván status CONNECTED a při testování posílání zpráv je zapotřebí změna statusu na MESSAGE_ACCEPTED.

Třída **Caller** implementuje rozhraní *SipListener* a stará se o chod veškeré problematiky spojené s protokolem SIP. Představuje uživatelského agenta. Jsou zde definovány například proměnné obsahující informace o uživateli nebo instance rozhraní a tříd ulehčující práci s protokolem SIP. Jedná se například o rozhraní *SipProvider*, jež se používá pro odesílání SIP zpráv nebo dokáže vytvořit SIP listener, jenž přijímá SIP odpovědi od serveru. Dalšími příklady mohou být instance tříd *AddressFactoryImpl*, *MessageFactoryImpl* and *HeaderFactoryImpl*, což jsou pomocné třídy, které ulehčují vytváření různých SIP objektů, jako například hlavičky zpráv nebo SIP adresy.

Konstruktor třídy přijímá jako parametry informace o uživateli a ty si ukládá do globálních proměnných. Nastavuje také status komunikace na DISCONNECTED.

O inicializaci dříve zmíněných proměnných obsahující pomocné rozhraní a třídy ulehčující práci s protokolem SIP se stará metoda *init()*. Při vytváření instance rozhraní *SipProvider* je volána metoda *getSipProvider()*. Tato metoda nejdříve vytváří objekt *ListeningPoint*, který obsahuje informace o IP adrese, portu a transportním protokolu. Pomocí tohoto objektu lze následně vytvořit instanci třídy *SipProvider* a této instanci přiřadit objekt *SipListener*, což je samotná třída *Caller*. Instance třídy *SipProvider* je na konci metody *getSipProvider()* vrácena.

Metoda s funkcionalitou odstranění objektu *SipProvider* se nazývá *deleteSipProvider()*. Nejdříve je odstraněn objekt *ListeningPoint*, následně se musí odstranit

objekt *SipListener* a až poté lze odstranit *SipProvider*. V žádném jiném pořadí toto odstranění knihovna *JAIN-SIP* neumožňuje. Na samotném konci metody je ještě odstraněn port, jenž byl použit pro vytvoření objektu *ListeningPoint*, ze seznamu použitých portů, aby mohl být znovu využit jiným vláknem, které se teprve chystá na vytvoření *SipProvider*. Tato metoda je volána při obdržení finální SIP odpovědi od serveru. Výjimkou je testování metody INVITE, kdy je tato metoda zavolána až po přijetí zprávy BYE, jež slouží pro ukončení spojení mezi klientem a serverem.

Metoda *doRegistration(String)* je jednoduchá metoda, která volá metodu *startRegistration(String)* a pokud vrátí volaná metoda *boolean true*, je nastaven status komunikace na REGISTERING. V opačném případě se neprovede nic a status zůstává nastavený jako DISCONNECTED. Metoda přijímá jako parametr IP adresu serveru, na který je odesílána SIP zpráva.

Metoda *startRegistration(String)* také přijímá jako parametr IP adresu SIP serveru a zajišťuje vytvoření nezbytných polí, ze kterých lze následně poskládat hlavičku zprávy REGISTER. Jedná se o:

- URI – kontaktní informace dalšího prvku na trase SIP zprávy.
- Call-ID – jedinečný identifikátor SIP komunikace.
- CSeq – sekvenční číslo, které se zvyšuje při každém požadavku.
- From – pole označující původce zprávy.
- To – pole označující konečného příjemce zprávy.
- Via – používá se k záznamu trasy SIP požadavku.
- Max-Forwards – omezuje počet, kolikrát lze požadavek předat mezi zařízeními.
- Expires – označuje časový interval, ve kterém je obsah zprávy platný. Při použití u zprávy REGISTER označuje jak dlouho bude uživatel registrovaný na ústředně.
- Contact – informuje o adrese původce požadavku.

Z těchto polí je vytvořena hlavička zprávy REGISTER, jež je předána jako parametr metodě *sendRequest(Request)*, jenž se stará o odeslání zprávy na SIP server. Tato metoda bude podrobněji popsána později v textu.

Metoda *sendInvite(String, String)* je velice podobná předchozí metodě *startRegistration(String)*. Vytváří pole, ze kterých je následně vytvořena hlavička zprávy INVITE. Pole jsou stejná, ale místo pole *Expires* je přidáno do požadavku pole *Allow*, jenž uvádí SIP metody, které může volající používat během hovoru. Dále je do požadavku přidáno tělo zprávy v podobě protokolu SDP, jenž je zobrazeno ve výpise 3.4. S přidáním těla je nutné nastavit pole hlavičky *Content-Type*, jenž určuje, jak je tělo zprávy naformátováno. Metoda kromě IP adresy ústředny přijímá jako parametr také uživatelské jméno uživatele, se kterým bude navázáno spojení.

Výpis 3.4: SDP tělo metody INVITE.

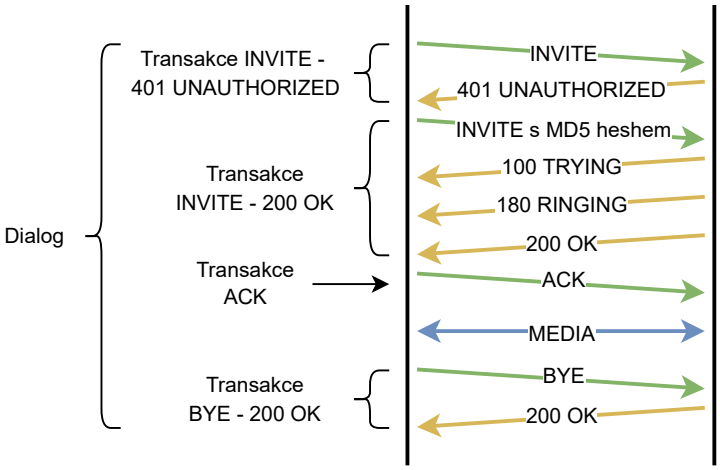
```
v=0                                                                    1
o=SIP_sampler <ID> <verze> IN IP4 <IP adresa uživatele>                2
s=SIP sampler                                                          3
c=IN IP4 <IP adresa uživatele>                                         4
t=0 0                                                                  5
m=audio 8000 RTP/AVP 9 98 0 8 3                                        6
a=rtpmap:98 telephone-event/8000                                       7
a=fmtp:98 0-16                                                         8
```

Po úspěšném vytvoření zprávy INVITE je zpráva předána jako parametr metodě *sendRequest(Request)* k odeslání.

Metoda, která se stará o vytváření polí hlavičky zprávy MESSAGE se nazývá *sendMessage(String, String, String)*. Opět jsou vytvořena výše zmíněná pole, kromě polí *Expires* a *Allow*. Z těchto polí je vytvořena SIP zpráva MESSAGE. Jedná se ovšem o zprávu, která má i tělo, proto se musí nastavit pole *Content-Type*. V tomto případě se jedná o typ těla *text/plain*. Tělo obsahuje uživatelem specifikovanou zprávu, kterou chce odeslat. Ta je přijímána jako parametr metody. Na konci metody je vytvořená zpráva MESSAGE předána jako parametr metodě *sendRequest(Request)*.

Již několikrát zmiňovaná metoda *sendRequest(Request)* odesílá vytvořené požadavky na SIP server. Nejdříve je zkontrolováno, zdali byla vytvořena instance rozhraní *SipProvider*, což je nutné pro vytvoření instance rozhraní *ClientTransaction*, jež představuje SIP transakci. Jedná se o výměnu zpráv mezi dvěma SIP prvky, která začíná požadavkem a končí finální odpovědí. Například během ukončení SIP relace jeden uživatel ukončí hovor odesláním požadavku BYE a druhá strana odpoví odpovědí 200 OK. Tato výměna zpráv se nazývá transakce. Pokud je podmínka splněna, je vytvořena nová transakce, díky níž je odeslán požadavek. Po odeslání je z této transakce získán dialog, což je kompletní výměna SIP zpráv mezi dvěma prvky. To znamená, že transakce jsou součástí dialogu. V případě navázání SIP relace dialog začíná transakcí INVITE – 401 UNAUTHORIZED, pokračuje INVITE – 200 OK, na to je reagováno ACK, což je speciální typ transakce, která nemusí končit finální odpovědí, a končí transakcí BYE – 200 OK. Tato situace je zobrazena na obrázku 3.3. Dialog je dobré si uložit do proměnné, jelikož se díky němu lehčeji reaguje na odpovědi od SIP serveru. Například odeslání zprávy ACK při obdržení odpovědi 200 OK na požadavek INVITE. Po získání dialogu vrací metoda *boolean true*, pokud vše proběhlo bez chyb. V opačném případě je vrácen *boolean false*.

Metoda *createAddress(String, String, int)* slouží k vytváření SIP adres. Jako parametry přijímá uživatelské jméno, IP adresu a port a návratová hodnota je typu *Address*. Příklad SIP adresy je uveden ve výpise 3.5.



Obr. 3.3: Grafické rozdělení transakce a dialogu.

Výpis 3.5: Příklad SIP adresy

```
sip:10@192.168.89.222:5060;transport=udp
```

Metoda sloužící ke změně statusu komunikace, což je důležité při vyhodnocování výsledků testování ve třídě *Sampler*, se nazývá *changeClientStatusTo(ClientStatus)*. Jako jediný parametr přijímá status, na který chceme změnit ten stávající.

Metoda *makeAuthorizationHeader(Request, Response)* je volána v moment, kdy je na požadavek obdržena odpověď 401 UNAUTHORIZED. V takovém případě se musí odeslat stejný požadavek ještě jednou s MD5 výzvou. K vytvoření pole s takovou výzvou slouží právě tato metoda. Jako parametry přijímá požadavek, na který bylo odpovídáno a samotnou odpověď. Vytvoření požadované MD5 výzvy lze rozdělit do tří kroků.

1. Vytvoření prvního MD5 hashe s použitím uživatelského jména, oblasti, což se dá považovat jako doména v protokolu SIP a hesla. Konkrétně je nutné vytvořit hash z textového řetězce: <uživatelské jméno>:<oblast>:<heslo>. Například „10:asterisk:asdf“.
2. Vytvoření druhého MD5 hashe s použitím typu požadavku a URI požadavku. Jedná se tedy o hash textového řetězce: <typ požadavku>:<URI požadavku>. Jako příklad lze uvést „REGISTER:sip:192.168.89.222;transport=UDP“.
3. Vytvoření finálního MD5 hashe s použitím vytvořených hashů z předchozích dvou kroků a hodnoty nonce nacházející se v poli *WWW-Authenticate* hlavičky odpovědi 401 UNAUTHORIZED. Finální hash má následující podobu: <první hash>:<nonce>:<druhý hash>.

S existujícím finálním hashem lze vytvořit pole *Authorization*, které se skládá z hodnot: uživatelské jméno, oblast, nonce, URI, typ hashe a finální hash. Toto pole je návratový typ metody.

Metoda *processRequest(RequestEvent)* je implementovaná metoda rozhraní *SipListener*, která je volána vždy, když je obdržen nějaký požadavek od SIP serveru. Jelikož třída *Caller* reprezentuje uživatelského agenta, nejčastěji přijímaný požadavek je BYE. V takovém případě je volána metoda *processBye(Request, ServerTransaction)*. V případě, kdy je obdržen požadavek jiný, je na něho odpovězeno odpovědí 202 ACCEPTED.

Metoda *processBye(Request, ServerTransaction)* je jednoduchá metoda reagující na obdržený požadavek BYE odpovědí 200 OK. Po odeslání této odpovědi je zavolána metoda *deleteSipProvider()*. Jelikož je hovor ukončen, je zbytečné, aby byly plýtvány výpočetní prostředky na poslouchání příchozích SIP zpráv, a proto je *SipListener* odstraněn.

Metoda *processResponse(ResponseEvent)* reaguje na přijaté odpovědi od SIP serveru. Jedná se o implementovanou metodu rozhraní *SipListener*. Nejdříve se ověřuje, zdali je odpověď součástí nějaké transakce. Pokud ne, a zároveň na tuto odpověď bylo již reagováno v minulosti požadavkem ACK, znamená to, že se jedná o odpověď 2XX a požadavek byl během transportu po síti ztracen, nebo došlo k takovému zpoždění na síti, že SIP server odeslal odpověď opakovaně. Dle [20] je nutné odeslat požadavek ACK na každou 2XX odpověď v rámci SIP relace, proto se musí požadavek ACK odeslat znovu, i když byl v minulosti odeslán. Pokud odpověď je součástí transakce, je vykonán komplexní *if* blok, který rozhoduje, o který typ odpovědi se jedná. Pokud se jedná o odpověď 401 UNAUTHORIZED, je zavolána metoda *makeAuthorizationHeader(Request, Response)*, jež vytvoří autorizační pole a to je následně přidáno do hlavičky požadavku, na který bylo reagováno. Po přidání je nutné zvýšit sekvenční číslo požadavku o jedna a v poli Via se musí odstranit parametr branch. To je nezbytné, jelikož podle [20] každá SIP transakce musí mít tento parametr jedinečný. Vytvoření nové hodnoty tohoto parametru není nutné, o to se stará knihovna *JAIN-SIP*. Po odstranění parametru lze požadavek znovu odeslat. Pokud se jedná o odpověď 200 OK, jenž odpovídá na požadavek REGISTER, je změněn status komunikace na REGISTERED a je odstraněn *SipListener*. V případech, kdy jsou obdrženy odpovědi 100 TRYING nebo 180 RINGING je status komunikace změněn podle obdržené odpovědi, tedy na TRYING, nebo RINGING. Tyto statusy neslouží k vyhodnocení žádného testování, jsou měněny pouze pro snadnější debugování modulu. V okamžiku obdržení odpovědi 202 ACCEPTED na požadavek MESSAGE je změněn status na MESSAGE_ACCEPTED a je odstraněn *SipListener*. Při obdržení odpovědi 200 OK na požadavek INVITE je na odpověď zareagováno požadavkem ACK a status je změněn na CONNECTED. Pokud byla ovšem odpověď 200 OK odpovědí na požadavek CANCEL, i když je aktuální stav dialogu CONFIRMED, znamená to, že se požadavek CANCEL ztratil, nebo zpozdil na síti. Musí se proto odeslat požadavek BYE, aby se hovor ukončil.

Metoda *processTimeout(TimeoutEvent)* je volána, když koncové zařízení přijímající zprávu neodpoví včas. Jedná se o zvláštní situaci, na kterou neexistuje žádná SIP odpověď.

Když dojde k problému se vstupem, nebo výstupem protokolu SIP, je volána metoda, která se nazývá *processIOException(IOExceptionEvent)*.

Dvě metody *processTransactionTerminated(TransactionTerminatedEvent)* a *processDialogTerminated(DialogTerminatedEvent)* jsou volány, když dojde k situaci, kdy je transakce nebo dialog mezi dvěma entitami ukončen.

Poslední čtyři popisované metody pouze vypisují do konzole informace o tom, že se daná událost stala, nijak na ně ovšem nereagují.

Grafické uživatelské rozhraní

Strukturu grafického uživatelského rozhraní zajišťuje balíček *cz.vut.fekt.utko.sipsampler.gui*. Obsahuje celkem tři třídy, a to konkrétně *SamplerGui*, *SamplerPanel* a *HintJTextField*. UML diagram tohoto balíčku je zobrazen na obrázku 3.4.

Třída **SamplerGui** dědí ze třídy *AbstractSamplerGui* a je zodpovědná za implementaci grafického uživatelského rozhraní. Třída nesmí obsahovat žádné odkazy na testovací prvek, protože program JMeter používá tuto třídu pro více testovacích prvků, kvůli optimalizaci využití výpočetních prostředků.

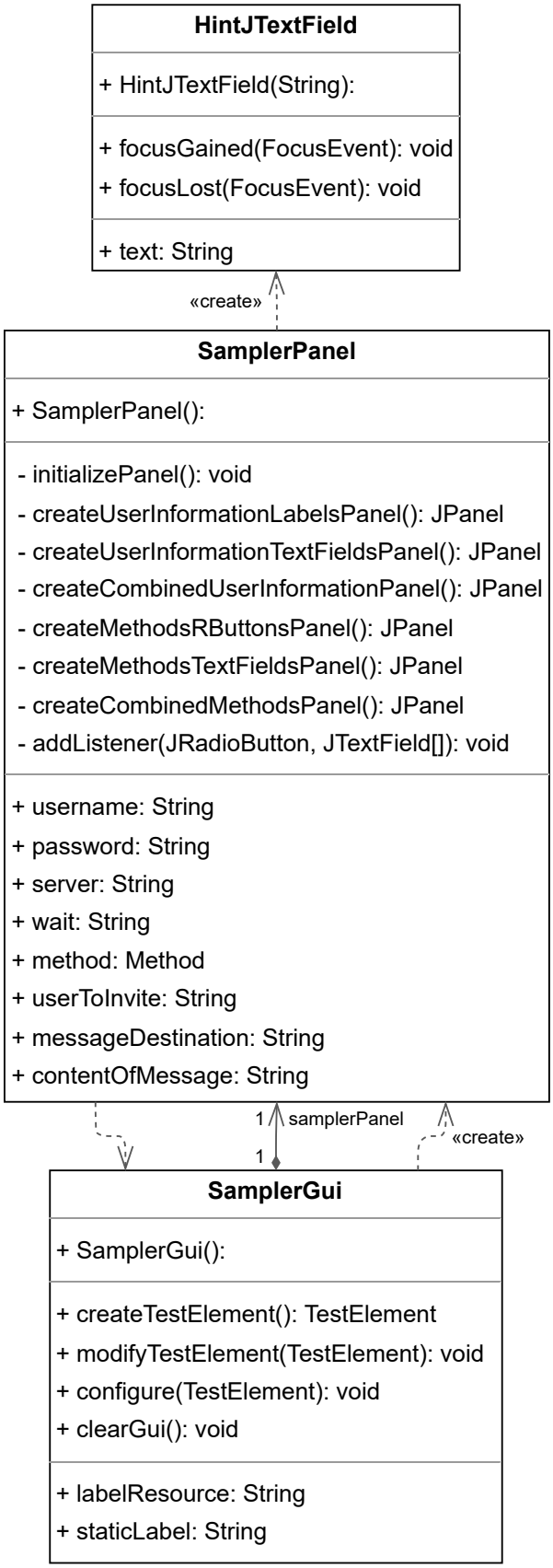
Konstruktor třídy vytváří instanci třídy *SamplerPanel*, která se stará o vzhled GUI a bude popsána později v textu. Je zde také volána metoda *makeTitlePanel()*, která je implementována programem JMeter a slouží k vytvoření panelu s názvem a komentářem daného testovacího prvku.

Po konstruktoru již následují pouze metody, které je nutné přepsat ze třídy *AbstractSamplerGui*. První dvě se nazývají *getLabelResource()* a *getStaticLabel()*. Definují název modulu, jenž lze vidět v grafickém uživatelském rozhraní programu JMeteru a v rozevíracím seznamu komponent JMeteru. Vrací textový řetězec „SIP Sampler“.

Následující metoda *createTestElement()* zodpovídá za vytvoření nové instance třídy *Sampler*. Po jejím vytvoření musí být GUI aktualizováno a instance je metodou vrácena.

Jak již bylo zmíněno, JMeter používá tuto třídu pro všechny testovací prvky. Tudíž je nutné implementovat dvě metody, které tuto funkcionalitu podporují. Jedná se o *modifyTestElement(TestElement)* a *configure(TestElement)*. Tyto dvě metody jsou svým způsobem protikladné, ale zároveň se doplňují.

Metoda *modifyTestElement(TestElement)* předává všechny informace z grafického uživatelského rozhraní do testovacího prvku. Nejdříve je zavolána metoda *configureTestElement(TestElement)* z rodičovské třídy k nastavení názvu, komentáře



Obr. 3.4: UML diagram balíčku cz.vut.fekt.utko.sipsampler.gui.

a stavu Sampleru. Poté jsou již veškeré informace z GUI předány testovacímu prvku.

Metoda *configure(TestElement)* načítá pole grafického užiatelského rozhraní hodnotami z testovacího prvku. Logika je stejná jako v předchozí metodě, pouze opačná. Je tedy volána metoda *configure(TestElement)* z rodičovské třídy a následně jsou všechny informace z testovacího prvku předány do GUI.

Poslední přepisovaná metoda třídy *SamplerGui* je *clearGui()*. Ta odstraňuje vyplněné hodnoty uživatelem v GUI a nastaví je na výchozí. Metoda je obvykle volána při vytváření nových testovacích prvků. Stejně jako u předchozích metod je zavolána metoda z rodičovské třídy, konkrétně metoda *clearGui()* a následuje nastavení výchozích hodnot v GUI.

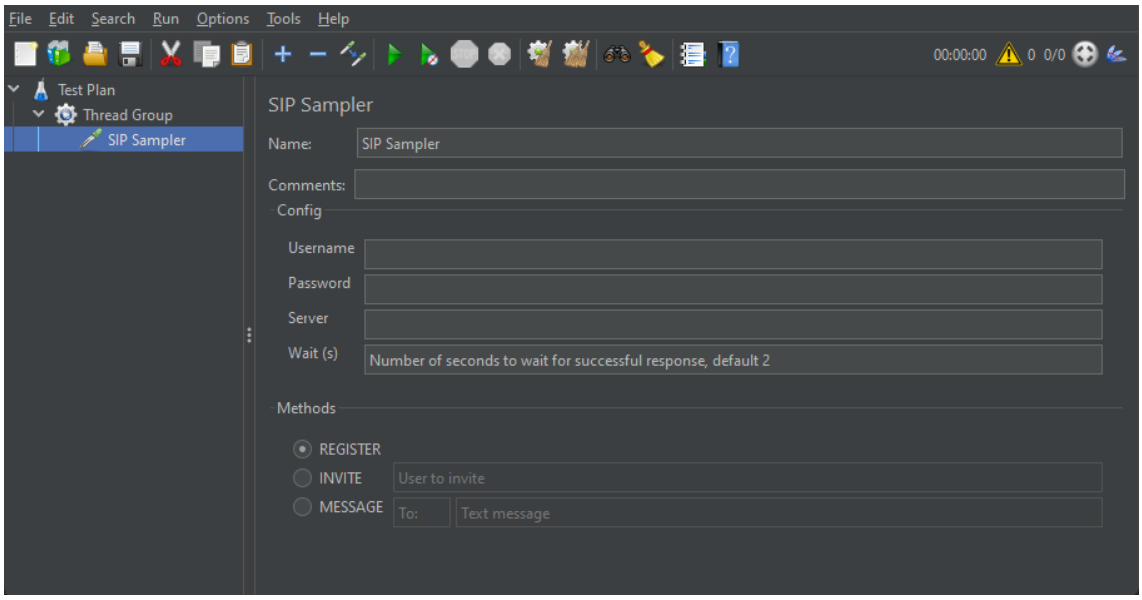
Třída **SamplerPanel** dědí ze třídy *JPanel*, jež reprezentuje grafickou komponentu panel, na kterou se dají vkládat další komponenty jako například textové pole, přepínače nebo zaškrtávací políčka. V této třídě jsou definovány všechny grafické prvky nacházející se v grafickém uživatelském rozhraní. Jedná se například o textová pole pro zadání uživatelského jména a hesla nebo o přepínače, pomocí kterých si uživatel vybírá, jakými SIP požadavky bude zatěžovat testovaný server.

Grafické uživatelské rozhraní je rozděleno na dva různé sektory a každý sektor vytváří k tomu určená metoda. V prvním sektoru nazvaným Config se nachází pole určená k obdržení informací o uživateli, IP adrese testovaného SIP serveru a době, jak dlouho má modul čekat na finální odpověď od serveru. V druhém sektoru s názvem Methods jsou přepínače, které určují jakými SIP zprávami bude testovaný server zatížen. V případě vybrání možnosti INVITE nebo MESSAGE jsou aktivována textová pole, která přijímají doplňující informace nezbytné k odeslání těchto požadavků. U vybrání možnosti INVITE je aktivováno pole, jež přijímá jméno uživatele, kterému má být požadavek odeslán. U možnosti MESSAGE je aktivováno pole se stejnou funkcí, a navíc je aktivováno pole určené pro vyplnění zprávy, která má být danému uživateli odeslána.

Konstruktor třídy volá metodu *initializePanel()*, sloužící k poskládání výsledného panelu ze dvou výše zmíněných sektorů, tedy Config a Methods. Jedná se o další panely, které jsou vytvořeny v metodách *createCombinedUserInformationPanel()* a *createCombinedMethodsPanel()*. Každý z těchto panelů je sestaven ze dvou dalších panelů z důvodu správného zarovnání textových polí.

Metoda *addListener(JRadioButton, JTextField[])* slouží k přidání listeneru k přepínačům za účelem aktivace, nebo deaktivace příslušných textových polí. Metoda je volána pouze u přepínačů INVITE a MESSAGE, jelikož přepínač REGISTER nemá žádná přiřazená textová pole. Celá metoda pracuje ve smyčce for, která prochází seznam všech textových polích předaný metodě jako parametr.

Výsledné grafické uživatelské rozhraní modulu v programu JMeter je zobrazeno na obrázku 3.5.



Obr. 3.5: Grafické rozhraní modulu.

Třída **HintJTextField** dědí ze třídy *JTextField*, což je reprezentace grafické komponenty textového pole, a implementuje rozhraní *FocusListener*. Tato třída poskytuje přidanou funkcionalitu textovému poli v podobě nápovědy. Jedná se o text, který je zobrazen v poli a popisuje jeho funkci. Při kliknutí do pole text zmizí a uživatel může pole libovolně upravovat, tedy psát nebo mazat text v poli. Pokud uživatel opustí pole v momentě, kdy je prázdné, nápověda se opět zobrazí. V opačném případě zůstane v poli text, který uživatel zadal.

Konstruktor třídy volá konstruktor rodičovské třídy, čímž se nastaví text pole na danou nápovědu a následně se přidá k tomuto textovému poli *FocusListener*, který je zavolaný, když je v poli zahájena úprava textu, nebo je tato úprava ukončena.

Metoda *focusGained(FocusEvent)* je volána při zahájení editace pole. Pokud je textové pole prázdné, tedy uživatel v něm v minulosti neprovedl ještě žádné změny, text se nastaví na prázdný řetězec. V opačném případě se nic neděje a uživatelem zadaný vstup z minulosti zůstává nezměněný.

Metoda *focusLost(FocusEvent)* je zavolána při ukončení editace pole. Pokud uživatel nezadal žádný text do pole během editace, je nápověda opět zobrazena. V opačném případě není provedeno nic.

3.3.3 Import modulu do programu JMeter

Po dokončení modulu je nutné projekt exportovat do souboru .jar. Jelikož se v projektu nachází soubor pom.xml, je možné export provést jednoduše příkazem *mvn clean package* v kořenové složce projektu. Export lze také provést přímo ve vý-

vojovém prostředí IntelliJ IDEA nastavením konfigurace Maven. Po dokončení exportu se musí vygenerovaný soubor přesunout do složky *lib/ext*, která se nachází v hlavní složce programu JMeter. Jelikož JMeter nemá importované všechny použité knihovny v projektu, je nutné je vložit do složky *lib*. Jedná se o knihovny *awaitility* a *JAIN-SIP*. V této fázi vývoje lze program JMeter spustit a nově vytvořený modul bude viditelný v seznamu samplerů.

3.4 Pobočková ústředna Asterisk

K testování, zdali je modul vyvinut správně, je potřeba telefonní server, jenž bude přijímat SIP zprávy odesílané modulem a odpovídat na ně. Byl zvolen server Asterisk, protože se jedná o jednu z nejrozšířenějších otevřených pobočkových ústředen. Server funguje v operačním systému Ubuntu 20.04 běžící ve virtuálním prostředí VMware.

3.4.1 Instalace

Jelikož Ubuntu repozitáře v době vypracovávání této práce obsahují staré verze Asterisku, je instalace provedena ze zdrojového kódu. Před instalací je nutné zajistit prerekvizity, které jsou nezbytné pro stažení a sestavení Asterisku. To lze provést následujícími příkazy v příkazové řádce.

Výpis 3.6: Instalace prerekvizit Asterisku.

```
$ sudo apt update                                                    1
$ sudo apt install wget build-essential git autoconf \               2
> subversion pkg-config libtool                                      3
```

V případě, kdy by bylo potřeba, aby ústředna komunikovala s analogovými nebo ISDN telefony, bylo by nezbytné nainstalovat knihovny *DAHDI* a *LibPRI*, které tuto možnost umožňují. V tomto případě to není potřeba, proto tyto knihovny nainstalovány nebudou.

Dalším krokem je stažení ústředny Asterisk z Internetu. Lokace instalace je volitelná a v této práci byla zvolena složka */usr/src*. Stažení je provedeno následovně.

Výpis 3.7: Stažení zdrojových souborů Asterisku.

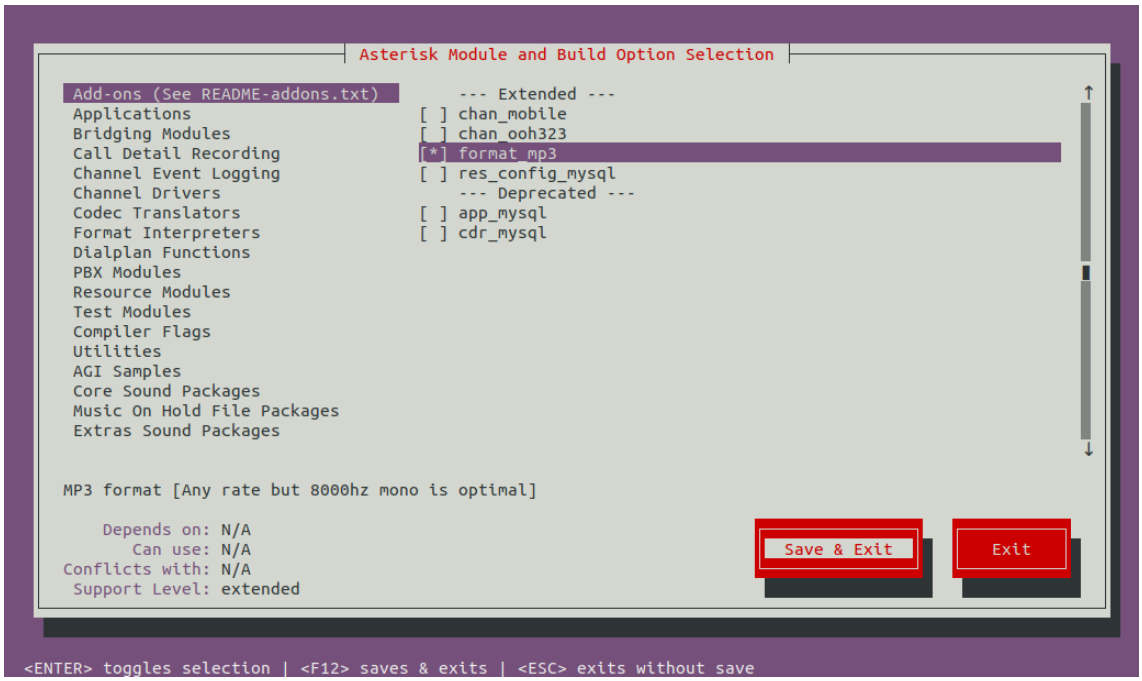
```
$ cd /usr/src                                                        1
$ sudo git clone -b 18 https://gerrit.asterisk.org/asterisk \        2
> asterisk-18                                                        3
```

V době psaní této práce je aktuální verze Asterisku 18. Pokud by byla k dispozici verze novější, stačí změnit číslo 18 v příkazu výše.

Po stažení nelze Asterisk hned instalovat, nejdříve se musí stažené zdrojové soubory zkompilovat a před samotným zkompilováním je potřeba provést několik nastavení. Nejdříve se musí spustit skript *get_mp3_source.sh*, který se postará o stažení mp3 zdrojů potřebných ke kompilaci mp3 modulu. Ten je nezbytný k používání mp3 souborů s ústřednou Asterisk. Dále se musí spustit skript *install_prereq*, který jak už název napovídá, nainstaluje potřebné knihovny k fungování ústředny. Skript *configure* následně ověří, zdali se všechny knihovny nainstalovaly správně. Po úspěšné kontrole je nutné vybrat moduly, které se mají zkompilovat a nainstalovat. Tento výběr je zobrazen na obrázku 3.6. Po tomto kroku nezbývá nic jiného, než kompilaci spustit. V závislosti na výpočetním výkonu stroje může kompilace určitou dobu trvat. Příznakem *-j* lze nastavit počet jader procesoru, jenž bude využit. Celý postup výše zmíněných úkonů je zobrazen ve výpise 3.8

Výpis 3.8: Nastavení kompilace Asterisku.

```
$ cd /usr/src/asterisk-18                                    1
$ sudo contrib/scripts/get_mp3_source.sh                     2
$ sudo contrib/scripts/install_prereq install                3
$ sudo ./configure                                           4
$ sudo make menuselect                                       5
$ sudo make -j2                                              6
```



Obr. 3.6: Vybrání volitelných modulů pro Asterisk.

Po úspěšné kompilaci je na řadě spuštění instalace. Celý postup je zobrazen

ve výpise 3.9. První příkazem se provede instalace ústředny. Dalším příkazem se volí, zdali chce uživatel instalaci s předpřipravenými, nebo základními konfiguračními soubory. V tomto případě jsou zvoleny předpřipravené soubory. Pro možnost základních souborů by místo příkazu na druhém řádku v popisovaném výpise 3.9 musel být příkaz *sudo make basic-pbx*. Dalším krokem je instalace inicializačního skriptu Asterisku a na závěr je vhodné spustit aktualizaci mezipaměti sdílených knihoven.

Výpis 3.9: Instalace Asterisku.

```
$ sudo make install                                                    1
$ sudo make samples                                                    2
$ sudo make config                                                     3
$ sudo ldconfig                                                        4
```

Ve výchozím nastavení je Asterisk spuštěn jako uživatel root. Z bezpečnostních důvodů je nejlepší praktikou vytvoření nového uživatele, jenž nebude mít rootovská práva, pouze práva potřebná k běhu ústředny Asterisk. Proto je nutné vytvořit nového systémového uživatele nazvaného například asterisk. To lze uskutečnit příkazem zobrazeným ve výpise 3.10.

Výpis 3.10: Vytvoření nového uživatele asterisk.

```
$ sudo adduser --system --group --home /var/lib/asterisk \             1
> --no-create-home asterisk                                            2
```

Aby se ústředna spouštěla pod nově vytvořeným uživatelem, je nutné změnit nastavení v konfiguračním souboru */etc/default/asterisk*. V tomto souboru je potřeba najít dva řádky zobrazené ve výpise 3.11 a odkomentovat je, neboli smazat symbol # na začátku každého řádku.

Výpis 3.11: Úprava konfiguračního souboru.

```
AST_USER="asterisk"                                                    1
AST_GROUP="asterisk"                                                   2
```

Dále je zapotřebí přidat nově vytvořeného uživatele do již existujících skupin *dialout* a *audio* a změnit vlastnictví a práva všech souborů a adresářů Asterisku, aby k nim měl uživatel asterisk oprávnění přistupovat. To se provede následujícím postupem.

Výpis 3.12: Změna práv souborů a adresářů Asterisku.

```
$ sudo usermod -a -G dialout,audio asterisk                            1
$ sudo chown -R asterisk: /var/{lib,log,run,spool}/asterisk \          2
> /usr/lib/asterisk /etc/asterisk                                      3
$ sudo chmod -R 750 /var/{lib,log,run,spool}/asterisk \                4
> /usr/lib/asterisk /etc/asterisk                                      5
```

Po upravení přístupových práv souborů ústředny Asterisk je vše připraveno k jejímu spuštění. To lze provést příkazem na prvním řádku ve výpise 3.13. Pro přístup do ovládací konzole Asterisku lze použít příkaz na druhém řádku výpisu 3.13, kde počet písmen *v* ovlivňuje výmluvnost (verbosity) ústředny, tedy počet zobrazených informací v konzoli. Pokud by uživatel chtěl, aby se Asterisk spouštěl při každém startu serveru, lze použít příkaz na třetím řádku výpisu 3.13.

Výpis 3.13: Spuštění ústředny Asterisk.

```
$ sudo systemctl start asterisk
$ sudo asterisk -vvr
$ sudo systemctl enable asterisk
```

3.4.2 Konfigurace

Pobočková ústředna Asterisk se konfiguruje pomocí konfiguračních souborů, které jsou uloženy v adresáři */etc/asterisk*. Pro testování vyvinutého modulu bude dostačující vytvoření jednoho uživatele. To lze provést v souboru *pjsip.conf*, nacházející se v již zmíněném adresáři. Jeho obsah je zobrazen ve výpise 3.14 a byl inspirován z oficiální Asterisk dokumentace [21]. Konfigurační soubory jsou rozděleny do bloků textu. Jedná se o organizační jednotky v rámci konfiguračního souboru. Díky nim jsou různé části souboru na sobě nezávislé a nastavení je přehledné a snadněji čitelné. V prvním bloku je definovaný transportní protokol, v tomto případě se jedná o protokol UDP. Další bloky již obsahují samotné nastavení uživatele, jehož název je 10. Jsou zde vypsány podporované kodeky, odkaz na kontext v souboru *extensions.conf*, což bude probráno v následující části práce, nebo způsob autorizace uživatele. Na řádcích sedmnáct a osmnáct lze vidět uživatelovo heslo a jméno.

Výpis 3.14: Vytvoření uživatele v souboru pjsip.conf.

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[10]
type=endpoint
context=kontext1
disallow=all
allow=alaw
auth=auth10
aors=10
```

```
                                                                    13
[auth10]                                                            14
type=auth                                                           15
auth_type=userpass                                                  16
password=asdf                                                       17
username=10                                                         18
                                                                    19
[10]                                                                20
type=aor                                                            21
max_contacts=100000                                                 22
```

Vytvoření uživatele je zobrazeno ve výpise 3.14. Jak se samotná ústředna chová při vytočení různých klapek, je definováno v souboru *extensions.conf* nacházející se opět v adresáři */etc/asterisk*. Obsah tohoto souboru je zobrazen ve výpise 3.15. V souboru jsou definovány tři kontexty. Kontext s názvem *kontext1* obsahuje definici tří klapek a to konkrétně klapky *9999*, na kterou když se zavolá, je uživateli umožněno nahrát hlasovou zprávu, která je uložena do souboru. Klapka byla využita pro nahrání hlasových zpráv, ze kterých byl následně vytvořen IVR systém. Další definovanou klapkou je *111*. Její jediná funkce je při vytočení vypsat do konzole zprávu „Message Received“, jelikož slouží k zátěžovému testování serveru MESSAGE požadavky. Poslední definovanou klapkou je klapka *100*, která při vytočení odkazuje na kontext s názvem *zakladni_nabidka*, ve kterém je definovaný jednoduchý IVR systém, neboli Interactive Voice Response systém, což lze přeložit jako systém interaktivní hlasové odezvy. Jedná se o předem nahrané hlasové záznamy, které informují uživatele o různých skutečnostech nebo od uživatele získávají informace a podle nich směrují hovor na oddělení, se kterým by si přál uživatel mluvit. IVR systém definovaný ve výpise 3.15 po stisku číslice jedna přehraje zvuk kytary a při stisku číslice dvě přehraje zvuk piána. Po přehrání zvuku se hovor ukončí. V případě, kdy by uživatel zvolil špatnou možnost, IVR systém o tom uživatele informuje a vyzve ho k opětovnému zadání možnosti. V případě nezvolení žádné možnosti po dobu deseti sekund je hovor ukončen odesláním SIP požadavku BYE. To je definováno ve třetím kontextu s názvem *kontext_hangup*, jehož jediná funkce je ukončení hovoru.

Výpis 3.15: Nastavení chování ústředny v souboru extensions.conf.

```
[zakladni_nabidka]                                                  1
exten => s,1,Set(COUNTER=0)                                         2
exten => s,n,Playback(uvitani)                                      3
exten => s,n(start),Background(nabidka)                             4
exten => s,n,WaitExten(10)                                          5
                                                                    6
exten => 1,1,Playback(guitar)                                       7
```

```
exten => 1,n,Wait(2)                                                    8
exten => 1,n,Goto(kontext_hangup,s,1)                                   9
exten => 2,1,Playback(piano)                                            10
exten => 2,n,Wait(2)                                                    11
exten => 2,n,Goto(kontext_hangup,s,1)                                   12
                                                                        13
exten => i,1,Playback(spatnavolba)                                      14
exten => i,n,Wait(2)                                                    15
exten => i,n,Goto(s,start)                                              16
                                                                        17
exten => t,1,Set(COUNTER=$[${COUNTER} + 1])                             18
exten => t,n,GotoIf($[ ${COUNTER} >= 1]?kontext_hangup,s,1)             19
exten => t,n,Playback(vyprsenicasu)                                     20
exten => t,n,Wait(2)                                                    21
exten => t,n,Goto(s,start)                                              22
                                                                        23
[kontext1]                                                              24
exten => 100,1,Goto(zakladni_nabidka,s,1)                               25
                                                                        26
exten => 111,1,NoOp(Message Received)                                   27
                                                                        28
exten => 9999,1,Answer                                                  29
exten => 9999,n,Wait(2)                                                 30
exten => 9999,n,Record(asterisk-test_sound%d:alaw)                      31
exten => 9999,n,Wait(2)                                                 32
exten => 9999,n,Playback(${RECORDED_FILE})                              33
exten => 9999,n,Wait(2)                                                 34
exten => 9999,n,Hangup                                                  35
                                                                        36
[kontext_hangup]                                                        37
exten => s,1,HangUp()                                                   38
```

Tento IVR systém byl vyvinut, aby sloužil jako cíl zátěžového testování serveru SIP požadavky INVITE a MESSAGE.

3.5 Testování modulu

Po vyvinutí vlastního modulu, jeho exportu a vložení do složky s externími knihovnami JMeteru a instalaci a konfiguraci telefonní ústředny Asterisk je experimentální pracoviště připraveno k otestování modulu.

Testování bylo provedeno za pomoci školního ICT testeru, na kterém byl spuštěn program JMeter s importovaným modulem. Tento tester byl využit za účelem získání lepších vypovídajících hodnot měření, než kterých by bylo dosaženo při testování na domácím počítači s dvěma virtuálními stroji. Parametry školního testeru jsou vypsány v tabulce 3.1.

Tab. 3.1: Parametry školního ICT testeru.

Procesor	Intel(R) Xeon(R) CPU E5-2680 v4
Frekvence procesoru	2400 MHz
Počet jader	14
Operační paměť	128 GB
Přenosová rychlost LAN portu	10 Gb/s
Operační systém	Ubuntu 20.04

Server s pobočkovou ústřednou Asterisk běžel ve virtuálním prostředí VMware s operačním systémem Ubuntu 20.04. Veškeré parametry testovaného systému jsou vypsány v tabulce 3.2.

Tab. 3.2: Parametry testovaného systému.

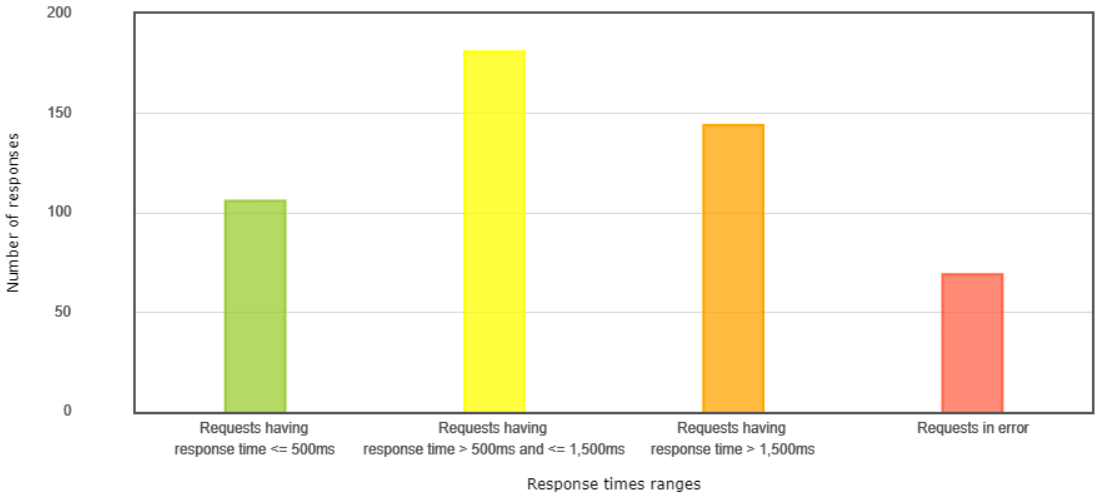
Procesor	Intel Core i5
Frekvence procesoru	1600 MHz
Počet jader	2
Operační paměť	4 GB
Přenosová rychlost LAN portu	1 Gb/s
Operační systém	Ubuntu 20.04

Byly navrženy 4 testovací scénáře. V prvních třech byla telefonní ústředna zatěžována pouze jedním typem SIP požadavku současně, ve čtvrtém scénáři byla ústředna zatížena všemi třemi typy SIP požadavků současně, tedy byly na ústřednu posílány požadavky REGISTER, INVITE i MESSAGE ve stejný moment.

3.5.1 Zatěžování požadavky REGISTER

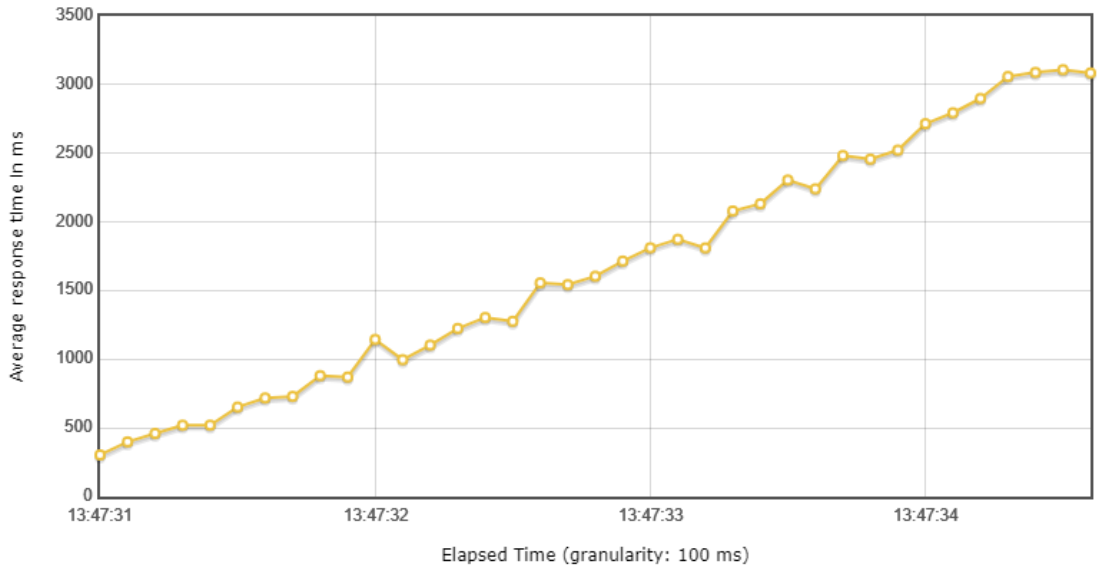
Prvním realizovaným scénářem bylo zatížení ústředny požadavky typu REGISTER. Jak probíhá komunikace mezi SIP klientem a registračním serverem bylo znázorněno na obrázku 1.1. Bylo vytvořeno 500 vláken a všechna paralelně odeslala na SIP server žádost o registraci uživatele. Doba čekání na finální odpověď byla nastavena na 3 sekundy. Celková úspěšnost tohoto testování byla 86,2 %. Z grafu 3.7 lze vyčíst, že u 69 požadavků byla odezva finální odpovědi delší než 3 sekundy, což

modul vyhodnotil jako chybu. Největší počet požadavků spadá do kategorie finálních odpovědí s odezvou od 500 do 1500 milisekund. Z grafů 3.8 a 3.9 je patrné, že

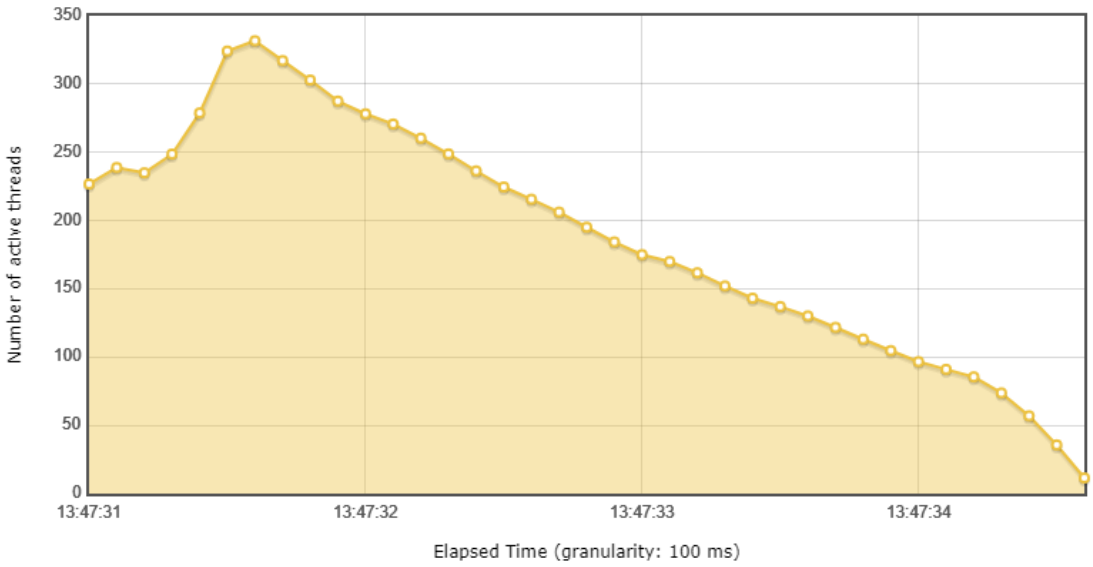


Obr. 3.7: Přehled odezvy ústředny při zatěžování požadavky REGISTER.

i když počet vláken první půl sekundu stále rostl až na počet 332 aktivních vláken ve stejný moment a pak postupně v průběhu 3 sekund klesal až do okamžiku, kdy nebyla žádná vlákna aktivní, doba odezvy finálních odpovědí se postupně vyšplhala až ke 3 sekundám, kdy se testovací prvky začali vyhodnocovat jako chybné.



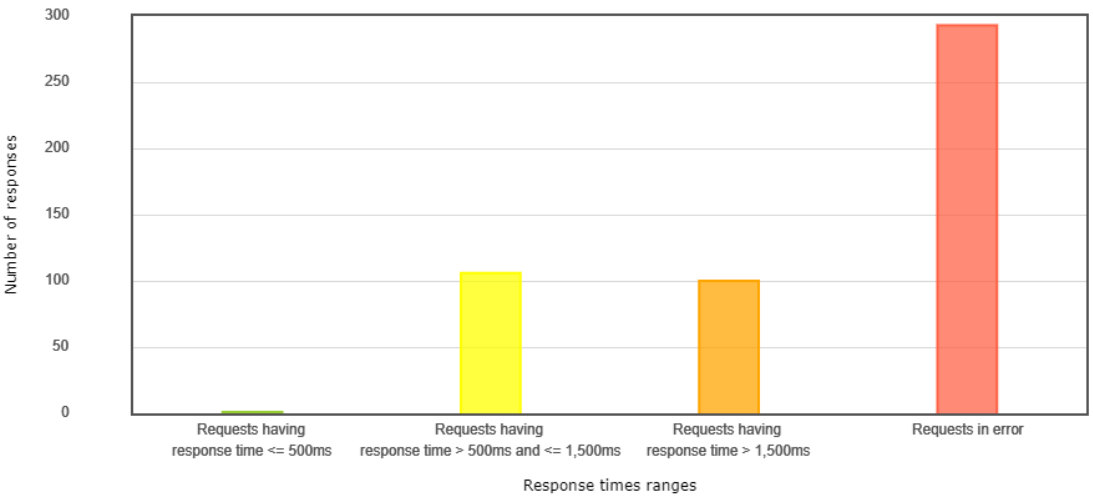
Obr. 3.8: Vývoj odezvy ústředny při zatěžování požadavky REGISTER.



Obr. 3.9: Počet aktivních vláken při zatěžování požadavky REGISTER.

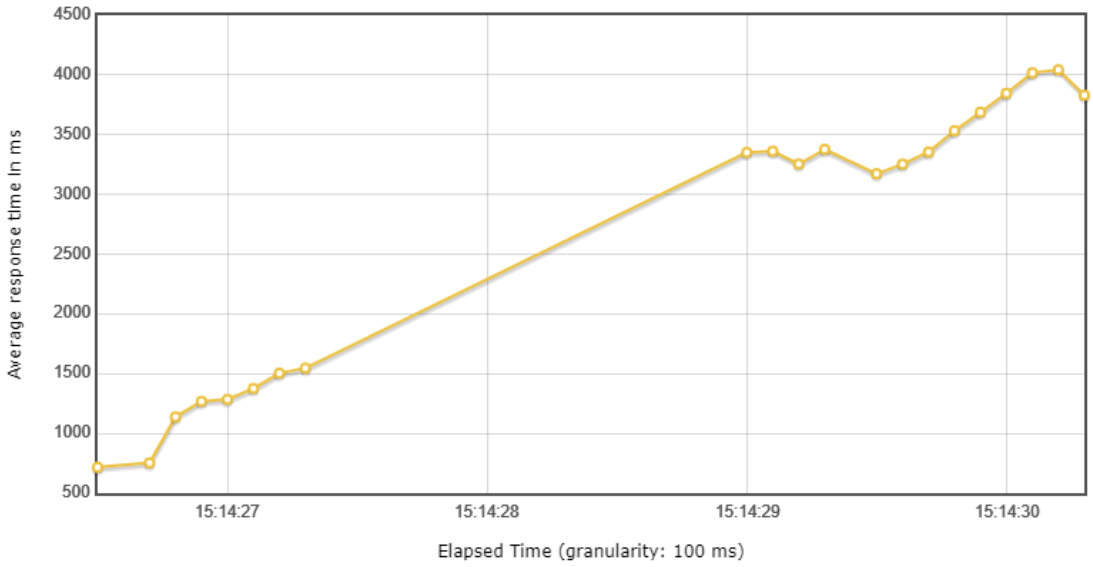
3.5.2 Zatěžování požadavky INVITE

V tomto testovacím scénáři byla telefonní ústředna zatěžována požadavky typu INVITE. Průběh komunikace při navazování SIP relace je znázorněn na obrázku 3.3. Jedná se o nejkomplexnější požadavek, jelikož tentokrát neprobíhá výměna dat pouze v podobě signalizačního protokolu SIP, ale jsou přenášena i data samotného hovoru v podobě protokolu RTP, která musí SIP server generovat. Byl použit kodek G.711A s bitovou rychlostí 64 Kb/s.

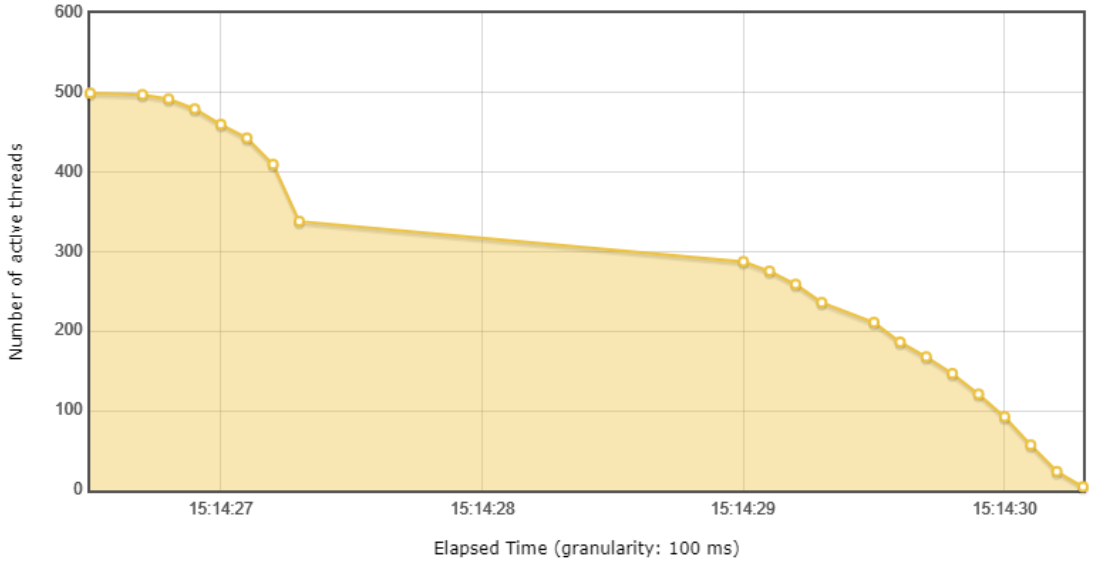


Obr. 3.10: Přehled odezvy ústředny při zatěžování požadavky INVITE.

Opět bylo vytvořeno 500 vláken, která paralelně odeslala požadavek INVITE s nastavenou čekací dobou 3 sekundy na finální odpověď. Tentokrát bylo úspěšných pouze 41,4 % testovacích prvků. Tuto skutečnost zobrazuje graf 3.12, kdy počet chybných prvků byl 293 a odezva úspěšných testovacích prvků se pohybovala mezi 500 až 3000 milisekundami.



Obr. 3.11: Vývoj odezvy ústředny při zatěžování požadavky INVITE.

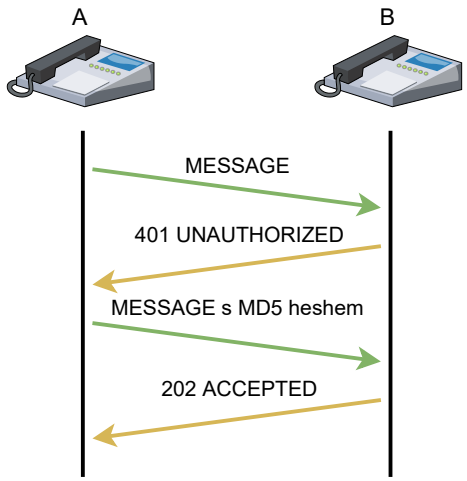


Obr. 3.12: Počet aktivních vláken při zatěžování požadavky INVITE.

Z grafů 3.11 a 3.12 lze vyčíst, že hned na samém počátku zatěžování bylo všech 500 vláken aktivních a pobočková ústředna dokázala zhruba jednu sekundu zpracovávat příchozí požadavky. Zvládla odpovědět 162 vláknům před tím, než se kompletně zahltila mediálními daty odesílanými na ICT tester, a přestala reagovat na téměř 2 sekundy. Než opět začala odpovídat uběhly nastavené 3 sekundy a všechny obdržené odpovědi byly vyhodnoceny jako chybné. Důvodem tohoto selhání nebyla nedostatečná přenosová kapacita dat, jelikož propustnost dat v průběhu testování byla pouze 1,5 Mb/s, kdežto stroje byly spojeny 1 Gb spojením. Důvodem byla pravděpodobně nedostatečná velikost operační paměti nebo slabý procesor SIP serveru, který tudíž nedokázal zpracovávat takovou zátěž.

3.5.3 Zatěžování požadavky MESSAGE

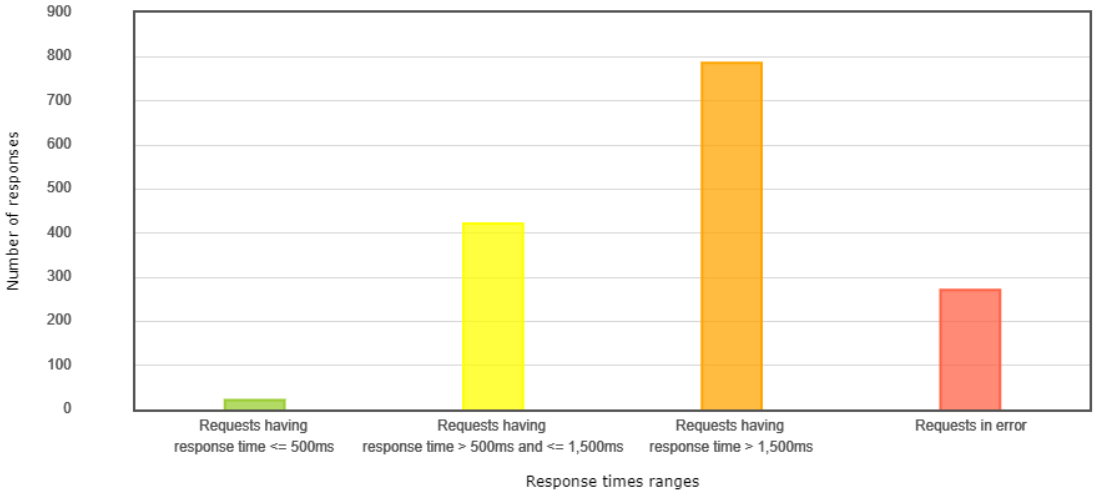
Třetím realizovaným testovacím scénářem bylo zatížení SIP serveru požadavky typu MESSAGE. Komunikace mezi jednotlivými SIP prvky během odesílání požadavku MESSAGE je velice podobná té, kde uživatel odesílá požadavek REGISTER. Proces odeslání požadavku MESSAGE je zobrazen na obrázku 3.13. Na požadavek MESSAGE je od ústředny odeslána výzva k ověření uživatele. Na tu je reagováno vytvořením MD5 hashe obsahující uživatelovo heslo a požadavek je znovu odeslán. V případě úspěšné autentizace je požadavek potvrzen odpovědí 202 ACCEPTED.



Obr. 3.13: Průběh komunikace při odesílání požadavku typu MESSAGE.

Přestože komunikace během odesílání požadavku MESSAGE a REGISTER jsou si velice podobné, úkony, které musí při přijetí těchto požadavků ústředna vykonat, se zcela liší. Při přijetí požadavku REGISTER musí SIP server přiřadit uživatelské jméno ke konkrétní síťové adrese a následně tuto kombinaci uložit do databáze. Až poté odpovídá zprávou 200 OK. Kdežto při přijetí požadavku MESSAGE ústředna

nemusí vykonávat žádné další úkony, rovnou je odeslána odpověď 202 ACCEPTED potvrzující úspěšné doručení požadavku. Z tohoto důvodu bylo nutné vytvořit o 1000 vláken více než u předchozích zátěžových testováních, tedy 1500 vláken, aby byly v grafech reprezentovány i nějaké chybné testovací prvky. Čekací doba na finální odpověď zůstala nastavena na 3 sekundy. Graf 3.14 ukazuje, že více než polovina testovacích prvků měla dobu odezvy mezi 1500 a 3000 milisekundami a 271 prvků bylo vyhodnoceno jako chybných, což tvoří 18,07 % z celkového počtu. Z toho vyplývá, že úspěšnost tohoto zátěžového testování byla 81,93 %.

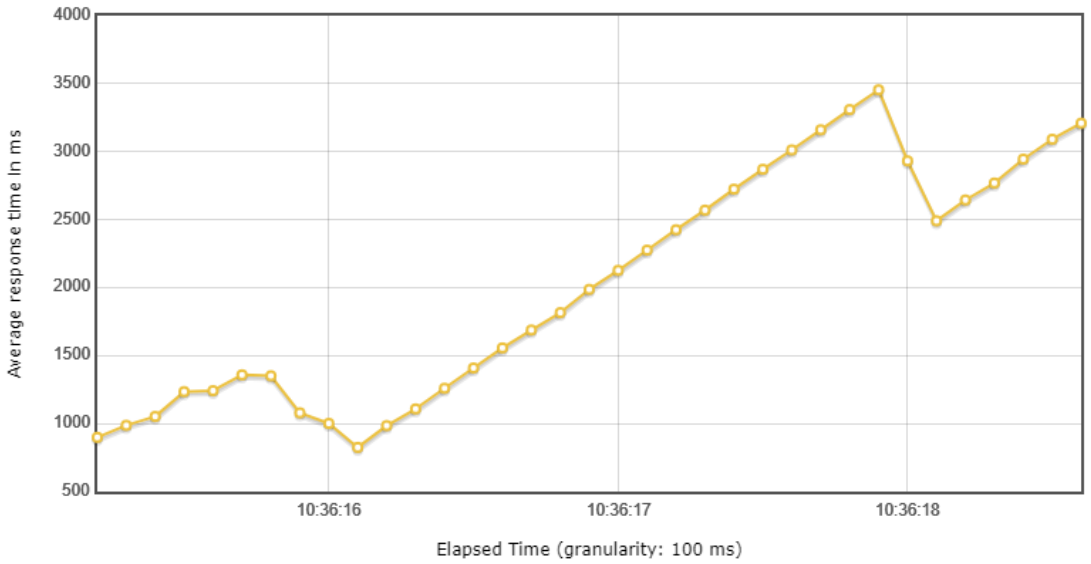


Obr. 3.14: Přehled odezvy ústředny při zatěžování požadavky MESSAGE.

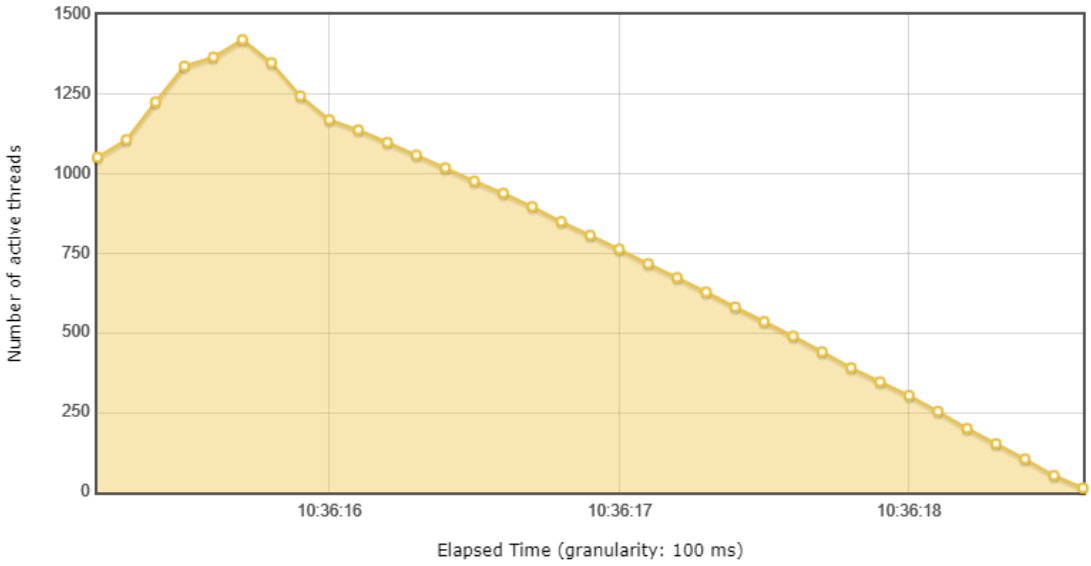
Z grafů 3.15 a 3.16 vyplývá, že zátěžové testování začalo přibližně s 1000 paralelně běžícími vlákny a tento počet se během první půl sekundy vyšplhal až na 1422 aktivních vláken. Od toho okamžiku počet aktivních vláken lineárně klesal až do momentu, kdy byly všechny SIP požadavky ústřednou zpracovány. Zároveň lze vidět, že po první sekundě testování ústředna ještě relativně zvládala zpracovávat přicházející požadavky s dobou odezvy mezi 900 a 1350 milisekundami, ale potom začala doba odezvy lineárně stoupat až k hodnotám 3500 milisekund.

3.5.4 Zatěžování požadavky REGISTER, INVITE a MESSAGE

V posledním připraveném testovaném scénáři byla ústředna zatěžována všemi třemi typy požadavků současně. Každý typ požadavku byl na SIP server odeslán 500 krát, tudíž celkem bylo odesláno 1500 požadavků. Každé vytvořené vlákno nejdříve registrovalo uživatele na ústředně a čekalo na finální odpověď 200 OK. Po obdržení této odpovědi se zahájilo navázaní SIP relace v podobě odeslání INVITE požadavku. Při obdržení odpovědi 200 OK je odeslán požadavek MESSAGE a jakmile je



Obr. 3.15: Vývoj odezvy ústředny při zatěžování požadavky MESSAGE.



Obr. 3.16: Počet aktivních vláken při zatěžování požadavky MESSAGE.

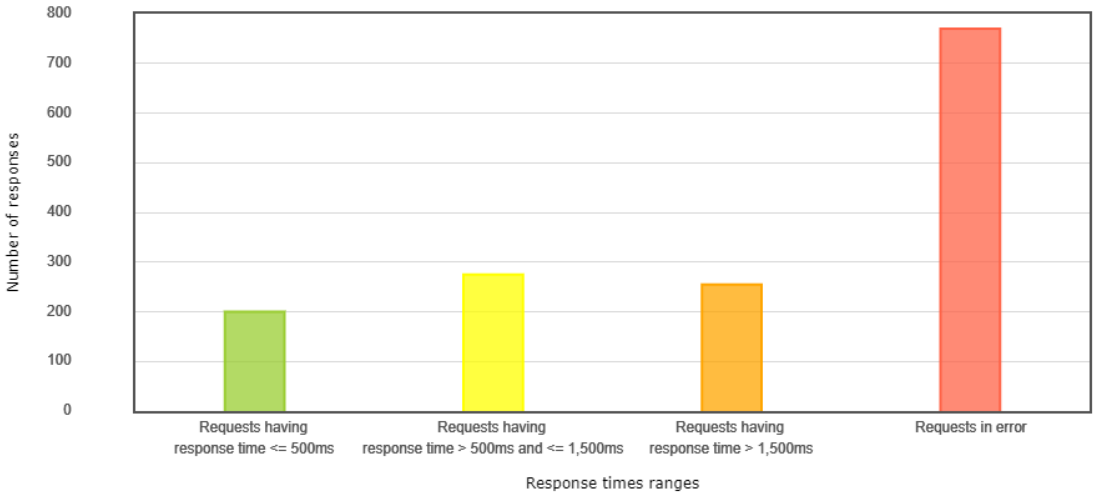
na něj odpovězeno zprávou 202 ACCEPTED, je vlákno ukončeno. Doba čekání na finální odpovědi zůstala nastavena na 3 sekundy pro přehlednější porovnání úspěšnosti s předešlými testovacími scénáři. Celková úspěšnost tohoto scénáře, včetně úspěšnosti jednotlivých SIP požadavků, je shrnuta v tabulce 3.3.

Graf 3.17 potvrzuje vysoké zastoupení chyb z celkového počtu testovaných prvků a to konkrétně 770. Oproti předchozím zátěžovým testováním jsou sloupce úspěšných

Tab. 3.3: Úspěšnost jednotlivých požadavků.

Požadavek	**Počet chyb**	**Úspěšnost** [%]
REGISTER	134	73,2
INVITE	382	23,6
MESSAGE	254	49,2
Celkem	770	48,67

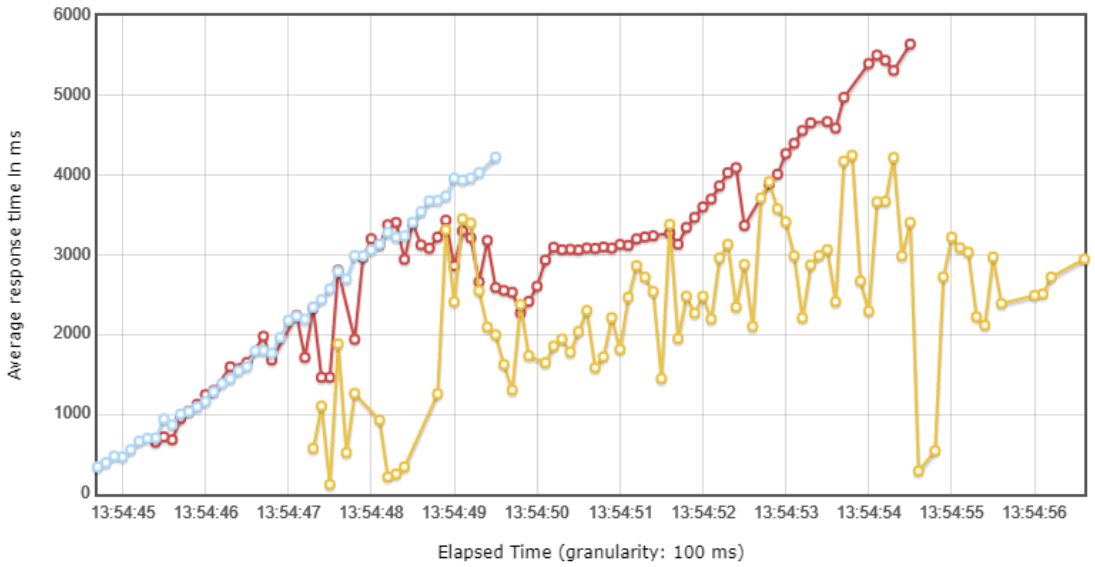
prvků relativně vyrovnány, kdy na 200 prvků, bylo odpovězeno do 500 milisekund, 275 požadavků mělo dobu odezvy mezi 500 a 1500 milisekundami a 255 testovacích prvků bylo s odezvou mezi 1500 a 3000 milisekund.



Obr. 3.17: Přehled odezvy ústředny při zatěžování požadavky REGISTER, INVITE a MESSAGE.

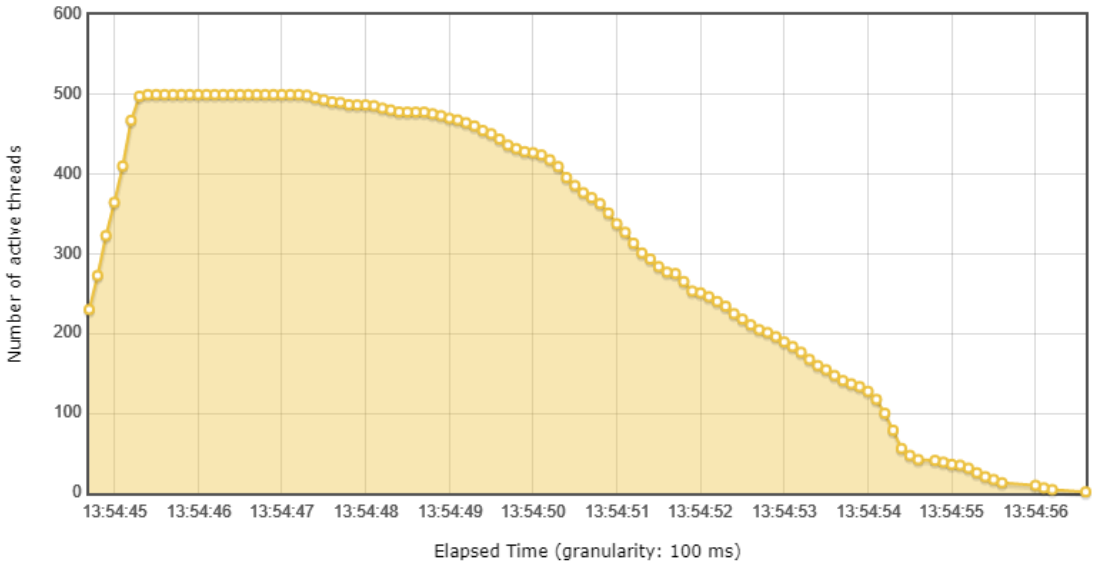
Jak již bylo zmíněno dříve v této kapitole, z grafu 3.18 lze vyčíst, že nejdříve byly na ústřednu odesílány žádosti typu REGISTER, jež jsou značeny modrou křivkou. Jejich doba odezvy měla lineárně rostoucí charakter a průměrná hodnota se vyšplhala až k 4200 milisekundám. První vlákna, která obdržela odpověď od ústředny, což bylo zhruba po půl sekundě, začala okamžitě navazovat SIP relace odesláním požadavku INVITE, jenž je značený červenou křivkou na grafu. Ta měla z počátku taky lineárně rostoucí charakter, ale při dosažení okamžiku, kdy vlákna s největším časovým předstihem začala odesílat i požadavky typu MESSAGE, hodnoty odezvy se začaly výrazně lišit. Nejvyšší průměrná doba odezvy na požadavek INVITE dosáhla hodnoty přes 5600 milisekund. Jelikož se požadavky typu MESAGE, reprezentované žlutou křivkou, začaly odesílat až po požadavcích typu INVITE, ústředna již byla natolik zahlcena veškerým provozem, jak protokolem SIP, tak protokolem

RTP, že doba odezvy na tyto požadavky byla zcela nekonzistentní. Hodnoty odezvy se pohybovaly od 121 až po 4251 milisekund.



Obr. 3.18: Vývoj odezvy ústředny při zatěžování požadavky REGISTER, INVITE a MESSAGE.

Graf 3.19 zobrazuje prvotní nárůst paralelně běžících vláken z 230 až na 500. Tato hodnota zůstala stabilní po dobu 2 sekund, a poté se začala vlákna ukončovat po dobu téměř 10 sekund.



Obr. 3.19: Počet aktivních vláken při zatěžování požadavky REGISTER, INVITE a MESSAGE.

Závěr

Práce se zabývala problematikou zátěžového testování telefonních ústředen. Cílem práce bylo navrhnout a implementovat software využívající program JMeter k zátěžovému testování ústředen VoIP a pomocí tohoto softwaru realizovat sadu zátěžových testů. Výsledky takového testování následně zpracovat do přehledných grafů. Dílčími úkoly bylo seznámení se s protokolem SIP a audio kodeky, které se využívají právě v internetové telefonii a nakonfigurování IVR systému v softwarové ústředně Asterisk.

První kapitola se věnuje protokolu SIP, zkratka pro Session Initiation Protocol. První část je opět věnována obecnému popisu protokolu, jako například co to je nebo proč je potřeba. První podkapitola popisuje základní funkce protokolu, jež jsou seřazeny do přehledné tabulky. Další podkapitola se věnuje zařízením, které tvoří SIP sítě a podkapitola číslo tři popisuje zprávy, jež si tyto zařízení mezi sebou vyměňují. V poslední podkapitole je přiblížena problematika kodeků a je popsáno několik příkladů kodeků. Kodek je software, nebo hardwarové zařízení, které slouží ke komprimaci digitálního signálu a po přenesení po síti zpět do nekomprimovaného signálu pro přehrání.

Druhá kapitola se zaměřuje na problematiku zátěžového testování. Nejdříve je vysvětleno co to zátěžové testování je a proč je dobré ho využívat. V následující podkapitole je testování rozděleno na několik typů. V podkapitole dvě a tři jsou vysvětleny metriky, které se při testování měří, a je popsán obecný postup typického zátěžového testování. V poslední podkapitole jsou analyzovány dostupné nástroje, a to konkrétně program JMeter a nástroj BlazeMeter.

Kapitola tři se věnuje praktické části diplomové práce. Je zde vysvětleno, jak importovat program JMeter do vývojového prostředí IntelliJ IDEA, a je popsána struktura tohoto programu. Ve třetí podkapitole následuje důkladný popis vývoje navrženého modulu pro program JMeter v programovacím jazyce Java a je podrobně popsán kód modulu, jenž je graficky znázorněn v UML diagramech. Tato podkapitola je rozdělena na prvotní nastavení samotného projektu a popis struktury celého projektu, kterou lze rozdělit na logickou část a část grafického uživatelského rozhraní. Závěr podkapitoly se věnuje importování navrženého modulu do programu JMeter. Dále se třetí kapitola zaměřuje na instalaci a konfiguraci pobočkové ústředny Asterisk, jenž slouží jako cíl zátěžového testování, a závěr kapitoly je dedikovaný samotnému testování, včetně vyhodnocení naměřených výsledků. Celkem byla uskutečněna čtyři zátěžová testování, a to zatížení ústředny odesíláním požadavků typu REGISTER, INVITE, MESSAGE a všechny tyto požadavky zároveň. Následně byly výsledky z každého zmíněného testování zpracovány do přehledných grafů a důkladně analyzovány. Při zatížení ústředny pěti set požadavky typu

REGISTER bylo 86,2 % úspěšných, kdežto u zatížení se stejně nastavenými parametry, ale použitými požadavky typu INVITE, byla úspěšnost pouhých 41,4 %. To je způsobeno zahlcením serveru nejenom protokolem SIP, ale také mediálními daty v podobě protokolu RTP. Požadavky typu MESSAGE jsou relativně nenáročné na výpočetní výkon ústředny proti předchozím dvou typům, proto bylo na SIP server vygenerováno tisíc pět set požadavků současně, aby byly ve výsledcích reprezentovány i chybné testovací prvky. Úspěšnost tohoto zatěžování byla 81,93 %. Během posledního testování bylo na ústřednu odesláno pět set požadavků od každého zmíněného typu, tedy tisíc pět set požadavků celkově, s celkovou úspěšností 48,67 %. Jak lze předpokládat, ústředna byla natolik zatížena příchozím provozem, že jednotlivé úspěšnosti každého typu požadavku, byly menší než při testování daného požadavku samostatně. Konkrétní hodnoty jsou zobrazeny v tabulce 3.3.

Literatura

[1] VALDES, Robert a Dave ROOS. *How VoIP Works* [online]. Apr 13, 2021 [cit. 2021-12-06]. Dostupné z: https://computer.howstuffworks.com/ip-telephony.htm

[2] WARREN, Matt. *How do VoIP phones work – a step by step guide* [online]. [cit. 2021-12-06]. Dostupné z: https://www.structuredcommunications.co.uk/how-do-voip-phones-work-a-step-by-step-guide

[3] *Performance and Stress Testing of SIP Servers, Clients and IP Networks* [online]. [cit. 2022-05-16]. Dostupné z: http://startrinity.com/VoIP/TestingSipPbxSoftswitchServer.aspx

[4] MARTIN, Matt. *Guide to VoIP Audio Codecs* [online]. June 23, 2017 [cit. 2021-12-06]. Dostupné z: https://www.nurango.ca/blog/voip-codecs-guide

[5] ROZENFELD, Sam. *Beginners Guide to VoIP Audio CODECs* [online]. October 7, 2019 [cit. 2021-12-06]. Dostupné z: https://www.dls.net/beginners-guide-to-voip-audio-codecs

[6] *What is SIP – Session Initiation Protocol?* [online]. [cit. 2021-12-06]. Dostupné z: https://www.3cx.com/pbx/sip

[7] *SIP Tutorial* [online]. [cit. 2021-12-06]. Dostupné z: https://www.tutorialspoint.com/session_initiation_protocol/index.htm

[8] *What is SIP?: An InteropNet Labs white paper* [online]. May 11, 2004 [cit. 2021-12-06]. Dostupné z: https://www.networkworld.com/article/2332980/lan-wan-what-is-sip.html

[9] DINARDI, Gaetano. *What Is a SIP Proxy? How Does a SIP Server Work?* [online]. [cit. 2021-12-06]. Dostupné z: https://www.nextiva.com/blog/sip-proxy-server.html

[10] PORTER, Thomas. *How to Cheat at VoIP Security* [online]. 2007 [cit. 2021-12-06]. ISBN 9780080553535. Dostupné z: https://www.elsevier.com/books/T/A/9781597491693

[11] *List of SIP response codes* [online]. 20 November 2021 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/List_of_SIP_response_codes

[12] BENEDIKT, Jan. *Automatizovaná tvorba statistik síťového provozu*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 68 s. Vedoucí práce byl Ing. Petr Číka, Ph.D.

[13] HAMILTON, Thomas. *Performance Testing Tutorial: What is, Types, Metrics & Example* [online]. November 10, 2021 [cit. 2021-12-06]. Dostupné z: `https://www.guru99.com/performance-testing.html`

[14] HUDDLESTON, Tom Jr. *Facebook probably lost millions of dollars this week — and the pain might not be over yet, experts say* [online]. Oct 6 2021 [cit. 2021-12-06]. Dostupné z: `https://www.cnbc.com/2021/10/06/facebook-outage-lost-ad-revenue-advertisers-could-seek-refunds.html`

[15] GILLIS, Alexander. *Performance testing* [online]. December 2020 [cit. 2021-12-06]. Dostupné z: `https://searchsoftwarequality.techtarget.com/definition/performance-testing`

[16] *JMeter Tutorial* [online]. [cit. 2021-12-06]. Dostupné z: `https://www.javatpoint.com/jmeter-tutorial`

[17] SHPAK, Kyrylo. *Zátěžový tester*. Brno, 2020, 55 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

[18] GILLIS, Alexander. *BLAZEMETER – A FULL REVIEW OF A PLATFORM FOR LOAD TESTING* [online]. August 18, 2020 [cit. 2021-12-06]. Dostupné z: `https://testmatick.com/blazemeter-a-full-review-of-a-platform-for-load-testing`

[19] *Download Apache JMeter* [online]. [cit. 2021-12-06]. Dostupné z: `https://jmeter.apache.org/download_jmeter.cgi`

[20] ROSENBERG, J., H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY a E. SCHOOLER. *SIP: Session Initiation Protocol* [online]. 2002 [cit. 2022-05-15]. Dostupné z: `https://datatracker.ietf.org/doc/html/rfc3261`

[21] NEWTON, Rusty, DAVENPORT, Malcolm, ed. *Res_pjsip Configuration Examples* [online]. Apr 27, 2018 [cit. 2021-12-06]. Dostupné z: `https://wiki.asterisk.org/wiki/display/AST/res_pjsip+Configuration+Examples`

Seznam symbolů a zkratek

SIP	Session Initiation Protocol
VoIP	Voice over Internet Protocol
IVR	Interactive Voice Response
PSTN	Public Switched Telephone Network
DDoS	Distributed-Denial of Service
RTP	Real-Time Transmission Protocol
IP	Internet Protocol
HTTP	HyperText Transfer Protocol
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ISO/OSI	International Standards Organization Open Systems Interconnection
SDP	Session Description Protocol
IETF	Internet Engineering Task Force
RFC	Request For Comment
URI	Uniform Resource Identifier
MD5	Message-Digest 5
PCM	Pulse-Code Modulation
MOS	Mean Opinion Score
ITU	International Telecommunication Union
LAN	Local Area Network
ADPCM	Adaptive Differential Pulse-Code Modulation
DoS	Denial of Service
CNBC	Consumer News and Business Channel

URL	Unified Resource Locator
API	Application Programming Interface
JAIN	Java APIs for Integrated Networks
UML	Unified Modeling Language
GUI	Graphic User Interface
ISDN	Integrated Services Digital Network
DAHDI	Digium Asterisk Hardware Device Interface
PBX	Private Branch eXchange
ICT	Information and Communications Technology
CPU	Central Processing Unit

A Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého CD
  - `jmeter-sip-plugin` ..... kořenový adresář IntelliJ projektu s vlastním modulem
    - `.git` ..... kořenový adresář pro git
    - `.idea` ..... kořenový adresář pro nastavení projektu (IntelliJ IDEA)
    - `necessary_libs` adresář obsahující knihovny nezbytné pro správné fungování modulu
    - `src` ..... adresář obsahující zdrojové soubory modulu
    - `target` ..... vygenerovaný adresář programem Maven
    - `.gitignore` ..... soubor specifikující záměrně nesledované objekty gitem
    - `pom.xml` ..... konfigurační soubor Maven projektu
    - `README.md` ..... soubor obsahující základní informace o projektu
  - `Konfigurační soubory Asterisk`
    - `extensions.conf` ..... konfigurační soubor klapek SIP serveru
    - `pjsip.conf` ..... konfigurační soubor SIP uživatelů