VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. Tomáš Srnec

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

TESTOVÁNÍ PROTOKOLŮ PRO VIDEO NA VYŽÁDÁNÍ V PROGRAMU APACHE JMETER

VIDEO ON DEMAND PROTOCOLS TESTING USING APACHE JMETER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Srnec**
AUTHOR

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

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Tomáš Srnec ***ID:*** 164783
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Testování protokolů pro video na vyžádání v programu Apache JMeter

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je implementace zásuvného modulu do programu Apache JMeter. Zásuvný modul bude uživateli umožňovat testování služby video na vyžádání prostřednictvím různých standardů, například HLS (HTTP Live Stream), RTSP (Real Time Streaming Protocol) a dalších. V teoretické části práce se seznamte se streamováním multimediálních dat různými technikami. Dále se seznamte se zátěžovými testy využívanými pro testování serverů s video obsahem. V rámci praktické části navrhněte a implementujte zásuvný modul do programu Apache JMeter vhodný pro zátěžové testování a testování komunikace dle různých standardů.

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: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

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

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

ABSTRAKT

Diplomová práce se zabývá testováním aplikačních protokolů HLS a RTSP v programu JMeter. Cílem práce je pro tyto protokoly navrhnout a zrealizovat testovací moduly, pomocí kterých se provedou zátěžové testy. V první části práce jsou popsány typy zátěžových testů, testovací program JMeter a služba video na vyžádání. Další část popisuje použité protokoly, zejména HLS a RTSP, které jsou v práci použity. Praktická část obsahuje návrh a implementaci testovacích modulů včetně testovacích plánu a jejich realizaci pro vytvořené servery. Na závěr jsou zpracovány a okomentovány výsledky.

KLÍČOVÁ SLOVA

zátěžové testy, JMeter, sampler, video na vyžádání, HLS, HTTP, RTSP, RTP, segment, playlist

ABSTRACT

The master's thesis deals with testing the application protocol HLS and RTSP in JMeter program. The aim of this thesis is to design and implement a test modules for both protocols, which will perform stress tests. The first part of thesis describes the types of stress tests, JMeter program for performance testing and video on demand services. Next chapter describes selected protokols, especially HLS and RTSP, which are used in this thesis. The practical part contains the design and implementation of test modules including test plans. Finally, the results are processed and commented.

KEYWORDS

performance testing, JMeter, sampler, video on demand, HLS, HTTP, RTSP, RTP segment, playlist

SRNEC, Tomáš. *Testování protokolů pro video na vyžádání v programu Apache JMeter*. Brno, Rok, 63 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Číka Petr, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Testování protokolů pro video na vyžádání v programu Apache JMeter“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) 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(-ky)

PODĚKOVÁNÍ

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

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

Problematika zátěžového testování je úzce spjata s návrhem každého programu. Kvalitní komerční řešení si žádá alespoň nějakou formu zátěžového testování, což platí zvláště pro multimediální služby. Jejich podíl na celkovém objemu přenesených dat v internetu rok od roku roste a i prognostiky předních firem tento trend potvrzují. Často skloňovanými jsou termíny video na vyžádání a přenos dat v reálném čase.

Cílem této práce je navrhnout a zprovoznit moduly v testovacím programu JMeter, které umožní zátěžově testovat protokoly HLS a RTSP. V první části jsou popsány tři typy zátěžového testování a rozdíly mezi nimi. Druhá část popisuje nástroj JMeter a jeho moduly pro vytvoření testovacího plánu. Další kapitola vysvětluje princip videa na vyžádání. Ve čtvrté kapitole jsou teoreticky popsány protokoly potřebné k pochopení a implementaci modulů. Praktická část seznamuje s návrhem a implementací zadaných modulů. Nedílnou součástí je i vytvoření vlastních serverů, které se vytvořenými samplery testují. V závěru jsou okomentovány výsledky několika měření, nejprve u HLS protokolu, následně u RTSP protokolu.

1 ZÁTĚŽOVÉ TESTOVÁNÍ

Zátěžové testování slouží zejména k ověření funkčnosti aplikace pří různé velikosti provozu (zatížení). V této kapitole jsou popsány tři základní typy zátěžového testování a to testování výkonu (performance test), zátěžový test (load test) a test hraniční zátěže (stress test).

1.1 Testování výkonu

Cílem testování výkonu je zjistit, jak se daný systém chová při stanovené zátěži. Nejčastěji jsou sledovány parametry jako propustnost, spolehlivost, škálovatelnost a schopnost odpovídat. [1]
Testování výkonu se provádí zejména pro docílení následujícího:

- nalezení zdroje problémů,
- zhodnocení připravenosti produktu ke komerčnímu nasazení,
- porovnání charakteristik více systémů a konfigurací,
- nalezení úrovně propustnosti.

1.1.1 Základní postup při testování výkonu

Testováním výkonu se hledají slabá místa aplikace, vytváří výchozí nastavení pro další testování a konfiguraci aplikace. Z výsledků můžeme vhodně navrhnout i hardwarovou část aplikace. Body níže popisují postup práce při tomto typu testování.

1. **Identifikace testovacího prostředí** - Identifikace testovaného hardwaru, softwaru a síťové konfigurace pro správné nastavení testovacího plánu a zlepšení efektivity testování. V některých situacích je nutné aktualizovat v průběhu různých fází projektu.
2. **Stanovení požadovaného výkonu aplikace** - Určení parametrů aplikace jako jsou propustnost, doba odezvy a množství hardwarových prostředků. Mohou se použít i další parametry, které pomohou dosáhnout co nejlepší výkon aplikace.
3. **Vytváření testovacího plánu** - Nastavení scénářů, určení proměnlivosti na reprezentativním vzorku uživatelů, definice dat a metrik k testování.
4. **Nastavení testovacího prostředí** - Příprava testovacího prostředí a nástrojů pro sledování systémových zdrojů.
5. **Implementace testovacího plánu** - Zakomponování testovacího plánu do testovacího softwaru.

6. **Testování** - Spuštění testu a monitorování testovaného objektu. Současně se sbírají data jako například odezva, pro budoucí zhodnocení.
7. **Analýza výsledků a opětovné testování** - Vyhodnocení nasbíraných dat pro odstranění slabých míst systému. Pokud je to možné, upraví se parametry testovacího objektu tak, aby při opakovaném testování byly měřené hodnoty ve stanovených limitech.

1.2 Zátěžový test

Zátěžový test je proces hodnocení chování testovaného systému pod zátěží s cílem odhalit chyby související se zatížením. Míra s jakou jednotlivé služby přistupují k testovanému systému se říká zátěž. Chyby, které se snažíme tímto testováním odhalit mohou být chyby funkcionality, jenž se projevují pouze pod zátěží (např. deadlock, přetečení vyrovnávací paměti nebo únik paměti). Dalším typem jsou chyby projevující se zhoršenou kvalitou služby, kterou testovaný systém poskytuje (např. spolehlivost, robustnost nebo stabilita). [2]

1.2.1 Dva možné přístupy k zátěžovému testu

Cílem zátěžového testu je vytvořit takovou zátěž, která dokáže odhalit chyby systému. V závislosti na požadovaných výsledcích jsou možné dva přístupy:

1. **Navrhnutí realistické zátěže** - Hlavním cílem testování zátěže je, aby systém při nasazení do ostrého provozu fungoval správně. Proto když systém zvládne nastavenou zátěž bez chyb funkcionality a zhoršení kvality služeb, obstojí dobře v provozu. Tento typ je obecnější, ale testování může zabrat delší dobu. Stejně tak vyhodnocení výsledků zabere delší dobu z důvodu většího množství dat.
2. **Navrhnutí zátěže generující poruchy** - Dalším přístupem je vygenerování většího množství zátěže, což způsobí u testovaného systému chybové stavy. V porovnání s realistickou zátěží, test bývá vetšinou deterministický s lépe čitelnými výsledky. Čas potřebný k testování bývá kratší, zvlášť pokud si nastavíme ukončení testu po první detekci chyby.

1.3 Test hraniční zátěže

Test hraniční zátěže (Stress test) je způsob testování systému, při kterém je zatížen nadměrným množstvím zátěže, než na který byl navrhován, s cílem jej poškodit. Test hraniční zátěže může donutit systém k vyřazení některých jeho zdrojů jako například RAM, paměťový disk,... Správně nastavený test by měl v systému vyvolat chybové

stavy, čímž se zároveň testuje schopnost obnovení poruchy. Očekává se, že systém nebude zpracovávat přetížení bez odpovídajících zdrojů, ale chovat se (např. selhat) rozumným způsobem (např. nepoškozování nebo ztráta dat). [3]

Zatímco předchozí dva testy vyžadovaly kontrolované prostředí a pravidelně se opakující měření, stress test využívá chaos a nepředvídatelnost. Jako příklad si můžeme vzít webovou aplikaci, na kterou je aplikován stress test těmito způsoby [4]:

- zdvojnásobení běžného množství uživatelů/Hypertextový transportní protokol, HTTP (Hypertext Transfer Protocol) připojení,
- náhodné vypínání a restartování portů na přepínači/směrovači připojeného k serveru,
- náhlý výpadek databáze případně její restart,
- spuštění dalších nesouvisejících procesů na Web a databázovém serveru,
- překonfigurování RAID pole za chodu.

1.4 Porovnání zátěžových testů

V tabulce níže 1.1 a 1.2 je uvedeno shrnutí a nevýhody všech tří zmíněných testů[1].

Tab. 1.1: Porovnání zátěžových testů.

Název	**Popis**
Testování výkonu	**Popis:** • Určuje či ověřuje nastavení rychlosti, stability a škálovatelnosti systému. • Pomáhá vývojářům odhadnout, jak bude uživatel spokojený s výkonem systému. • Pomáhá při úpravách, plánování kapacit a optimalizaci. • Odhaluje rozdíly mezi očekávaným a skutečným výkonem systému. **Nevýhody:** • Neodhaluje chyby které se objeví pouze pod zátěží. • Pokud není pečlivě naplánován, je platný pouze pro malé množství scénářů.

Tab. 1.2: Pokračování tabulky Porovnání zátěžových testů.

Název	**Popis**
Zátěžový test	**Popis:** • Prověřuje aplikaci při normálním a špičkovém zatížení. • Sbírá data pro zlepšení škálovatelnosti a plánování kapacity. • Pomáhá určit jaký je nejvyšší počet uživatelů, který ještě neovlivní hladký chod systému. • Určuje maximální zátěž kterou jsou schopná hardwarové prostředky zvládnout. **Nevýhody:** • Není navržen tak, aby se primárně soustředil na rychlost odezvy. • Výsledky by měly být použity pouze pro srovnání se souvisejícími zátěžovými testy.
Test hraniční zátěže	**Popis:** • Prověřuje systém za hranicí normálního a špičkového zatížení. • Zjišťuje zda-li se poškodí data přetížením systému. • Pomáhá nastavit monitorovací systémy pro varování před přetížením. • Odhaluje se kterými chybami se musí nejpravděpodobněji počítat. • Poskytuje odhad, kam za hranice cílové zátěže může systém dojít předtím než se dostane do chybového stavu. **Nevýhody:** • Je složité odhadnout velikost generovaného zatížení. • Přetížením mohou vzniknout trvalé chyby jednak uvnirř systému, tak i na celé síti.

2 NÁSTROJ JMETER

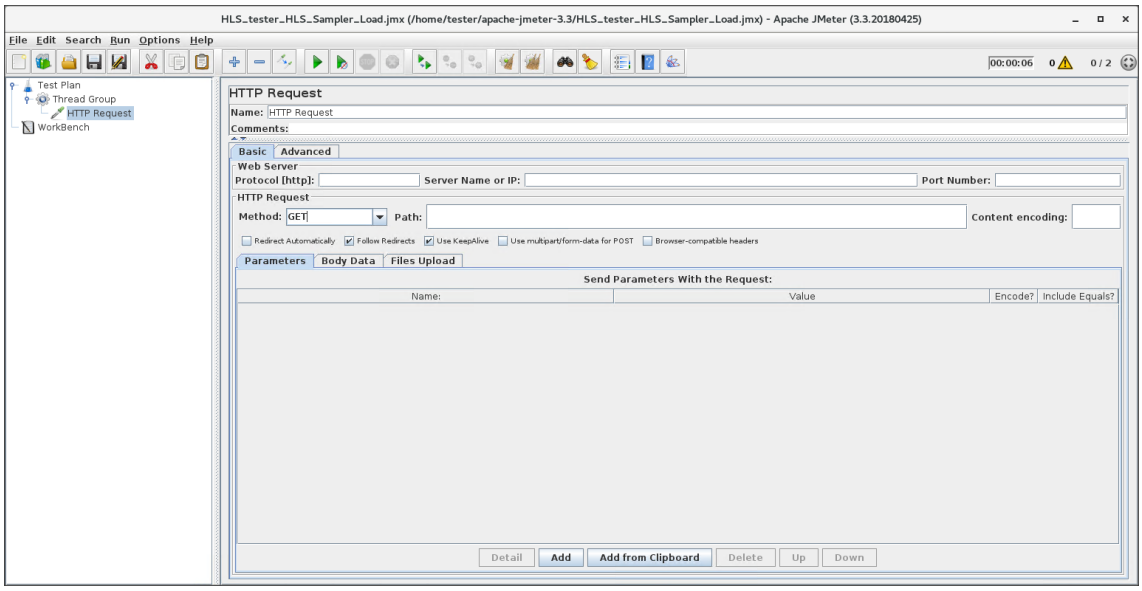
JMeter je desktopová aplikace určená k testování a měření výkonu a funkčního chování aplikací klient/server jako jsou webové aplikace, FTP, JDBC, LDAP a další. Jedná se o jednu z nejpoužívanějších volně šiřitelných testovacích aplikací. Je založena výhradně na jazyce Java a je vysoce rozšířitelná prostřednictvím API (Application Programming Interface).

JMeter zastává funkci klienta v rámci klient/server aplikace. Měří dobu odezvy a všechny ostatní serverové zdroje, jako je zatížení CPU, využití paměti a využití zdrojů. [14]

2.1 Testovací plán a jeho prvky

Testovací plán definuje a poskytuje rozvržení jak a co testovat. Lze jej chápat jako kontejner pro běžné testy. Poskytuje framwork, ve kterém se provádí sled operací nutných k provedení testu. Testovací plán obsahuje tyto prvky:

- **Thread Group** - Je základním prvkem testovacího plánu. Obsahuje všechny ostatní prvky JMeteru. Jeho hlavním úkolem je vytváření a správa paralelně běžících vláken po celou dobu testu, kde jedno vlákno odpovídá jednomu uživateli. Dále je zde možné nastavit dobu náběhu na určený počet uživatelů a počet opakování celého testu. Prvek Thread Group se dá nahradit jeho vylepšenou verzí s názvem Ultimate Thread Group. Ta navíc obsahuje grafické rozhraní pro vytvoření schodovitého provozu s libovolným počtem náběhů.
- **Samplers** - Definují požadavky, které lze odeslat na server. Simulují žádost uživatele, například o stažení webové stránky. Každý Sampler ukládá záznam o průběhu zpracování požadavku. Nejčastěji se zaznamenává výkon, uplynulý čas, propustnost, odezva a další. Ve výchozím nastavení odesílá JMeter požadavky v pořadí, v jakém jsou umístěny do Thread Group. Pořadí však může být upraveno pomocí Logic Controllerů. Pokud to Sampler umožňuje, lze jeho nastavení upravit více pomocí prvku Configuration Element. Ukázka Sampleru na obrázku 2.1
- **Logic Controllers** - Umožňují definovat pořadí Samplerů uvnitř Thread Grupy. Logic Controller může měnit pořadí svých vnořených prvků, podmíněně spouštět, ukončovat apod.
- **Listeners** - Slouží k zobrazení výsledků zachycených v Sampleru formou tabulek, grafů nebo dat v textové podobě. Výsledky se zobrazují buď v reálném čase nebo po ukončení Testovacího plánu. Listener zpracovává pouze data prvků ze stejné nebo nižší úrovně. Při velkém počtu vláken představují Listenery značnou zátěž pro paměť.



Obr. 2.1: Ukázka HTTP Sampleru a základního testovacího plánu.

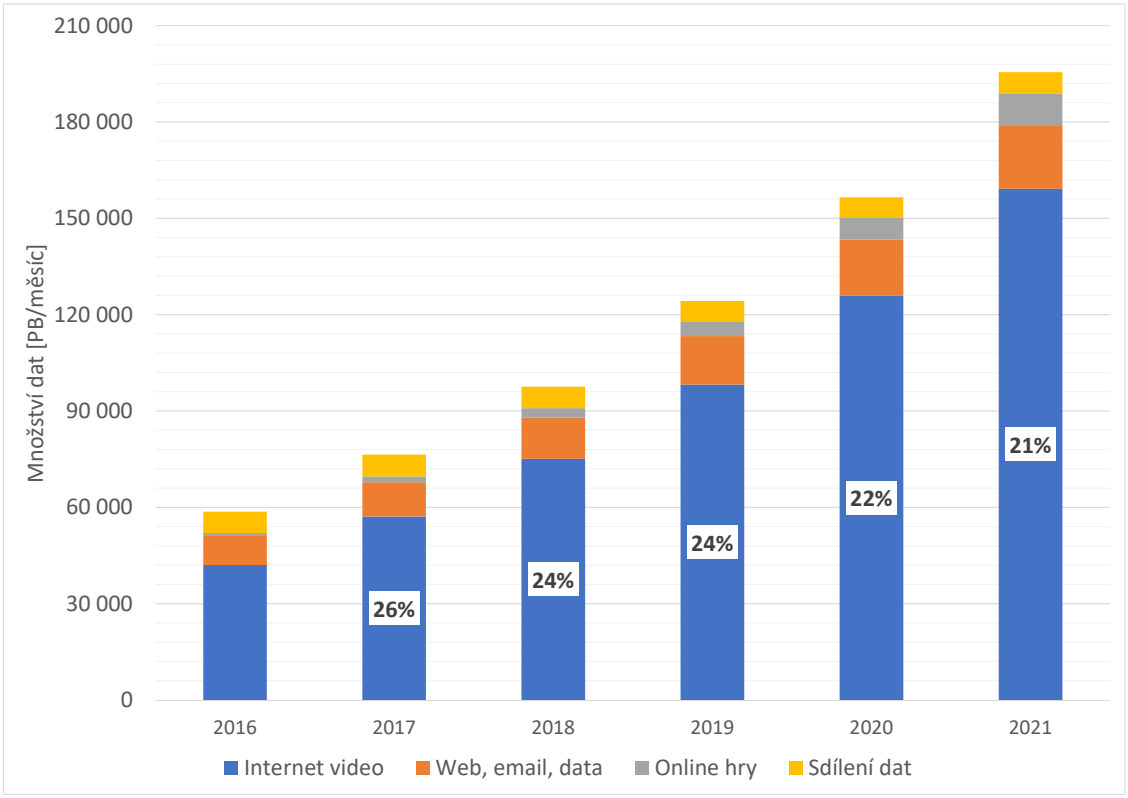
- **Timers** - Prvek Timer způsobí, že se JMeter mezi dvěma po sobě následujícími žádostmi z Thread Groupy na určitou dobu zastaví. Ve výchozím stavu se žádosti posílají okamžitě za sebou, což může způsobit zastavení serveru. Použitím Timeru se zamezí riziku snížení jeho výkonu což je užitečné při testování výkonu.
- **Assertions** - V testovacím plánu slouží jako test na ověření odpovědi na žádost vygenerovanou pomocí Sampleru. Jsou vloženy jako jeho podřízená část. Podobně jako Timery se i Asserions používají hlavně při testování výkonu a funkčnosti, pro zajištění korektnosti přijímaných odpovědí.
- **Configuration Elements** - Umožňují konfigurovat proměnné a konstanty využívané Samplery. Přidávají se na začátek Thread Groupy, jejíž jsou součástí. Configuration Elements se doporučují použít zejména v případě, kdy využíváme několik stejných Samplerů a jejich jednotlivá konfigurace by byla příliš zdlouhavá.
- **Pre-Processor Elements** - Jsou podřízeným prvkem Sampleru a slouží k modifikaci jeho parametrů před vytvořením výsledného požadavku.
- **Post-Processor Elements** - Obdobně jako Pre-Processor Elements je i Post-Processor Elements podřízeným prvkem Sampleru, s tím rozdílem, že je spuštěn až po vytvoření požadavku Sampleru. Nejčastěji je používán pro zpracování odpovědí, například uložení hodnoty z odpovědi pro pozdější použití.

3 VIDEO NA VYŽÁDÁNÍ

Video na vyžádání, VoD (Video on Demand) je systém, který umožňuje uživateli si poslechnout/zhlédnout jeho oblíbený obsah, kdykoliv chce, na rozdíl od pozemního televizního vysílání, kdy je divák závislý na aktuálně vysílaném programu. Pro přenos VoD se nejčastěji používá technologie Internetové televize, IPTV (Internet Protocol Television). VoD se rozděluje na dva systémy:

- **stream** - Posílání obsahu v reálném čase přes set-top box, počítač, nebo další zařízení k uživateli.
- **stáhnutí na uložiště** - Obsah se stáhne do uložiště a je přístupný kdykoliv.

Objem dat přenášených po sítí každým rokem stoupá. Největší podíl na tom má právě přenos videa, díky lepší dostupnosti a rozšířenosti chytrých telefonu nebo neustálým zrychlováním Internetového připojení. Největší digitální přehrávače jako Netflix, Amazon nebo Apple TV velmi výrazně ohrožují klasické televizní kanály, které byly doteď hlavním zdrojem zábavy. [5]

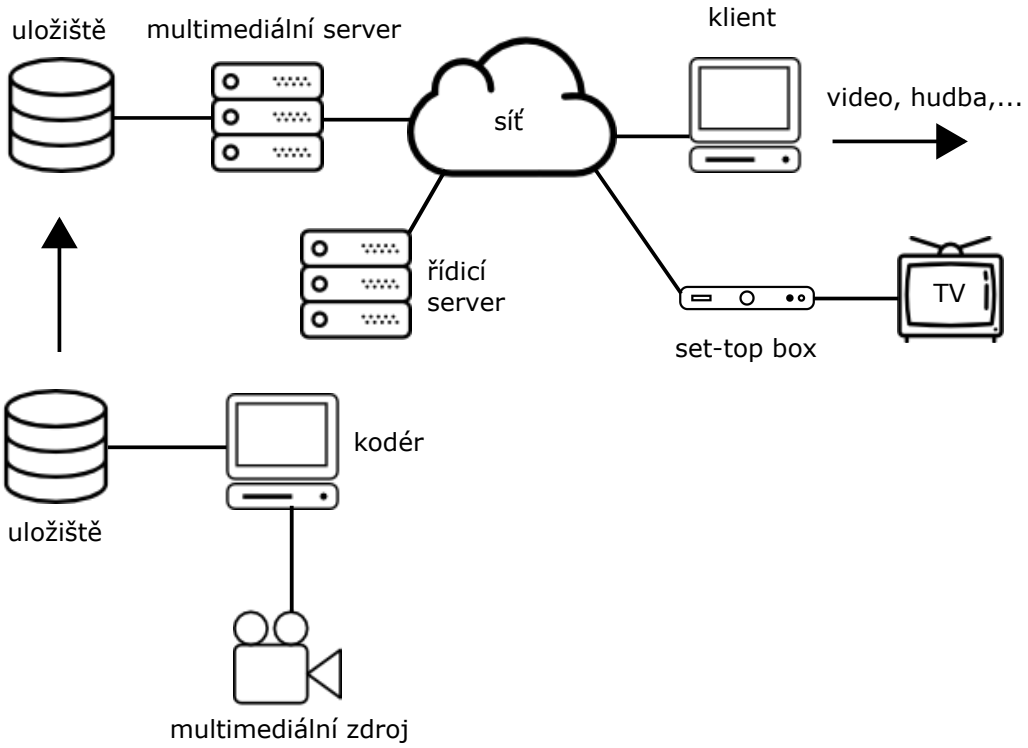


Obr. 3.1: Globální předpověď intenetového provozu, CISCO.

V grafu 3.1 je předpověď globálního provozu do roku 2021 od společnosti CISCO [6]. Jak je patrné, každý rok se objem přeneseného Internetového videa zvětší téměř

o čtvrtinu. Velkou část z toho tvoří právě Video na vyžádání.

Princip jakým Video na vyžádání funguje je znázorněn na obrázku níže 3.2. Zaznamenaný multimediální obsah se kodérem zpracuje do požadovaného formátu a uloží se na úložiště. Požadavky od klienta v podobě počítače, či televize se po síti dostanou k řídícímu serveru, který vydá pokyn multimediálnímu server pro zahájení přenosu obsahu uživateli. Ten si obsah dekóduje buď sám, nebo za pomocí set-top boxu [7].



Obr. 3.2: Obecný přehled systému Videa na vyžádání.

3.1 Internetová televize

IPTV poskytuje digitální televizní služby pomocí Internetového protokolu (IP). Přináší služby jako VoD, triple play (televize, Internet a VoIP telefon) a další, kde kvalita služby je zajištěna pomocí QoS. Správné nastavení QoS je klíčové pro celé IPTV, zvlášť na nízkokapacitních linkách. Výhoda oproti klasickému televiznímu vysílání, je menší nárok na šířku pásma, kdy se nepřenáší všechny kanály, ale uživatel si až na koncovém zařízení vybere co ho zajímá a tento obsah se přenese. [8]

4 POUŽITÉ PROTOKOLY

V této kapitole jsou popsány protokoly aplikační vsrtvy TCP/IP modelu, které jsou v práci použity.

4.1 Hypertextový přenosový protokol

HTTP je protokol aplikační vrstvy pro distribuční systémy sloužící k přenosu hypertextových dokumentů ve formátu HTML. HTTP se začalo používat v roce 1990 s globální sítí World-Wide Web. První verze protokolu sloužila pouze k přenosu nezpracovaných dat po Internetu. Verze HTTP/1.0 přinesla rozšíření MIME, díky kterému umí přenášet jakýkoliv soubor a používá se společně s formátem XML pro webové služby (spouštění vzdálených aplikací). HTTP se také používá jako obecný protokol pro komunikaci mezi uživatelskými agenty a proxy servery/bránami k jiným Internetovým systémům, včetně SMTP, NNTP, FTP,... Tímto způsobem HTTP umožňuje dalším aplikacím přístup k hypermédiím. [9]

HTTP používá jednotný lokátor prostředků URL, který jednoznačně specifikuje umístění zdroje v Internetu. Jeho vzor je zobrazený níže.

```
http_URL = "http:""//"host[":"port][cesta["?"dotaz]]
```

Protokol HTTP je typu požadavek/odpověď. Klient pošle serveru požadavek, který musí obsahovat dotazovací metodu, URI, verzi protokolu a MIME zprávu. Server reaguje stavovým řádkem s verzí protokolu, chybovou nebo potvrzující zprávou a MIME zprávou s daty.

Většina HTTP komunikace inicializuje uživatel s požadavkem na data, ležící na HTTP serveru. Jako transportní protokol je použitý TCP. Výchozí port je 80.

4.1.1 Dotazovací metody

HTTP definuje několik metod označující požadovanou akci, která má být provedena na identifikovaném zdroji. Co představuje tento zdroj, zda již existující data nebo data, která jsou generována dynamicky, závisí na implementaci serveru. Často zdroj odpovídá souborem nebo výstupem spustitelného souboru umístěného na serveru.

- **GET** - Tato metoda žádá o výslednou reprezentaci zdroje, nikoliv jeho procesních částí. Je to výchozí metoda při požadavku na zobrazení hypertextových stránek.
- **HEAD** - Metoda podobná GET, ale nepředává žádná data, pouze metadata o cíli.
- **POST** - Slouží k odesílání uživatelských dat na server. Narozdíl od požadavku GET zvládá velký objem dat.

- **PUT** - Metoda na nahrávání dat na server. Nepoužívá se, je nahrazena zejména FTP.
- **DELETE** - Smaže požadovaný objekt ze serveru.
- **TRACE** - Server po přijetí tohoto dotazu pošle uživateli zprávu zpátky, takže může vidět co na požadavku mění servery, kterými prochází.
- **OPTIONS** - Uživatel dotazuje server na podporované metody.
- **CONNECT** - Vytvoření spojení pomocí TCP/IP tunelu. Nejčastěji se používá k vytvoření zabezpečeného kanálu pro komunikaci (HTTPS).

4.1.2 Zabezpečený Hypertextový přenosový protokol

Zabezpečený Hypertextový přenosový protokol, HTTPS, (Hypertext Transfer Protocol Secure) je Internetový komunikační protokol, který chrání integritu a důvěryhodnost dat přenášených mezi počítačem a sítí. HTTPS umožňuje uživatelům bezpečně a privátně surfovat po Internetu. [10]

Posílaná data jsou zabezpečena pomocí TLS/SSL protokolu. Navázání spojení se dá shrnout do těchto tří kroků[11]:

1. **Hello zpráva** - Klient posílá zprávu serveru obsahující potřebné informace pro sestavení spojení, jako například verzi, nebo šifrovací systém. Server požadavek potvrdí s vybranými parametry.
2. **Výměna certifikátu** - Server prokazuje svoji identitu klientovi. Tento krok je dosáhnut pomocí SSL certifikátu, jenž obsahuje mimo jiné veřejný klíč, digitální podpis a datum platnosti certifikátu. Klient si ověří zda-li věří certifikátu implicitně, nebo je web ověřen pomocí známé certifikační autority. Ve vzácných případech může chtít server certifikát i po klientovi, hlavně ve velmi důvěrných aplikacích.
3. **Výměna klíče** - Šifrování dat přenášených mezi serverem a klientem probíhá pomocí symetrické šifry. Druh šifry je dohodnout v prvním kroku. Symetrická šifra používá jen jeden klíč pro šifrování a dešifrování, na rozdíl od asymetrické šifry, kde jsou klíče v páru a jeden je veřejný, druhý soukromý.

Klient vygeneruje nahodný klíč pomocí symetrické šifry a zašifruje jej veřejným klíčem serveru, který nalezl v certifikátu. Server tuto zprávu příjme a dešifruje pomocí svého soukromého klíče. Tímto je ověřování dokončeno. Protokol HTTPS běží na portu 443 a využívá transportní protokol TCP.

4.1.3 Přenos v reálním čase pomocí Hypertextového přenosového protokolu

Přenos v reálním čase pomocí Hypertextového přenosového protokolu, HLS, (HTTP Live Streaming) poskytuje spolehlivý, nákladově nenáročný způsob poskytování nepřetržitého a dlouhotrvajícího videa posílaného přes Internet. Umožňuje příjemci upravovat přenosovou rychlost média v závislosti na síťových podmínkách, za účelem nepřerušovaného přehrávání v co nejlepší možné kvalitě. [12]

HLS poskytuje flexibilní rámec pro šifrování médií. Může efektivně nabízet vícenásobné ztvárnění stejného obsahu, například audio překlady. Dostupnost pro širokou veřejnost získává díky kompatiblitě s HTTP protokolem, který je hojně rozšířený. Od svého prvního vydání v roce 2009 byl HTTP Live Streaming implementován a rozvinutý velkou řadou distributorů, vývojářů a výrobců zařízení. Protokol je neustále vylepšován na základě zpětné vazby od provozovatelů streamovacích služeb.

4.1.4 Media playlist

Playlist v pojetí protokolu HLS je UTF-8 textový soubor, obsahující URI adresy a popisné značky. Playlist obsahuje segmenty média (chunky), které se postupně na vyžádání přehrávají. Ukázka playlistu je níže.

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXTINF:9.009,
http://media.example.com/first.ts
#EXTINF:9.009,
http://media.example.com/second.ts
#EXTINF:3.003,
http://media.example.com/third.ts
```

První řádek je značka identifikace formátu `#EXTM3U`.
Značka `#EXT-X-TARGETDURATION` říká, že všechny segmenty musí mít 10 vteřin nebo míň. Následuje deklarace tří segmetnů, první a druhý je dlouhý 9,009 vteřiny, třetí 3,003 vteřiny.

Pro přehrání playlistu si jej klient stáhne a spustí deklarované segmenty. Klient playlist postupně aktualizuje, aby zjistil jestli se počet segmentů nezměnil. Data by měla být přenášena přes protokol HTTP, ale mohou se použít i ostatní protokoly umožňující přenos dat na vyžádání.

Master playlist je nadřazená složka playlistu obsahující různé varianty stejného obsahu. Ty se mohou lišit kódováním, přenosovou rychlostí, formátem, nebo rozlišením.

Playlist může také obsahovat různé interpretace obsahu. Například výběr dabingu u filmu, či video nahrané z více úhlů kamery. Klient si pak vybírá v závislosti na jeho preferencích a síťových podmínkách.

4.1.5 Media segmenty

Media playlist v sobě obsahuje URI adresy s Media segmenty, které tvoří žádaný obsah. Trvání každého segmentu je specifikováno pomocí `EXTINF` značky. Všechny segmenty mají v playlistu definované pořadové číslo. Standardně se první segment označí jako 0, další 1 a tak dále. Následující segment musí navazovat na bitový tok segmentu předchozího, v závislosti na pořadovém čísle. Hodnoty časového razítka a počítadel kontinuity musí zůstat nepřerušeny. Jedinou výjimkou je první segment a segment, který je označkován jako nespojitý. Neoznačené nespojité segmenty vyvolají playback chybu.

Všechny Media segmenty obsahující video by měly obsahovat dostatek informací pro video dekodér, aby mohl dekódovat průběžně všechny snímky od prvního po poslední.

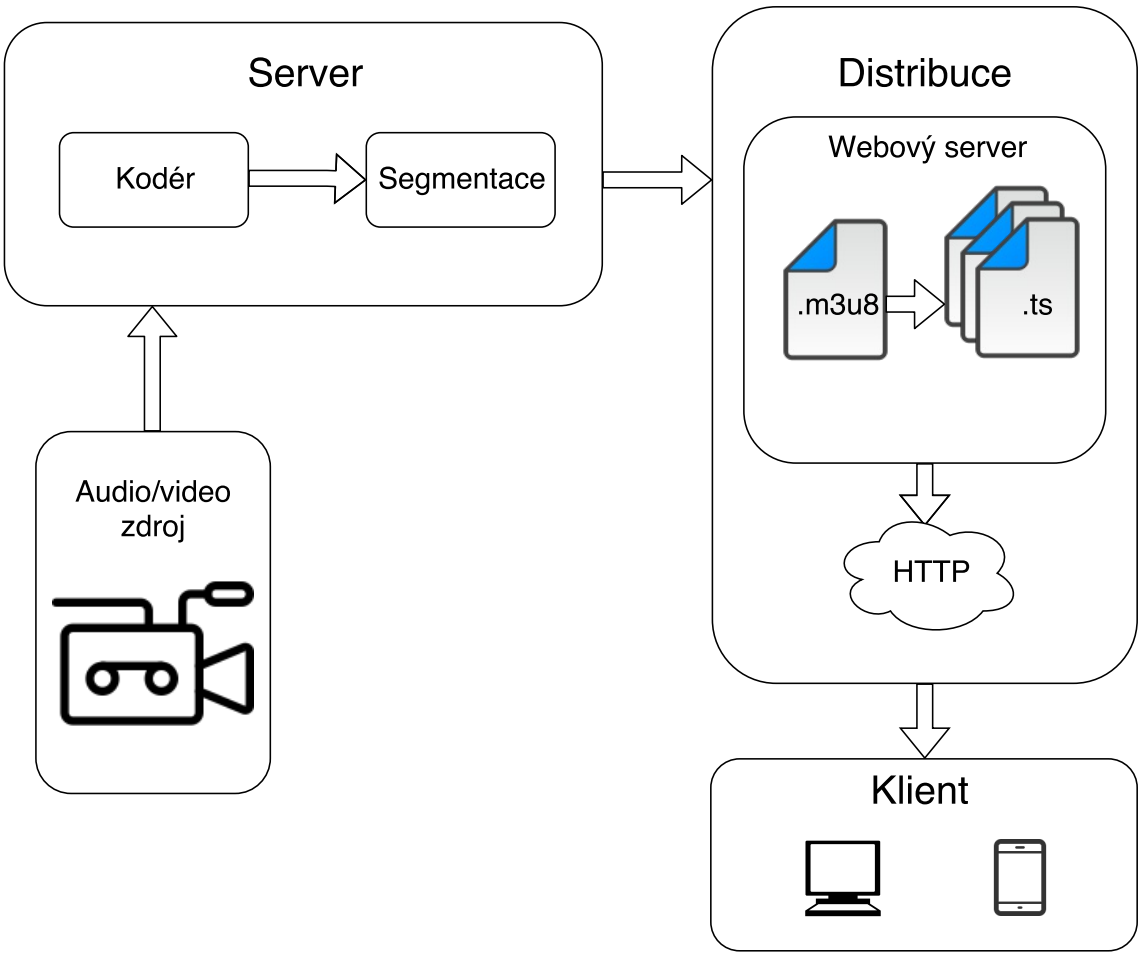
4.1.6 Streamovací architektura HTTP

HTTP Live streaming se skládá ze tří částí: server, distribuce a klient.

- Úkoly serveru jsou zakódovat přijatý audio/video vstup, zabalit jej do formátu vhodného pro přenos a připravit pro distribuci.
- Distribuce je v podobě klasických webových serverů. Ty jsou odpovědny za přijímání požadavků od klientů a posílání připraveného media zpět.
- Klient se odpovědný za zvolení vhodného média, jeho stáhnutí a sestavení do podoby, ve které se může kontinuálně přehrávat.

V klasické konfiguraci, hardwarový kodér příjme audio/video soubory, zakóduje je jako H.264 a AAC kodek a pustí je pomocí MPEG-2 transportního streamu do segmentačního softwaru, který vytvoří segmenty. Tyto soubory jsou potom umístěny na webový server. Segmentační software vytvoří i playlist, který taktéž umístí na web. Klient si od webového serveru vyžádá playlist odkud je schopen stahovat jednotlivé segmenty a zobrazit je bez přerušení.

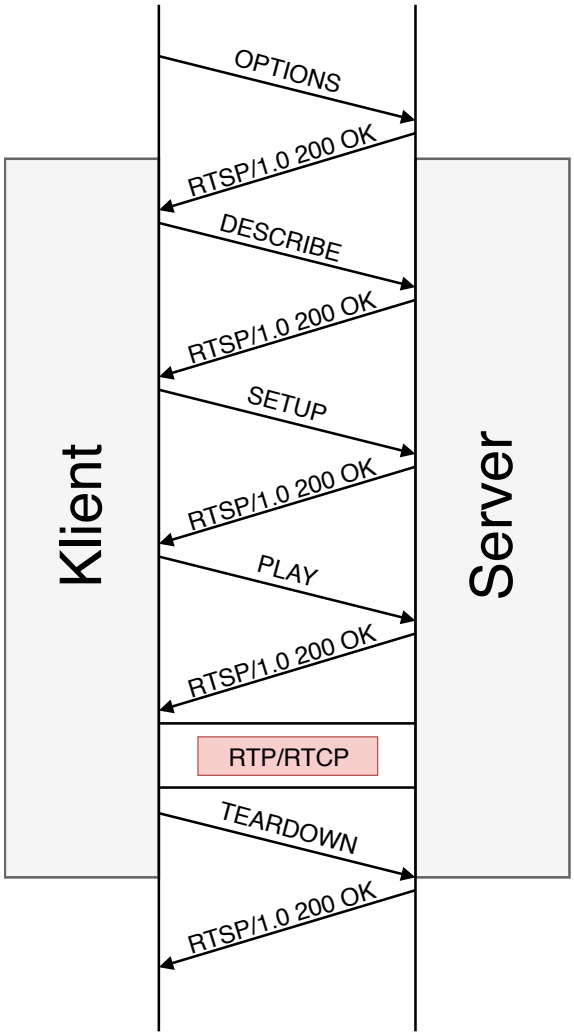
Audio/video vstup může být jak živé vysílání, tak nahraný záznam. Segmenty se nejčastěji ukládájí s koncovkou `.ts` a jsou doporučeny vytvářet pomocí programu Apple stream segmenter. Playlist bývá ve formátu `.m3u8`, který je rozšířením známého playlistu `.m3u` používaného ve zvukovém formátu MP3. Celá architektura je vyobrazena na obrázku níže.



Obr. 4.1: Streamovací architektura HTTP [13]

4.2 Protokol pro přenos dat v reálném čase

Protokol pro přenos dat v reálném čase, RTSP (Real-Time Streaming Protocol vytváří a řídí buď jeden nebo více časově synchronizovaných multimediálních streamů, například zvuku a videa. Jinými slovy, RTSP funguje jako síťové dálkové ovládání multimediálních serverů. [15]



Obr. 4.2: Ukázka RTSP komunikace mezi klientem a serverem. [16]

Protokol RTSP nezná pojem RTSP připojení, místo toho server udržuje relace označené identifikátorem. RTSP relace také není nijak svázána s protokoly transportní vrstvy jako například TCP spojením. Během RTSP relace může klient otevřít a zavřít několik transportních spojení pro zajištění spolehlivého přenosu RTSP zpráv. Alternativně se dá využít i transportní nespojový protokol UDP.

Multimediální streamy kontrolované protokolem RTSP mohou použít RTP, ale operace RTSP nejsou nijak závislé na typu protokolu, který se stará o samotný přesun

multimediálních dat. Protokol je záměrně syntakticky a funkčně podobný protokolu HTTP/1.1, avšak RTSP se liší v několika důležitých aspektech [15] :

- RTSP zavádí řadu nových metod a má jiný identifikátor protokolu.
- RTSP server musí ve většině případů udržovat stav spojení, zatímco HTTP je bezstavový.
- Jak RTSP klient, tak server mohou vytvářet požadavky.
- Data jsou přenášena jiným protokolem.
- URI požadavku vždy obsahuje absolutní identifikátor URI. Kvůli zpětné kompatibilitě, HTTP nese v žádosti pouze absolutní cestu a host name má odděleně v hlavičce.

Protokol podporuje následující operace:

- **Získání médií z mediálního serveru** Klient si může vyžádat popis prezentace pomocí HTTP nebo nějakou jinou metodou. Pokud se jedná o prezentaci přenášenou multicastem, jsou v popisu obsaženy všechny adresy multicastového vysílání a porty, pro plynulý přenos média.
- **Pozvání mediálního serveru do konference** Mediální server může být pozván, aby se připojil k existující konferenci, buď jako další přispívatel nebo pro záznam medií v prezentaci. Tento režim je užitečný pro distribuované výukové aplikace.
- **Přidání média k existující prezentaci** Zvláště pro živé prezentace je užitečné, když server dokáže oznámit klientům o nově dostupných mediích.

Každá prezentace a mediální stream by měl být identifikován pomocí RTSP URL. URL se skládá z těchto částí: „rtsp“, „//“, host, „:“, port (standardně 554), cesta.

Příklad RTSP URI:

```
rtsp://media.example.com:554/twister/audiotrack
```

4.2.1 Metody RTSP protokolu

RTSP protokol podporuje řadu zpráv vysílaných od klienta na server a naopak. První řádek zprávy obsahuje metodu, která se má použít na zdroj, identifikátor zdroje a protokol používané verze.[15] Některé metody jsou povinné, jiné ne, viz tabulka 4.1.

OPTIONS

Metoda OPTIONS vrací všechny další metody, které server přijímá. Může být zaslána v kterýkoliv čas a neovlivňuje stav serveru.

```
K → S: OPTIONS * RTSP/1.0
       CSeq: 1
       Require: implicit-play
```

```
        Proxy-Require: gzipped-messages

S → K: RTSP/1.0 200 OK
        CSeq: 1
        Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

Tab. 4.1: Seznam RTSP metod

Metoda	Směr	Důležitost
DESCRIBE	K→S	doporučená
ANNOUNCE	K→S, S→K	volitelná
GET_PARAMETER	K→S, S→K	volitelná
OPTIONS	K→S, S→K	povinná (S→K: volitelná)
PAUSE	K→S	doporučená
PLAY	K→S	povinná
RECORD	K→S	volitelná
REDIRECT	S→K	volitelná
SETUP	K→S	povinná
SET_PARAMETER	K→S, S→K	volitelná
TEARDOWN	K→S	povinná

DESCRIBE

Metoda DESCRIBE získá od serveru popis prezentace nebo média definovaného v požadavku pomocí URL adresy. Může být použita hodnota z hlavičky Accept, pro upřesnění formátů a zajištění správné funkcionality klienta. Server odpoví popisem požadovaného zdroje. DESCRIBE je součástí inicializační fáze RTSP komunikace.

DESCRIBE odpověď musí obsahovat všechny inicializační informace o zdroji který popisuje. Pokud si klient vyžádá popis prezentace jinou metodou než DESCRIBE a tento popis jíž obsahuje veškeré potřebné informace, klient by měl použít tyto informace a nežádat o ně znovu pomocí DESCRIBE metody. Inicializace spojení je nezbytná u každého RTSP systému, nicméně RTSP nevyžaduje striktně používat metodu DESCRIBE. Existují tři způsoby jak si klient může zjistit inicializační informace:

- pomocí RTSP DESCRIBE metody,
- pomocí dalšího protokolu (HTTP, příloha emailu,...),
- pomocí příkazové řádky.

```
K → S: DESCRIBE rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 312
```

```
      Accept: application/sdp, application/rtsl

S → K: RTSP/1.0 200 OK
      CSeq: 312
      Date: 23 Jan 1997 15:35:06 GMT
      Content-Type: application/sdp
      Content-Length: 376

      v=0
      o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
      s=SDP Seminar
      i=A Seminar on the session description protocol
      u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
      e=mjh@isi.edu (Mark Handley)
      c=IN IP4 224.2.17.12/127
      t=2873397496 2873404696
      a=recvonly
      m=audio 3456 RTP/AVP 0
      m=video 2232 RTP/AVP 31
      m=whiteboard 32416 UDP WB
      a=orient:portrait
```

ANNOUNCE

Metoda ANNOUNCE má dvě využití:

- Při poslání metody z klienta na server obsahuje zpráva popis požadované prezentace nebo média identifikovaného pomocí URL. U směru ze serveru na klienta ANNOUNCE metoda v reálném čase aktualizuje popis relace.
- Při přidání nového mediálního streamu do prezentace (například během živé prezentace), se zasílá celý popis prezentace znovu, místo zaslání jen jeho části.

```
K → S: ANNOUNCE rtsp://example.com/fizzle/foo RTSP/1.0
      CSeq: 312
      Date: 23 Jan 1997 15:35:06 GMT
      Session: 47112344
      Content-Type: application/sdp
      Content-Length: 332

      v=0
      o=mhandley 2890844526 2890845468 IN IP4 126.16.64.4
      s=SDP Seminar
```

```
        i=A Seminar on the session description protocol
        u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
        e=mjh@isi.edu (Mark Handley)
        c=IN IP4 224.2.17.12/127
        t=2873397496 2873404696
        a=recvonly
        m=audio 3456 RTP/AVP 0
        m=video 2232 RTP/AVP 31

S → K: RTSP/1.0 200 OK
        CSeq: 312
```

SETUP

Požadavek SETUP určuje transportní mechanismus, který má být použit pro streamované médium. Klient zašle požadavek SETUP, například na již aktivní stream a požádá server o změnu transportních parametrů. Server mu může, ale i nemusí vyhovět.

Transportní hlavička specifikuje transportní parametry vybrané klientem, odpověď serveru obsahuje parametry zvolené serverem.

```
K → S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
        CSeq: 302
        Transport: RTP/AVP;unicast;client_port=4588-4589

S → K: RTSP/1.0 200 OK
        CSeq: 302
        Date: 23 Jan 1997 15:35:06 GMT
        Session: 47112344
        Transport: RTP/AVP;unicast;
         client_port=4588-4589;server_port=6256-6257
```

PLAY

PLAY metoda říká serveru aby zahájil posílání dat mechanismem specifikovaným v metodě SETUP. Klient nesmí vyslat žádost PLAY dokud nebyla vyhodnocena žádost SETUP jako potvrzená.

Požadavek PLAY nastaví čas přehrávání na začátek určeného rozsahu a začne posílat data dokud nenarazí na konec rozsahu. Server si ukládá požadavky PLAY od klienta do fronty a nedovolí spuštění dalšího přehrávání, dokud není předchozí ukončeno. Pokud PLAY neobsahuje rozsah, začne přehrávat stream od začátku.

```
K → S: PLAY rtsp://audio.example.com/twister.en RTSP/1.0
        CSeq: 833
```

```
        Session: 12345678
        Range: smpte=0:10:20-;time=19970123T153600Z

S → K: RTSP/1.0 200 OK
        CSeq: 833
        Date: 23 Jan 1997 15:35:06 GMT
        Range: smpte=0:10:22-;time=19970123T153600Z
```

PAUSE

PAUSE požadavek způsobuje dočasné zastavení mediálního streamu. Pokud požadavek obsahuje konkrétní stream, pak služby přehrávání a nahrávání jsou pozastaveny. Pokud ovšem server zpracovává prezentaci nebo skupinu streamů, pozastaveny jsou všechny streamy. Po obnovení přehrávání se musí udržovat synchronizace. Během pozastavení si server drží veškeré zdroje. Může je ovšem uvolnit po dobu uvedenou ve zprávě SETUP.

Žádost PAUSE může obsahovat hodnotu určující po jakou dobu bude stream či prezentace pozastavena a ruší i veškeré PLAY požadavky ve frontě.

```
K → S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 834
        Session: 12345678

S → K: RTSP/1.0 200 OK
        CSeq: 834
        Date: 23 Jan 1997 15:35:06 GMT
```

TEARDOWN

TEARDOWN požadavek zastaví doručování dat pro daný URI identifikátor a uvolní všechny zdroje s ním spojené. Pokud nejsou transportní parametry definovány pomocí popisu relace, je nutné před spuštěním znovu poslat žádost SETUP.

```
K → S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 892
        Session: 12345678
S → K: RTSP/1.0 200 OK
        CSeq: 892
```

GET_PARAMETER

Požadavek GET_PARAMETER získá hodnotu zvoleného parametru z prezentace nebo média. Obsah žádosti a odpovědi je libovolný a záleží na implementaci. GET_PARAMETER bez těla může sloužit jako otestování životnosti klienta nebo serveru (ping).

```
S → K: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
       CSeq: 431
       Content-Type: text/parameters
       Session: 12345678
       Content-Length: 15

       packets_received
       jitter
K → S: RTSP/1.0 200 OK
       CSeq: 431
       Content-Length: 46
       Content-Type: text/parameters

       packets_received: 10
       jitter: 0.3838
```

SET_PARAMETER

Tato metoda žádá o změnu parametru prezentace nebo média specifikovaného pomocí URI.

Požadavek musí vždy obsahovat pouze jeden parametr, aby umožnil klientovi zjistit, proč se konkrétní požadavek nezdařil. Pokud požadavek obsahuje víc parametrů, musí server reagovat pouze v případě, že všechny parametry mohou být úspěšně nastaveny. Transportní parametry musí být nastaveny pouze pomocí SETUP metody.

```
K → S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
       CSeq: 421
       Content-length: 20
       Content-type: text/parameters

      barparam: barstuff

S → K: RTSP/1.0 451 Invalid Parameter
       CSeq: 421
       Content-length: 10
       Content-type: text/parameters

       barparam
```

REDIRECT

Žádost o přesměrování informuje klienta, že se musí připojit k jinému serveru. V záhlaví obsahuje povinný parametr Location, který informuje klienta o nové adrese serveru na kterou by měl vysílat požadavky. Může obsahovat i parametr Range sloužící jako informace za jakou dobu k přesměrování dojde. Pokud chce klient dostávat data z nové adresy, musí nejdříve vyslat žádost TEARDOWN pro aktuální relaci a SETUP pro relaci novou.

```
S → K: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0
       CSeq: 732
       Location: rtsp://bigserver.com:8001
       Range: clock=19960213T143205Z-
```

RECORD

Tato metoda inicializuje nahrávání mediálních dat pro rozsah popsaný v popisu prezentace. Časové razítko značí začátek a konec média. Pokud není zadáno, je použit čas z popisu prezentace. Pokud jíž relace začala, nahrávání se spustí okamžitě.

Server si sám určuje jakým způsobem bude ukládat data, buď pod URI požadavku nebo pod nově vytvořeným.

```
K → S: RECORD rtsp://example.com/meeting/audio.en RTSP/1.0
       CSeq: 954
       Session: 12345678
       Conference: 128.16.64.19/32492374
```

5 PRAKTICKÁ ČÁST PRÁCE

V kapitole je nejprve popsán vytvořený HLS a RTSP sampler a jejich implementace do testovacího plánu. Dále je popis nastavení testování na vytvořených serverech.

5.1 Popis použitého softwaru a hardwaru

Tvorba samplerů probíhala v připraveném prostředí od společnosti GiTy. To obsahovalo virtuální obraz operačního systému Centos 7, Eclipse a program pro zátěžové testování JMeter.

Veškeré testování a úpravy probíhaly na počítači umístěném v učebně Fakulty elektrotechniky a komunikačních technologií. Přístup byl zajištěn pomocí vzdálené plochy. Hardwarové parametry jsou uvedeny zde 5.1:

Tab. 5.1: Parametry školního počítače

Operační systém:	Windows 8 Pro 64bitový
Procesor:	Intel(R) Core(TM)i7-3770 CPU @ 3,40GHz (8 CPUs)
Paměť:	32 GB RAM

Na tomto počítači byly vytvořeny dva virtualizované systémy pomocí programu VMware. Jeden pro programování a konfiguraci JMeteru, druhý jako testovací server. Parametry jsou uvedeny v tabulkách 5.2 5.3.

Tab. 5.2: Parametry virtuálního serveru

Operační systém:	CentOS 7 64bitový
Procesor:	Intel(R) Core(TM)i7-3770 CPU @ 3,40GHz (2 CPUs, 1 CPU u RTSP sampleru)
Paměť:	8 GB RAM (1 GB RAM u RTSP sampleru)

Tab. 5.3: Parametry virtuálního JMeter testeru

Operační systém:	CentOS 7 64bitový
Procesor:	Intel(R) Core(TM)i7-3770 CPU @ 3,40GHz
Paměť:	6 GB RAM

5.2 Vytvoření HLS Sampleru

Z popisu v teoretické části práce je patrné, že protokol HLS pro přenos médií v reálném čase využívá známý protokol HTTP. Proto jsem při vytváření modulu vycházel z již existujícího sampleru HTTP, který je součástí základní instalace programu JMeter. Jedná se o nejsložitější sampler ze všech. Obsahuje 119 tříd v 14 balíčcích.

Jako první jsem převzal zdrojové kódy HTTP sampleru a překopíroval do nové zdrojové složky `src/hls`. Následovalo řešení problémů s chybnými importy, jelikož v nově vytvořené složce byly importy uvnitř tříd nastaveny na původní třídy uvnitř HTTP sampleru, muselo se je tedy ručně přepsat. Struktura vzniklých souborů HLS modulu vypadá takto 5.4:

Tab. 5.4: Parametry virtuálního JMeter testeru

Balíček	**Popis**
config	konfigurační třídy sampleru
config.gui	grafické rozhraní konfiguračního sampleru
control	ovladač sampleru
control.gui	grafické rozhraní ovladače sampleru
gui	grafická rozhraní podpůrných prvků
modifier	nadstavbové funkce
modifier.gui	grafické rozhraní nadstavbových funkcí
parser	parsovací nástroj
proxy	konfigurace proxy serveru
proxy.gui	uživatelské rozhraní proxy serveru
sampler	implementace sampleru
util	kodéry a nadstavbové funkce
util.accesslog	filtry
visualizer	panel pro zobrazení parsovaných odpovědí

Aby JMeter nově vytvořený modul rozpoznal, musí se exportovat celá složka jako `.jar` a uložit do adresáře `lib/ext`. Kompilace projektu se spouští souborem `build.xml`. Při každé úpravě kódu se musí celý postup zopakovat.

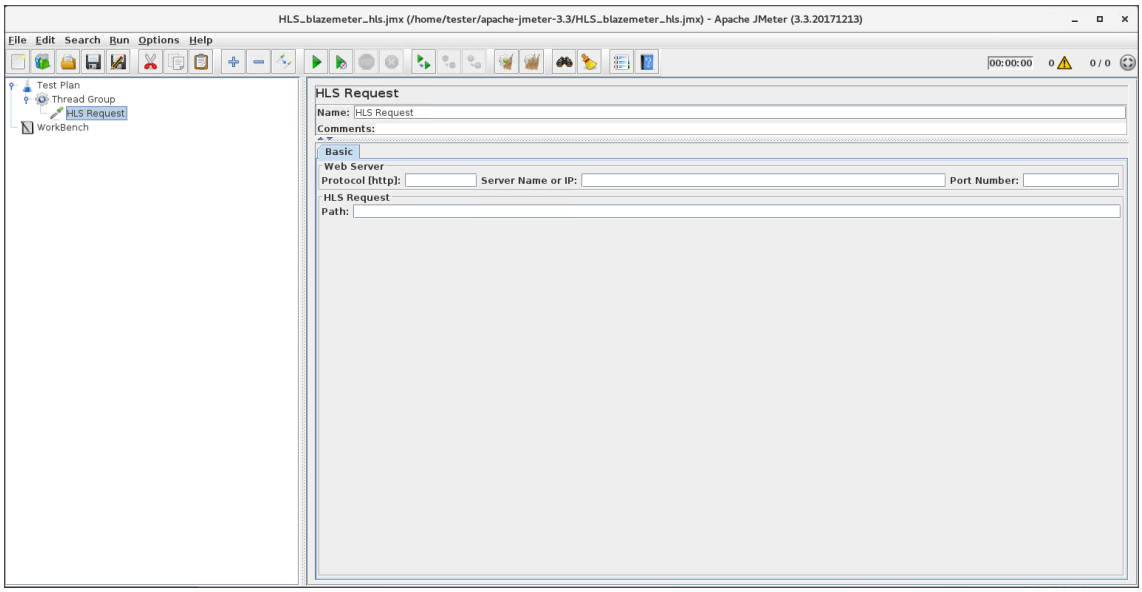
Další krok je úprava grafického rozhraní sampleru. Výchozí HTTP sampler obsahuje všechny potřebné metody k vytvoření HLS sampleru, tudíž jsem upravil pouze uživatelské grafické rozhraní.

Byly odstraněny nepotřebné panely a formuláře. Zásahy probíhaly ve třídě `UrlConfigGui.java` a `HttpTestSampleGui.java`. Seznam změn byl následující:

- odstraněn Advanced panel,
- odstraněna volba dotazovací metody a její výchozí nastavení na GET,

- Content encoding, pro dekódování obsahu,
- odstranění zaškrtávacích políček pro nastavení prohlížeče,
- odstraněny byly také panely Body Data, Files Upload a Parameters.

Pro editaci textových polí JMeter využívá metodu `JMeterUtils.getResString();`. Editace neprobíhá uvnitř třídy, ale v textovém souboru `messages.properties` uloženém v jádru programu, v balíčku `resources`. Výsledný modul je zobrazen na obrázku 5.1. Jeho jednotlivé komponenty budou popsány při implementaci testovacího plánu.



Obr. 5.1: HLS Request modul v programu JMeter.

5.2.1 Návrh a implementace testovacího modelu pro komerční server

Po vytvoření HLS sampleru je další krok jeho otestování. Nejdříve je třeba nalézt veřejně dostupný server, který funguje na protokolu HLS. Byl vybrán server na adrese: `http://playertest.longtailvideo.com/adaptive/wowzaid3/playlist.m3u8`. Pro ověření bylo video zobrazeno v progamu VLC. Ukázka na obrázku 5.2.

Obr. 5.2: Ukázka videa na HLS serveru.

Po ověření funkčnosti videa bylo zapotřebí zjisti náplň Master playlistu. Nejjednodušší způsob je playlist stáhnout a otevřít v textovém editoru. Playlist měl tento obsah:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=538063,
CODECS="avc1.77.21,mp4a.40.2",RESOLUTION=426x240
chunklist_w249832652.m3u8
```

Z Master playlistu můžeme vyčíst údaje jako použitý kodek videa(`H.264`), kodek audia (`MP4A`), rozlišení (`426x240`) a zejména pak navazující playlist obsahujicí segmenty (`chunklist_w249832652.m3u8`). Stejným postupem získáme náplň druhého playlistu.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-TARGETDURATION:16
#EXT-X-MEDIA-SEQUENCE:10713
#EXTINF:9.32,
media_w249832652_10713.ts
#EXTINF:11.76,
media_w249832652_10714.ts
```

```
#EXTINF:9.04,
media_w249832652_10715.ts
```

V tomto playlistu jsou uveden další informace o videu.
Značka `#EXT-X-TARGETDURATION:16` konfiguruje maximální délku jednoho segmentu na 16 vteřin. Podle značek `#EXTINF:` vidíme, že ani jeden segment hodnotu nepřekračuje. Jejich součet nám dá celkovou délku videa, tedy 30,12 vteřiny. Značka `#EXT-X-MEDIA-SEQUENCE:10713` uvádí číslo prvního segmentu, které jsou celkem tři.

5.2.2 Implementace do programu JMeter

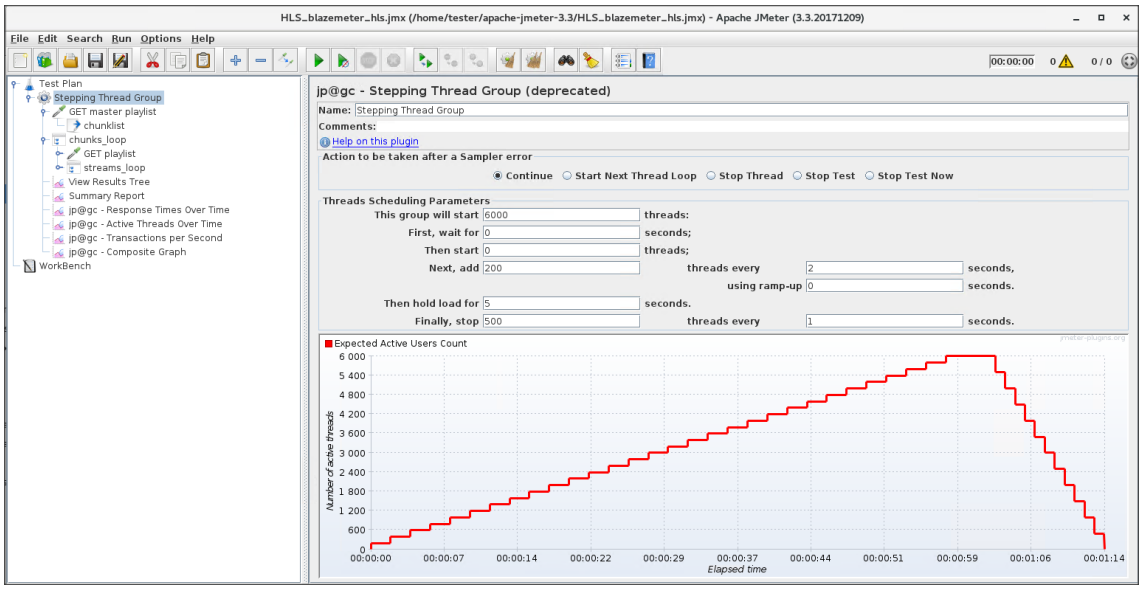
Tato kapitola popisuje použité moduly a jejich konfiguraci, potřebné k úspěšnému testování protokolu HLS pomocí JMeteru. Kompletní schéma je zobrazeno níže.

```
Test Plan
└─ Stepping Thread Group
   ├─ GET master playlist
   │  └─ chunklist
   └─ chunk_loop
      ├─ GET playlist
      │  └─ streams
      └─ streams_loop
         └─ GET Segment
```

Do výchozího testovacího plánu byl přidán modul `Stepping Thread Group`. Jde o modul s otevřeným kódém, který je možné stáhnout z oficiálních stránek JMeteru. Umožňuje schodovité navyšování uživatelů vysílacích požadavky na server. Jeho parametry byly nastaveny tak, aby testovací plán měl charakter testu hraniční zátěže. Cílem tedy bylo zahltit server a donutit přestat odpovídat. Zvolené hodnoty se určily díky několika zátěžovým testům s postupným přidáváním uživatelů. Výsledné nastavení je následovné:

- Celkový počet uživatelů: 6000
- Počáteční stav: 200
- Navyšování: plus 200 uživatelů každé 2 vteřiny s vteřinovým náběhem
- Po dosažení limitu: držet všechny uživatele 10 vteřin
- Ukončení: odhlásit 500 uživatelů každou vteřinu

Grafické prostředí s připraveným testovacím plánem je na obrázku 5.3.



Obr. 5.3: Grafické prostředí modulu Stepping Thread Group.

Modul `GET master playlist` je vytvořený HLS sampler. Jeho úkolem je vyslat na HTTP server dotaz GET a stáhnout Master Playlist. Nastaveny byly tyto parametry.

- Server Name or IP: `playertest.longtailvideo.com/adaptive/wowzaid3`
- Path: `/playlist.m3u8`

Funkce modulu `chunklist` (originální název Regular Expression Extractor) je zaznamenat přijatá data a umožnit ostatním jednotkám s nimi pracovat. Uvnitř Master playlistu se nachází seznam playlistů, které modul najde a uloží je do proměnné `chunklist`. Nastavují se tyto parametry.

- Reference Name: chunklist
- Regular Expression: chunklist_(.+?)\.m3u8
- Template: 1
- Match No. (0 for Random): -1

Další modul `chnuks_loop` (originální název ForEach Controller) funguje jako cyklus, který bere jednotlivě proměnné z pole `chunklist` a ukládá je jako proměnou `chunk`.

- Input variable prefix: chunklist
- Output variable name: chunk

Modul `GET playlist` je opět HLS sampler. Nyní požaduje od serveru dílčí playlisty, jejíž název je uveden v proměnné `chunk`.

- Server Name or IP: `playertest.longtailvideo.com/adaptive/wowzaid3`
- Path: `/chunklist_${chunk}.m3u8`

Stažené playlisty přečte modul `streams` a uloží do proměnné o stejném názvu, obdobně jako u modulu `chunklist`. Jeho konfigurace je taktéž velmi podobná.

- Reference Name: streams
- Regular Expression: media_(.+?)\.m3u8
- Template: 1
- Match No. (0 for Random): -1

Vytvořená proměnná se prochází pomocí `streams_loop`. Názvy segmentů jsou uloženy do nové proměnné `stream`.

- Input variable prefix: streams
- Output variable name: stream

Posledním použitým modulem je HLS sampler `GET segment`, který zasílá požadavky na stažení segmentů s videem.

- Server Name or IP: `playertest.longtailvideo.com/adaptive/wowzaid3`
- Path: `/media_${stream}.m3u8`

Pro zachytávání výsledků je ještě nutné přidat Listnery. Jejich výběr je individuální v závislosti na měřených parametrech. Jako nejlepší považuji `View Result Tree` zobrazující výsledky a obsah zaslaných žádostí od všech samplerů v reálném čase. Dále `Summary Report` pro získání souhrnných údajů o celém testování. Pro grafy bylo doinstalováno několik Listnerů kvůli zobrazení závislosti přihlášených uživatelů a chybovosti. Tímto je konfigurace testovacího plánu kompletní a může se spustit.

5.2.3 Návrh a implementace testovacího modelu pro vlastní server

Kvůli absenci konfigurace komerčního HLS serveru popsaného v předchozí kapitole byl vytvořen vlastní server. Jeho instalace probíhala ve virtuálním prostředí. Vytvoření serverové části bylo pomocí programu VLC a distribuční část byla zajištěna webovým serverm Apache. Instalovány byly tyto verze programů.

- Apache 2.4.6
- VLC 2.2.5

Jejich instalace je popsána v příloze A. Pro potřeby testování bylo stáhnuto nelicencované video o délce 03:03 minuty. Informace o obsaženém kodeku nejsou podstatné, jelikož během vytváření segmentů došlo k překódování na kodek H.264.

Ukázka videa je na obrázku 5.4.

Obr. 5.4: HLS Request modul v programu JMeter.

Vytvoření playlistu a dílčích segmentů umožnil tento příkaz zadaný do terminálového okna. Obsahuje cestu zdrojového souboru a ostatní parametry, které jsou popsány níže.

```
vlc -I dummy /home/qqsrnec/Downloads/sample.mp4 vlc://quit -sout=
'#transcode{width=320,height=240,fps=25,vcodec=h264,vb=256,
venc=x264{aud,profile=baseline,level=30,keyint=30,ref=1},
acodec=mp3,ab=96}:std{access=livehttp{seglen=10,delsegs=false,
numsegs=0,index=/var/www/streaming/mystream.m3u8,
index-url=http://192.168.157.130/streaming/mystream-########.ts},
mux=ts{use-key-frames},dst=/var/www/streaming/mystream-########.ts}'
```

- **seglen** - Udává maximální délku segmentu. Standardní hodnota podle doporučení společnosti Apple je 10 vteřin.
- **numsegs** - Udává počet aktuálně dostupných segmentů uvedených v playlistu. Pro přenos v reálném čase se udává minimálně 3. Pro VoD se konfiguruje 0, což znamená že všechny segmenty jsou uvedeny v playlistu.
- **delsegs** - Maže nepotřebné segmenty. False znamená ponechání všech segmentů.
- **dst** - Cesta do adresáře, ve kterém se budou vytvářet segmenty. Znaky # se změní na pořadová čísla.
- **index** - Cesta do adresáře kam se bude vytvářet playlist.

- **index-url** - URL na kterém se bude video spouštět. Je doporučeno nechat stejné jako **dst**.

Ověření funkčnosti vytvořených souborů můžeme ověřit podobně jako v předchozí kapitole programem VLC. V tomto případě se ale negeneroval žádný Master playlist, jen playlist přímo obsahující segmenty, kterých je vytvořeno celkem 20. Jeho obsah je uveden v příloze B. Uživateli tedy nebyla nabídnuta volba vhodného rozlišení ani možná úprava datového toku v závislosti na šířce pásma.

5.2.4 Implementace do programu JMeter

Postup při implementování testovacího plánu do JMeteru byl obdobný jako v případě komerčního serveru. Vzhledem k nepřítomnosti Master playlistu je schéma kratší o jednu větev. Obdobně obsahuje `Stepping Thread Group` pro schodovité navyšování provozu s těmito parametry.

- Celkový počet uživatelů: 6000
- Počáteční stav: 200
- Navyšování: plus 200 uživatelů každé 2 vteřiny s vteřinovým náběhem
- Po dosažení limitu: držet všechny uživatele 10 vteřin
- Ukončení: odhlásit 500 uživatelů každou vteřinu

Zasílání požadavků zprostředkovávají dva HLS samplery `GET playlist` a `GET segment`. Pro čtení přijatého playlistu slouží modul `streams`, který obdobně ukládá názvy segmentů do proměnné `streams`. Cyklus `streams_loop` zajišťuje předávání proměnné `stream` sampleru. Výsledné schéma vypadá následovně.

```
Test Plan
└─ Stepping Thread Group
   ├─ GET playlist
   │  └─ streams
   └─ streams_loop
      └─ GET Segment
```

Listenery pro zpracovávání dat byly nastaveny stejně jako u testování komerčního serveru. Opět se použily `View Result Tree` a `Summary Report` doplněné listnery zobrazující grafy.

5.3 Vytvoření RTSP Sampleru

Při implementaci a tvorbě HLS sampleru se vycházelo z již existujícího HTTP sampleru, který byl příslušně upraven. U RTSP žádná taková možnost nebyla, nebo se ukázala jako nefunkční (pokus o simulaci RTSP provozu za pomocí TCP sampleru). Bylo tedy nezbytné vyvinout vlastní řešení zcela od začátku.

Prvním krokem bylo vytvořit RTSP komunikaci, následně ji implementovat do JMeteru, vytvořit grafické uživatelské rozhraní a vše otestovat.

Pro potřeby testování bylo důležité vybrat správnou RTSP metodu. Jako nejvhodnější byla zvolena metoda `OPTIONS` z důvodu její jednoduché syntaxe a možnosti ji opakovaně zasílat. Jelikož nemění stav serveru, může na ni reagovat v libovolný čas.

Pomocí programu Wireshark se zjistil přesně jakým způsobem se tvoří požadavek a odpověď v programu VLC, který celou dobu sloužil jako testovací server pro požadavky od klientů. Výsledný požadavek vypadal následovně:

```
OPTIONS rtsp://192.168.157.133:8554/stream RTSP/1.0
CSeq: 1
```

Analyzována byla i odpověď serveru, jenž vypadala následovně:

```
RTSP/1.0 200 OK
Content-length: 0
CSeq: 1
Public: DESCRIBE,SETUP,TEARDOWN,PLAY,PAUSE,GET_PARAMETER
```

5.3.1 Návrh RTSP komunikace

Hlavním stavebním kamenem komunikace a později i samotného sampleru byla třída `Socket`, která je součástí JDK1.0 a vyšších. Tato třída se stará o vytvoření, průběh a ukončení celého TCP spojení. Vytvořením instance této třídy pomocí příslušného konstruktoru se zahájí spojení. Konstruktor vyžadoval dva parametry, IP adresu serveru a příslušný port.

Pro zápis a čtení dat ze soketu byly implementovány dva objekty, `BufferedReader` pro čtení přijatých odpovědí a `BufferedWriter` pro zápis požadavku od klienta. Po jejich vytvoření se naplnily daty pomocí odpovídající metody třídy `Socket` v závislosti na tom, zda-li byly data odchozí, či příchozí. Bylo velmi důležité, aby ani jeden objekt nebyl typu `static`. V opačném případě by byl sampler nefunkční.

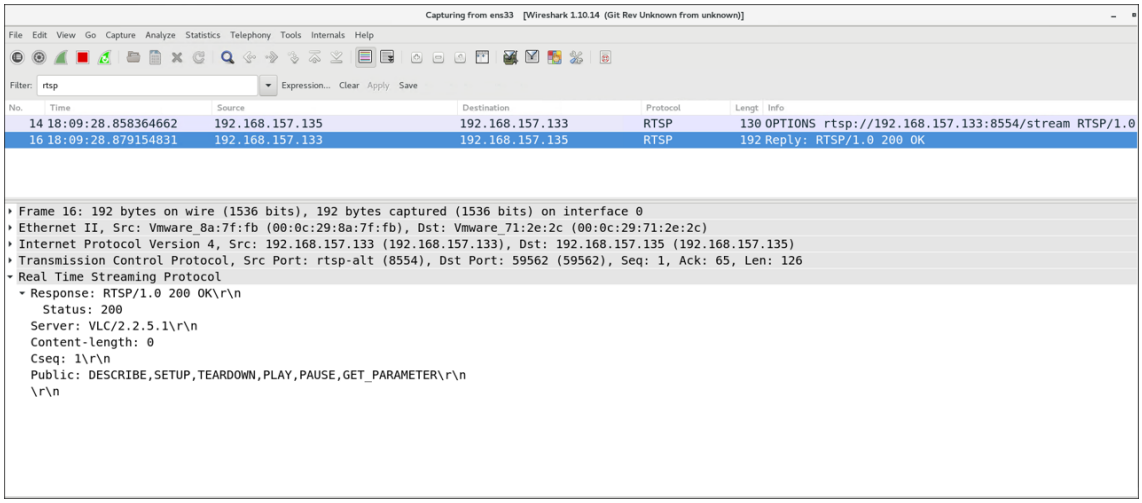
K vytvoření požadavku je určena metoda `send_RTSP_request(String)` s jedním parametrem, udávajícím typ metody. V našem případě šlo pouze o metodu `OPTIONS`. Návrh počítá s možností implementace i dalších metod. Z kódu uvedeného níže lze zjistit jakým způsobem se tvořila žádost `OPTIONS` 5.1.

Výpis 5.1: Kód metody send_RTSP_request(String) vytvářející žádost OPTIONS.

```
private void send_RTSP_request(String request_type) {
  try {
    if (request_type.equals("SETUP")) {
```

```
        RTSPBufferedWriter.write("OPTIONS "
          + getFileName() + " RTSP/1.0\r\n");
        RTSPBufferedWriter.write("CSeq: "
          + RTSPSeqNb + CRLF);
        RTSPBufferedWriter.write(CRLF);
      }
      RTSPBufferedWriter.flush();
    } catch (Exception ex) {
      System.out.println("Exception caught:" + ex);
    }
}
```

Celá komunikace je zachycena na obrázku 5.5 pomocí programu Wireshark, který sloužil pro ověření funkčnosti. První řádek představuje zprávu poslanou z klienta na server s žádostí `OPTIONS`. Druhý řádek je odpověď VLC serveru s výpisem dalších možných žádostí.



Obr. 5.5: Vygenerovaná komunikace zachycená programem Wireshark.

5.3.2 Implementace do programu JMeter a návrh GUI

Zajištění RTSP komunikace bylo první podmínkou úspěšné implementace RTSP sampleru. V druhém kroku bylo nutné zajistit provázání již vytvořeného kódu s JMeterem.

Základem sampleru je třída `TestSampler`, která dědí metody z nadřazené třídy `AbstractSampler`. Povinně implementována musí být metoda `SampleResult`. Jejím úkolem je zprostředkovat komunikaci mezi JMeterem a funkční částí kódu. K tomu

slouží stejnojmenný objekt, do kterého se ukládají všechny vstupy a výstupy, jenž chceme testovat a zároveň zobrazovat v Listenerech. Uvnitř metody se tedy vytvoří soket, zašle požadavek, zpracuje odpověď, vše se uloží do vytvořeného objektu `SampleResult` a ukončí komunikace.

Abychom mohli určit zda-li byl požadavek úspěšný musíme analyzovat přijatou odpověď. Pro tyto účely byla vytvořena metoda `parse_server_response()`, která vrací kód odpovědi serveru. Ten získá z prvního řádku přijatého soketu uloženého v `BufferedReader` 5.2.

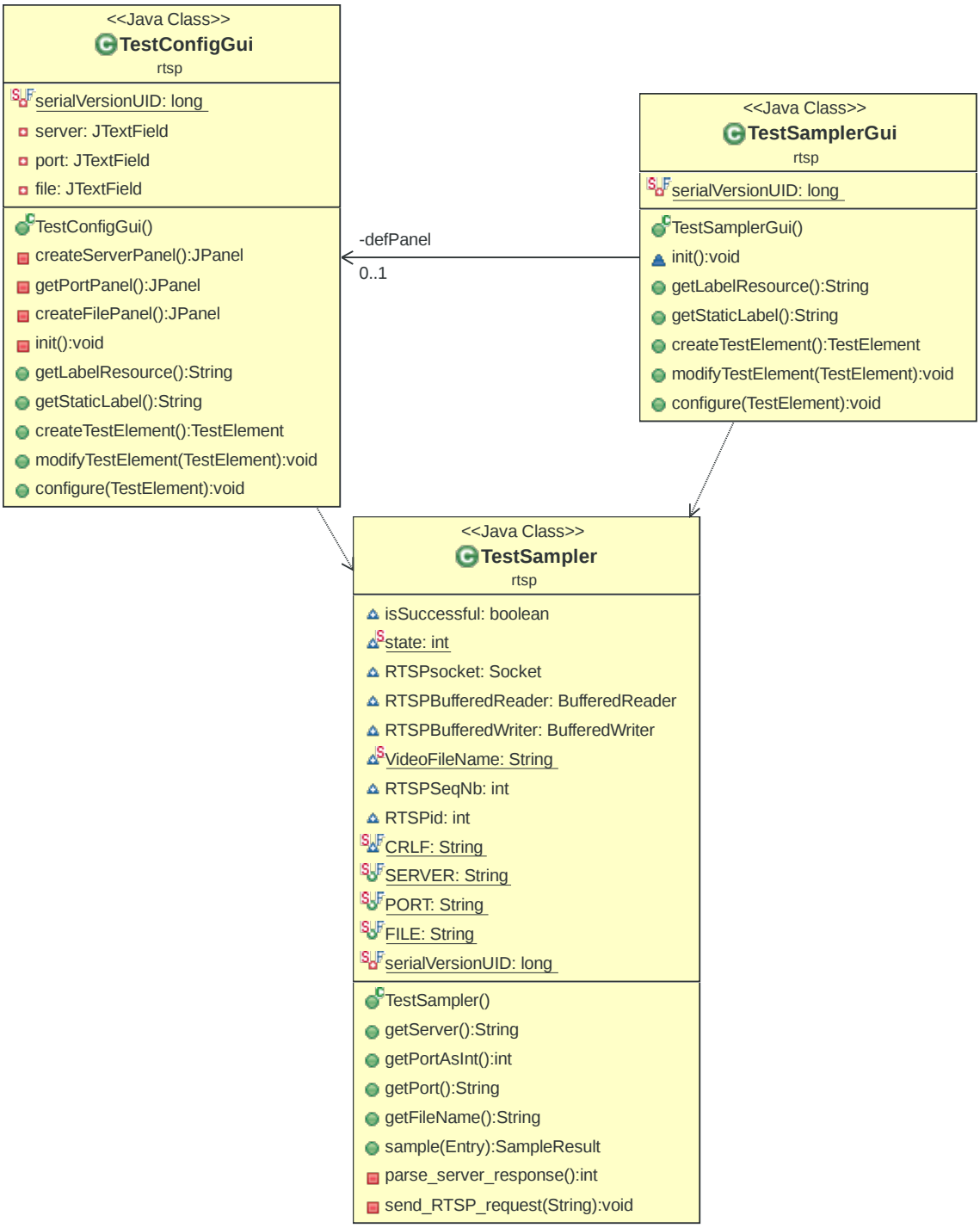
Výpis 5.2: Kód metody parse_server_response().

```
private int parse_server_response() {
    int reply_code = 0;
    try {
        // parse status line and extract the reply_code:
        String StatusLine = RTSPBufferedReader.readLine();
        StringTokenizer tokens
          = new StringTokenizer(StatusLine);
        tokens.nextToken(); // skip over the RTSP version
        reply_code = Integer.parseInt(tokens.nextToken());
    } catch (Exception ex) {
        System.out.println("Exception caught:
          Parse server RES: " + ex);
    }
    return (reply_code);
}
```

Posledním krokem implementace celého sampleru je vytvořit uživatelské grafické rozhraní pro nastavení parametrů požadavku. Za tímto účelem byly vytvořeny dvě třídy, `TestSamplerGui` a `TestConfigGui`. Třídá `TestSamplerGui` se stará o předání parametrů sampleru, zatímco `TestConfigGui` přímo o vzhled GUI. Nejdůležitější metoda `init()` v sobě inicializuje panely, do kterých se umísťují popisy a textová pole. Celkem se načítají tři paramtery a to: jméno (IP adresa) serveru, port a název streamovaného souboru. Víc parametrů k testování nebylo třeba. Celý obsah rtsp balíčku je zobrazený na UML diagramu 5.6. Vzhled sampleru pak na obrázku 5.7.

5.3.3 Testovací plán

Samotný návrh testovacího plánu byl oproti HLS sampleru jednodušší, pravě díky metodám RTSP sampleru, které se staraly o zasílání požadavku a jeho čtení.



Obr. 5.6: UML diagram tříd RTSP sampleru.

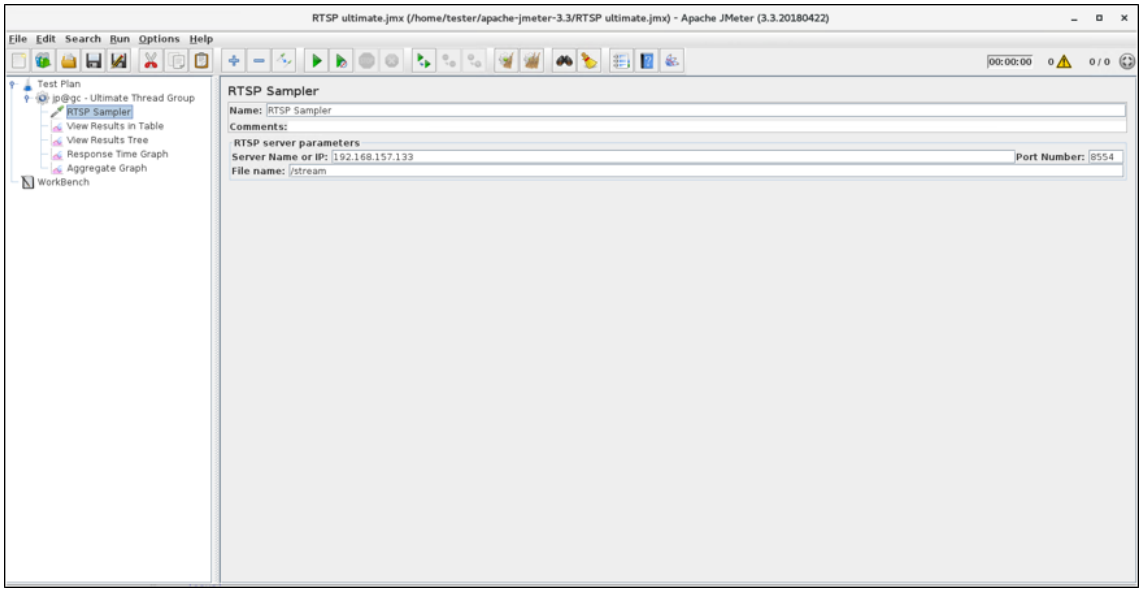
Celé testování tedy obstaral jeden RTSP sampler a nebylo potřeba dalších modulů JMeteru. Schéma testovacího plánu vypadalo následovně.

```
Test Plan
└─ Stepping Thread Group
   └─ RTSP Sampler
```

K vytvoření provozu byl stažen modul `Ultimate Thread Group`, nástupce `Stepping Thread Group` použitého u testování HLS protokolu. Jeho hlavní výhoda je v možnosti vytvořit členitější schodovité načítání uživatelů, čehož bylo patřičně využito. Generován byl tento provoz:

- Celkový počet uživatelů: 2500
- Počáteční stav: 0
- Navyšování: 1000 uživatelů za 5 vteřin
- Navýšování: 1500 uživatelů za 2 vteřiny, start v 10 vteřině
- Po dosažení limitu: držet všechny uživatele 5 vteřin
- Ukončení: odhlásit 1000 uživatelů za 5 vteřin a držet stav 5 vteřin
- Ukončení: odhlásit zbytek za 5 vteřin

Na VLC serveru byl puštěn RTSP stream ve smyčce. Video bylo v rozlišení `640x480`, kodeku `H.264 (MPEG-4)`, 25 snímcích za sekundu a délce 3:03 minuty. Z důvodu menších nároku na server z hlediska RTSP komunikace, byly jeho parametry upraveny. Aby byla ověřena schopnost sampleru kriticky zatížit server, snížil se počet procesorů na jeden a množství RAM paměti z 8GB na 1GB.



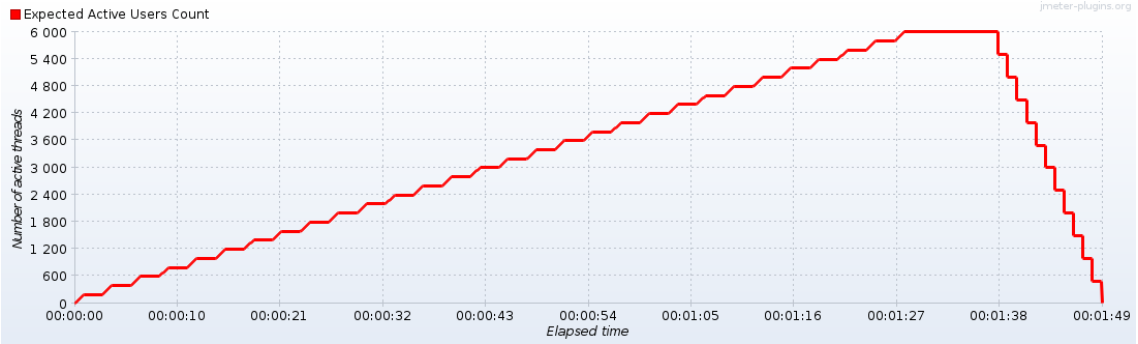
Obr. 5.7: Grafické rozhraní RTSP sampleru.

6 TESTOVÁNÍ VYVINUTÉHO ŘEŠENÍ

Kapitola Testování vyvinutého řešení je rozdělena na dvě sekce. V první jsou popsány výsledky testování HLS sampleru (komerční server a server vlastní), v druhé části jsou výsledky testu RTSP sampleru.

6.1 Testování HLS sampleru

Testování HLS sampleru probíhalo podle nastaveních popsaných v předchozích kapitolách. Graf 6.1 ukazuje nastavené přihlašování uživatelů pro oba servery. Cílem testování bylo se co nejvíce tomuto průběhu přiblížit (modrá série u grafu 6.2 a 6.1).



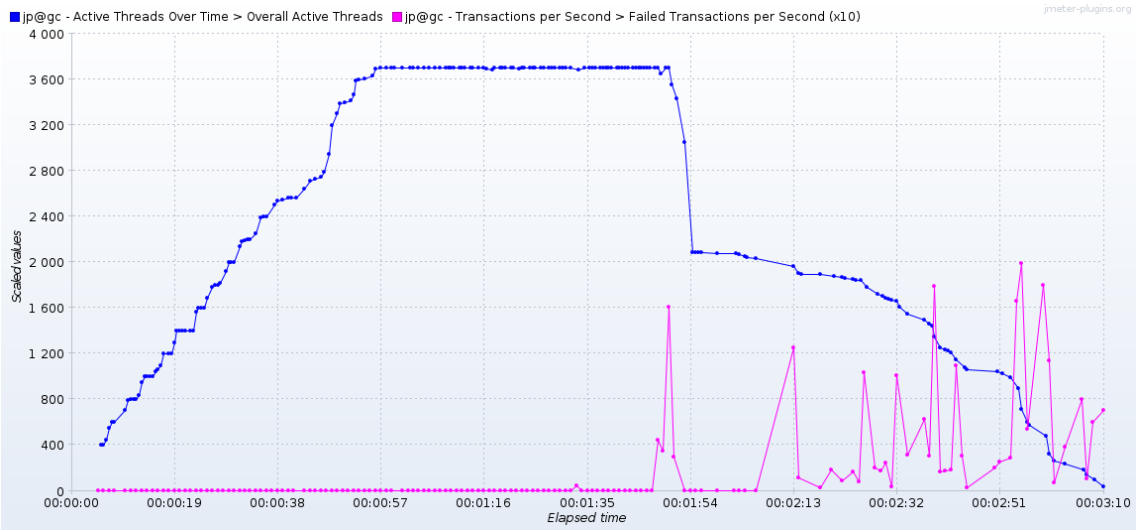
Obr. 6.1: Graf plánovaného přihlašování uživatelů.

6.1.1 Výsledky testu u komerčního serveru

Z průběhu grafu 6.2 můžeme vyčíst několik věcí. Celé testování probíhalo o minutu a 21 vteřin delší dobu než bylo plánováno. Prodloužení bylo způsobeno zejména pomalým odhlašováním uživatelů, které celkově způsobovalo největší generování chyb. JMeter jejich typ popsal jako Connection timed out (Spojení bylo příliš dlouho neaktivní). Postupné přihlašování uživatelů server zvládal bez problémů i když se počet zaráz přihlášených uživatelů zasekl na hodnotě 3 700. Vzhledem k tomu, že při testování na vlastním serveru se počet zasekl na obdobném čísle, jde nespíš o vnitřní nastavení Apache serveru.

Chyby se začaly generovat při odhlašování uživatelů. Server nezvládal velké množství požadavků na ukončení spojení, konkrétně 500 za vteřinu.

Tabulka 6.1 vygenerovaná listenerem `Summary Report` ukazuje dosažené parametry testování pro všechny tři typy požadavků generované HLS samplery. Z výsledků lze určit průměrnou chybovost pohybující se kolem 19%.



Obr. 6.2: Graf závislosti přihlášených uživatelů a chybovosti na čase.

Tab. 6.1: Výsledky testování komerčního serveru.

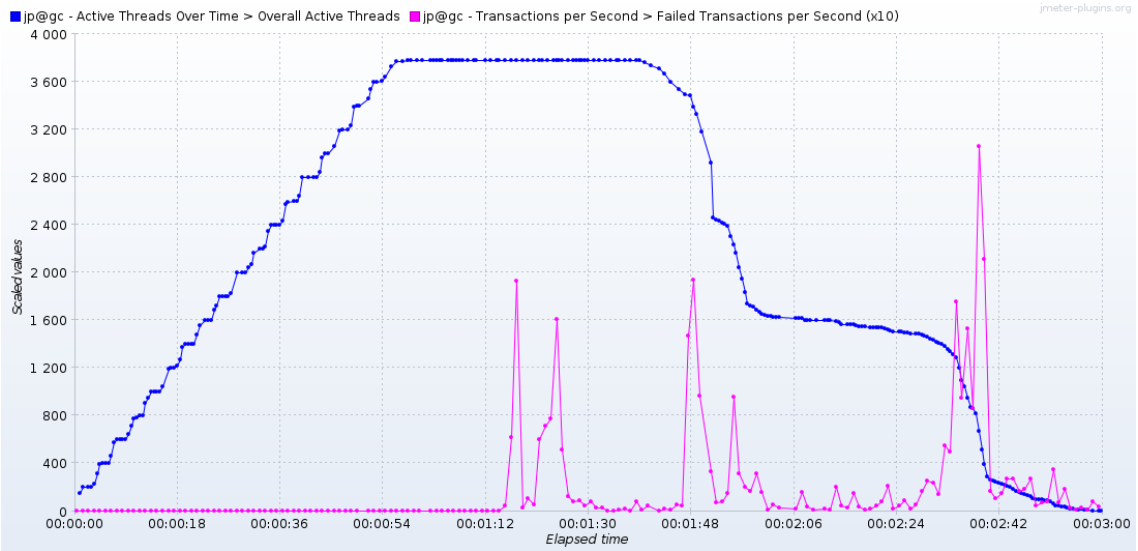
Název požadavku	**GET M pl.**	**GET pl.**	**GET seg.**	**TOTAL**
Počet požadavků	4 396	3 225	4 708	12 329
Chybovost [%]	18,31	20,22	18,03	18,7
Propustnost [pož./s]	23,20	17,54	25,64	65,07
Přijato [KB/s]	22,57	18,45	12 563,58	12 216,27
Posláno [KB/s]	3,02	2,39	3,59	8,82
Prům. množství [KB/s]	0,99	1,08	501,76	192,24

Dále lze vidět, že data potřebná k režii, tedy k posílání playlistů jsou zanedbatelná vůči objemu dat v podobě přenesených segmentů.

6.1.2 Výsledky testu u vlastního serveru

Výsledky testu vlastního serveru jsou částečně podobné s testováním serveru komerčního. I v tomto případě testování probíhalo déle, konkrétně o minutu a 11 vteřin 6.3. Generování chyb v případě plynulého načítání uživatelů nenastalo, ale při velkém množství požadavků na odhlášení. Výjimku tvoří chyby vyskytující se kolem minuty a dvaceti vteřin. Jsou důkazem toho, že server pravděpodobně dosáhl svého limitu.

V tabulce 6.2 jsou uvedeny výsledky testu, ovšem bez požadavku `Get Master playlist`, který zde nebyl použit. Největší rozdíl byl v chybovosti, která byla nevyrovnaná a u požadavku `Get playlist` dosahovala 46,9%, zatímco u požadavku `Get segment` pouze 10,8%. Tyto výkyvy mohou být způsobeny nestálostí virtuálního prostředí a faktem, že jeden fyzický počítač obsluhoval jak server, tak klienta.



Obr. 6.3: Graf závislosti přihlášených uživatelů a chybovosti na čase.

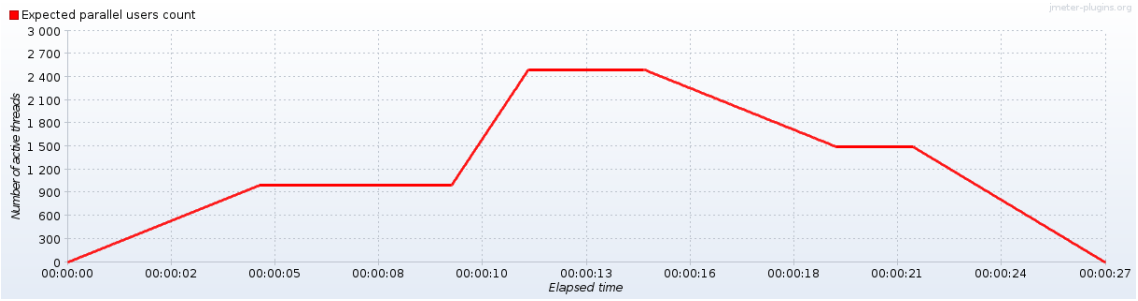
6.2 Testování RTSP sampleru

Testování RTSP sampleru probíhalo za rozdílných podmínek, než u HLS varianty. Bylo to dáno rozdílnou funkčností sampleru, kdy u RTSP nedochází k posílání multimediálních dat, tudíž nároky na server nejsou tak veliké. Z tohoto důvodu se snížily parametry VLC serveru, aby byla zachována charakteristika testu hraniční zátěže.

Na grafu 6.4 je zobrazeno načítání uživatelů v průběhu testu.

Tab. 6.2: Výsledky testování vlastního serveru.

Název požadavku	**GET playlist**	**GET segment**	**TOTAL**
Počet požadavků	4 196	10 716	14 912
Chybovost [%]	46,9	10,8	21
Propustnost [pož./s]	23,48	60,47	83,16
Přijato [KB/s]	49,57	26 382,66	26 125,61
Posláno [KB/s]	1,73	7,85	9,48
Prům. poč. bajtů [KB/s]	2,16	44,68	32,17

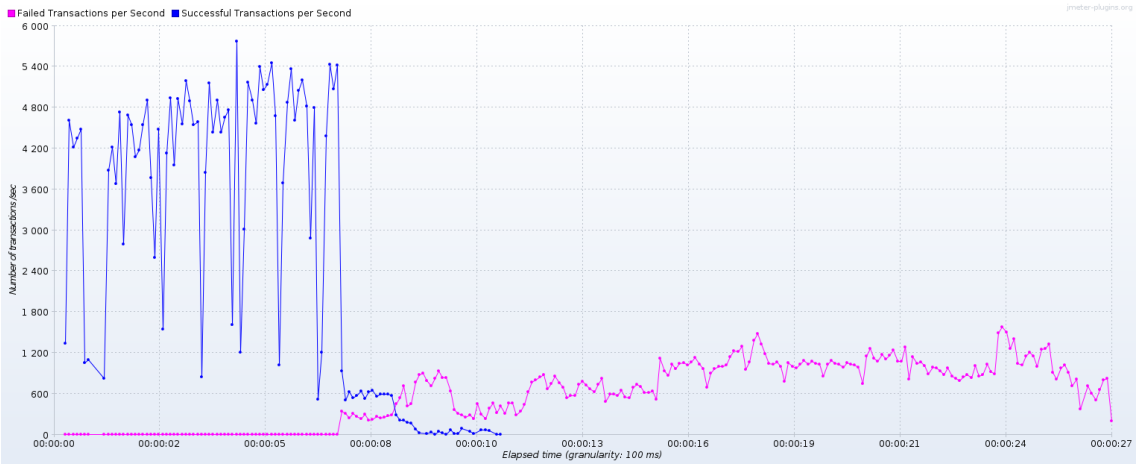


Obr. 6.4: Graf plánovaného přihlašování uživatelů.

6.2.1 Výsledky testu

Všechny výsledky byly zaznamenány pomocí tří listenerů a to: `Transactions per second`, `Summary report` a `Response time graph`. Plánovaná doba celého testu byla 21 vteřin. Z grafu 6.5 můžeme vyčíst, že test byl o 8 vteřin delší. V prvních 5 vteřinách je server zatížen plynulým a mírným provozem. Ani jeden z 6.5 6.6 nevykazuje chybovost a odezva je stabilní. Stejné výsledky jsou u dalších 5 vteřin, po kterých je na serveru přihlášeno stabilně 1000 uživatelů. První chybovost se vyskytuje v momentě prudkého zvýšení zátěže, cca v 10 vteřině testu.

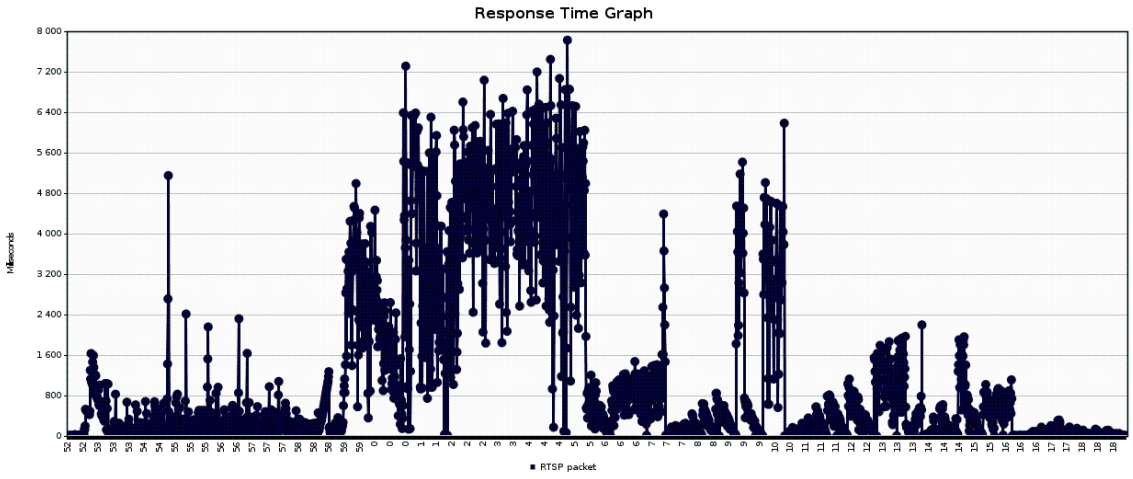
Server začíná generovat chybová hlášení a odezva se několikanásobně zvedá z 1 ms až na 7 ms. V této chvíli množství úspěšných odpovědí prudce klesá, až v 11 vteřině zašle odpověď 200 OK naposledy. Server sice dál odpovídá, nicméně odpovědi jsou vyhodnocovány chybově. K poslednímu výraznému zakolísání odezvy dojde kolem 20 vteřiny po zahájení poslední vlny ukončení spojení zbývajících uživatelů. Tabulky 6.3 ukazuje vysokou chybovost 34,97%, tedy víc jak třetina požadavků nebyla vyhodnocena jako úspěšná.



Obr. 6.5: Graf závislosti přihlášených uživatelů a chybovosti na čase.

Tab. 6.3: Výsledky testování vlastního serveru.

Název požadavku	**RTSP packet**
Počet požadavků	43 415
Chybovost [%]	34,97
Propustnost [pož./s]	1 528



Obr. 6.6: Graf závislosti odezvy na čase.

6.3 Souhrn výsledků

Z výsledků vyplývá, že největší chybovost generuje náhlá změna uživatelů. Jejich postupné přibývání problémy nevytváří. Tento poznatek vychází z testování dvou na sobě nezávislých samplerů, pro dva rozdílné protokoly. Samozřejmě testy byly do určité míry omezeny prostředky virtuálních serverů, ale vždy byla snaha o co největší přiblížení k reálnému provozu, který může v síti nastat.

7 ZÁVĚR

Cílem této práce bylo nastudovat funkce a možnosti rozšíření testovacího programu JMeter se zaměřením na testování aplikačních protokolů HLS a RTSP sloužících pro přenos multimediálního obsahu, zejména pak služby videa na vyžádání.

První kapitola popisuje tři typy zátěžového testování: testování výkonu, zátěžový test a test hraniční zátěže. Jsou zde uvedeny výhody a nevýhody jednotlivých typů a pro přípravu testovacího plánu je vybrán test hraniční zátěže.

Druhá kapitola popisuje program pro zátěžové testování JMeter a vysvětluje prvky, jenž se dají použít pro sestavení testovacího plánu.

V třetí kapitole je popsána služba video na vyžádání, její dva systémy a princip jakým funguje. Následující kapitola shrnuje protokoly použité v práci. Jako první je uveden protokol HTTP, jeho stručná historie, použití a popis dotazovacích metod. Detailní popis protokolu HLS zahrnuje jeho architekturu, která se dělí na část uživatelskou distribuční a serverovou, dále rozbor dvou základních prvků, playlistu a segmentu. Je vysvětlen jejich účel a popsány nejčastější značky. U protokolu RTSP je popsána komunikace mezi severem a klientem, jaké operace protokol podporuje a zejména jsou zde detailně popsány všechny metody, pomoci kterých probíhá komunikace.

Praktická část je rozdělena na dvě části. V každé je podrobně popsán vývoj včetně implementace vytvářeného HLS a RTSP sampleru, které byly použity v testovacích plánech. U HLS jsou popsány testovací plány, první se zaměřuje na komerčně používaný server, druhý na server vlastní. Oba servery při vytvořené zátěži generovaly značný počet chyb. Obdobně i u RTSP je vytvořen testovací plán a otestován na serveru.

Z výsledků vyplývá, že nejhůře server reagoval na rychle se měnící množství přihlášených uživatelů. Jejich plynulý nárůst problémy nezpůsoboval. Tento poznatek byl potvrzen u obou testovaných protokolů.

Všechny moduly do programu JMeter jsou funkční a vyžadují jen správnou konfiguraci, která by s pomocí této práce měla být snadná. Moduly budou také součástí projektu firmy GiTy pro komplexní testování síťové infrastruktury a budou testovány v praxi. Stanovené cíle diplomové práce byly splněny.

LITERATURA

[1] MEIER, J.D., Carlos FARRE, Prashant BANSODE, Scott BARBER a Dennis REA. *Performance Testing Guidance for Web Applications: patterns and practices* [online]. Microsoft Corporation, 2007, s. 221 [cit. 2017-11-18]. Dostupné z URL: <https://msdn.microsoft.com/en-us/library/bb924375.aspx>.

[2] MING JIANG, Zhen a Ahmed E. HASSAN. *A Survey on Load Testing of Large-Scale Software Systems* [online]. 2015 [cit. 2017-11-23]. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7123673>.

[3] BAYAN, Mohamad S. a João W. CANGUSSU. *Automatic Stress and Load Testing for Embedded Systems* [online]. University of Texas at Dallas, 2006 [cit. 2017-11-26]. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4020172>.

[4] G. Gheorghiu. (2005). *Performance vs. load vs. stress testing* [Online]. Dostupné z URL: <http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html>.

[5] BANERJI, Rajat. *Digital Media: Rise of On-demand Content* [online]. Deloitte., 2015, 20 [cit. 2017-12-03]. Dostupné z URL: <https://www2.deloitte.com/content/dam/Deloitte/in/Documents/technology-media-telecommunications/in-tmt-rise-of-on-demand-content.pdf>.

[6] *Global consumer internet traffic*, 2016–2021. In: Cisco.com [online]. Cisco, 2017 [cit. 2017-11-26]. Dostupné z URL: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>.

[7] LEE, Jack Yiu-bun. *Video-on-Demand: Technologies, Systems, and Applications* [online]. In: . The Chinese University of Hong Kong, s. 68 [cit. 2017-12-03]. Dostupné z URL: <http://www.mclab.info/internal/vodtsa.pdf>.

[8] XIAO, Yang. *Internet Protocol Television (IPTV): The Killer Application for the Next-Generation Internet* [online]. In: . University of Alabama, 2007, s. 9 [cit. 2017-12-04]. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4378332>.

[9] *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. USA: Network Working Group, 1999.

[10] *Secure your site with HTTPS: Protect your site and your users.* In: Https://support.google.com [online]. 2017 [cit. 2017-12-05]. Dostupné z URL: <https://support.google.com/webmasters/answer/6073543?hl=en>.

[11] *How does HTTPS actually work?* [online]. Robert Heaton, 2014 [cit. 2017-12-05]. Dostupné z URL: <https://robertheaton.com/2014/03/27/how-does-https-actually-work/>.

[12] HTTP Live Streaming. *https://developer.apple.com* [online]. Apple, 2017 [cit. 2017-12-06]. Dostupné z URL: <https://developer.apple.com/streaming/HLS-WWDC-2017-Preliminary-Spec.pdf>.

[13] HTTP Streaming Architecture. In: *https://developer.apple.com* [online]. 2016 [cit. 2017-12-06]. Dostupné z URL: <https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html#//apple_ref/doc/uid/TP40008332-CH101-SW2>.

[14] HALILI, Emily H. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites.* Birmingham: Packt Publishing, 2008. ISBN 978-1-847192-95-0.

[15] *Real Time Streaming Protocol (RTSP).* RFC 2326. Columbia University: RealNetworks, 1998.

[16] SYME, Matthew a Philip GOLDIE. *Optimizing network performance with content switching: server, firewall, and cache load balancing.* Upper Saddle River, NJ: Prentice Hall, 2004. ISBN 978-0-13-101468-8.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

RAM	Random-access memory
HTTP	Hypertext Transfer Protocol
RAID	Redundant Array of Independent Disks
JDBC	Java Database Connectivity
LDAP	Lightweight Directory Access Protocol
API	Application Programming Interface
CPU	Central Processing Unit
VoD	Video on Demand
IPTV	Internet Protocol television
IP	Internet Protocol
QoS	Quality of Service
TCP	Transmission Control Protocol
HTML	HyperText Markup Language
MIME	Multipurpose Internet Mail Extensions
XML	Extensible Markup Language
SMTP	Simple Mail Transfer Protocol
NNTP	Network News Transfer Protocol
FTP	File Transfer Protocol
URI	Uniform Resource Identifier
HTTPS	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
SSL	Secure Sockets Layer

HLS	HTTP Live Streaming
UTF	Unicode Transformation Format
AAC	Advanced Audio Coding
MPEG	Moving Picture Experts Group
VLC	VideoLAN Client
RTSP	Real Time Streaming Protocol
RTP	Real Time Protocol

SEZNAM PŘÍLOH

A INSTALACE WEBOVÉHO SERVERU APACHE A PROGRAMU VLC

Výpis A.1: Instalace HTTP serveru Apache

```
#aktualizace systému a všech balíčků
sudo yum update

#instalace pomocí nástroje yum
sudo yum install httpd

#automatické spouštění serveru po startu systému
sudo systemctl enable httpd.system

#spuštění serveru
sudo systemctl start httpd

#editace konfiguračního souboru
vi /etc/httpd/conf/httpd.conf

#přidání IP adresy a portu HTTP serveru do proměnné ServerName
ServerName 192.168.157.130:80

#zajištení, že server bude naslouchat na portu 80
Listen 80
```

Výpis A.2: Instalace programu VLC

```
#stažení instalačních souborů
sudo rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/i386/
/epel-release-6-8.noarch.rpm

#instalace VLC
sudo install VLC

#spuštění VLC
vlc -v
```

B PLAYLIST MYSTREAM.M3U8

Výpis B.1: Obsah playlistu mystream.m3u8

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:9.56,
http://192.168.157.130/streaming/mystream-00000001.ts
#EXTINF:8.83,
http://192.168.157.130/streaming/mystream-00000002.ts
#EXTINF:9.38,
http://192.168.157.130/streaming/mystream-00000003.ts
#EXTINF:9.20,
http://192.168.157.130/streaming/mystream-00000004.ts
#EXTINF:9.01,
http://192.168.157.130/streaming/mystream-00000005.ts
#EXTINF:9.58,
http://192.168.157.130/streaming/mystream-00000006.ts
#EXTINF:9.24,
http://192.168.157.130/streaming/mystream-00000007.ts
#EXTINF:9.42,
http://192.168.157.130/streaming/mystream-00000008.ts
#EXTINF:9.58,
http://192.168.157.130/streaming/mystream-00000009.ts
#EXTINF:9.61,
http://192.168.157.130/streaming/mystream-00000010.ts
#EXTINF:9.15,
http://192.168.157.130/streaming/mystream-00000011.ts
#EXTINF:9.24,
http://192.168.157.130/streaming/mystream-00000012.ts
#EXTINF:9.98,
http://192.168.157.130/streaming/mystream-00000013.ts
#EXTINF:9.39,
http://192.168.157.130/streaming/mystream-00000014.ts
#EXTINF:9.41,
http://192.168.157.130/streaming/mystream-00000015.ts
#EXTINF:9.77,
http://192.168.157.130/streaming/mystream-00000016.ts
```

```
#EXTINF:9.26,
http://192.168.157.130/streaming/mystream-00000017.ts
#EXTINF:9.97,
http://192.168.157.130/streaming/mystream-00000018.ts
#EXTINF:8.80,
http://192.168.157.130/streaming/mystream-00000019.ts
#EXTINF:4.78,
http://192.168.157.130/streaming/mystream-00000020.ts
#EXT-X-ENDLIST
```

C PŘÍLOHA

Přílohou elektronické verze této práce je složka Java projektu obsahující zdrojový kód vytvořeného modulu a vytvořené testovací plány programu JMeter.