VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Martin Moučka

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

LABORATORNÍ SCÉNÁŘE UMOŽŇUJÍCÍ SROVNÁNÍ PROTOKOLŮ PŘENOSU WEBOVÝCH STRÁNEK

LABORATORY SCENARIOS FOR COMPARISON OF PROTOCOLS ALLOWING TRANSMISSION OF WEB PAGES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Martin Moučka

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Jan Jeřábek, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Martin Moučka ***ID:*** 154811
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Laboratorní scénáře umožňující srovnání protokolů přenosu webových stránek

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku protokolů souvisejících s přenosem webových stránek, jako jsou zejména HTTP (ve všech verzích), QUICK, SPDY, TCP, UDP a SCTP. Dále nastudujte možnosti různých simulačních a vývojových prostředí vhodných pro tyto účely. Proveďte důkladné srovnání vlastností výše uvedených protokolů a navrhněte dva komplexní scénáře, na kterých bude možné vzájemně porovnat tyto protokoly v různých situacích. Tyto scénáře budou vhodné jako laboratorní úlohy pro předměty zaměřené na komunikační technologie. Výstupem práce budou dva kompletní scénáře včetně podrobných návodů pro studenty, předpřipravených výchozích situací, doplňujících úkolů pro studenty a to vše včetně vzorového řešení. Předpokládá se, že délka realizace jedné úlohy bude pro studenta přibližně 2 hodiny času.

DOPORUČENÁ LITERATURA:

[1] STENBERG, D. Http2 explained, online e-book [online]. [cit. 2016-09-12]. Dostupné z: https://daniel.haxx.se/http2/.

[2] JEŘÁBEK, J. Pokročilé komunikační techniky. Skriptum FEKT Vysoké učení technické v Brně, 2016. s. 1-193.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: doc. Ing. Jan Jeřábek, Ph.D.
Konzultant:

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

ABSTRAKT

Tato diplomová práce se zabývá teoretickým a praktickým srovnáním protokolů pro přenos webových stránek HTTP1.1 (Hypertext Transfer Protocol). HTTPS 1.1, SPDY, HTTP/2 a QUIC na přenosových protokolech UDP (User Datagram Protocol), TCP (Transmission Control Protocol) a SCTP (Stream Control Transmission Protocol). Návrhem a zadáním praktických laboratorních úloh pro ověření teoretických předpokladů a další využití ve výuce. Studenti si na těchto úlohách mohou ověřit chování výše zmíněných protokolů v různých podmínkách. Podmínkami, ve kterých se protokoly porovnávají, jsou ztrátovost paketů, zpoždění na přenosové lince a kolísání zpoždění (jitter).

KLÍČOVÁ SLOVA

HTTP, HTTP/2, HTTP2, SPDY, QUIC, TCP, UDP, SCTP, Laboratorní úloha, Srovnání, Webový protokol

ABSTRACT

This diploma thesis focuses on theoretical and practical comparison of webpage transport protocols such as HTTP/HTTPS (Hypertext Transfer Protocol) v1.1, SPDY, HTTP/2 and QUIC using UDP (User Datagram Protocol), TCP (Transmission Control Protocol) and SCTP (Stream Control Transmission Protocol) as transport protocols. This work also contains design and manual for practical laboratory tasks on which can students verify theoretical assumptions. These tasks compares protocols in different conditions such as packet loss, latency and jitter.

KEYWORDS

HTTP, HTTP/2, HTTP2, SPDY, QUIC, TCP, UDP, SCTP, Laboratory task, Comparison, Web protocol

MOUČKA, Martin *Laboratorní scénáře umožňující srovnání protokolů přenosu webových stránek*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 88 s. Vedoucí práce byl doc. Ing. Jan Jeřábek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „ Laboratorní scénáře umožňující srovnání protokolů přenosu webových stránek“ 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

PODĚKOVÁNÍ

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

Brno ....

podpis autora

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

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Webové stránky jsou v dnešní době složené především z velkého množství obrázků a jiných multimediálních bloků (video, audio), v počátcích Internetu byly naopak složeny především z jednoduché grafiky a textu. Právě webové stránky dnes tvoří většinovou část přenášených dat sítí Internet a k jejich přenášení se stále používají starší protokoly, které však dnešním požadavkům nestačí, a proto vznikají nové.

Cílem práce je porovnat tyto starší protokoly s jejich novějšími verzemi a nebo s protokoly, které se snaží přistoupit k přenosu stránek docela jiným způsobem. Srovnávanými parametry jsou především ztrátovost paketů, obousměrné zpoždění na lince a kolísavé zpoždění (jitter). Výstupem práce jsou především dvě úlohy, které slouží jako modelová situace pro srovnávání protokolů pro přenos webových stránek. Tyto úlohy mohou být dále využity ve výuce a studenti tak na nich mohou sami ověřit teoretické znalosti. Úlohy jsou koncipované tak, aby se student mohl maximálně soustředit pouze na rozdíly mezi protokoly, celé nastavení laboratorního prostředí, včetně výběru měřené úlohy je totiž automatizované.

1 TRANSPORTNÍ PROTOKOLY

Transportní protokoly operují na transportní vrstvě referenčního modelu ISO/OSI (Open Systems Interconnection) [2]. Slouží pro přímé adresování aplikací systému tak, aby věděl, které aplikaci má daný datagram předat.

Dělí se na protokoly, které zaručují pořadí a právnost přenesených dat tzv. spojově orientované protokoly, a které nezaručují správné pořadí ani ucelenost přenesených dat, ale jsou určeny pro kontinuální přenos dat tzv. nespojově orientované protokoly.

K rozpoznání komunikujících aplikací se používají porty, které definují, jak aplikaci na straně příjemce, tak aplikaci na straně odesílatele. Protože pole určující port je v záhlaví dlouhé 16 bitů [3], existuje tedy 65536 portů a jsou děleny do tří skupin:

- **dobře známé (well known)** - porty 0 až 1023 byly vyhrazené pro nejvíce používané služby.
- **registrované (registered)** - v rozsahu 1024 až 49151 jsou přidělovány organizací IANA (Internet Assigned Numbers Authority).
- **dynamické a soukromé (dynamic and private)** - od 49152 do 65535 jsou porty určené pro dynamické přidělování aplikacím (například jako zdrojový port), ale mohou také sloužit jako porty pro cílové aplikace, příliš se nevyužívá.

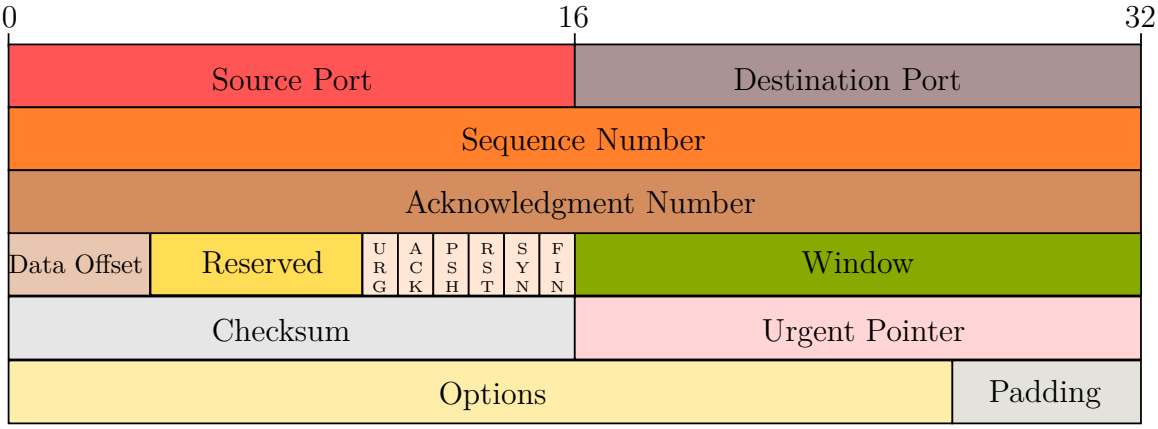
1.1 Transmission Control Protocol (TCP)

TCP (Transmission Control Protocol) byl definován v RFC675 již v roce 1974 a dodnes je udržovaný formou nových rozšíření a úprav standardu tak, aby vyhovoval aktuálním požadavkům počítačových sítí. Dnes zdokumentovaný především v RFC793 [16]. Tento protokol patří mezi spojově orientované protokoly, zajišťuje spolehlivý přenos dat. Jedná se o nejvíce používaný protokol v dnešních počítačových sítích.

Protokol TCP najde využití především v aplikacích, které vyžadujících spolehlivý přenos dat i na úkor zpomalení. Ke zpomalení může dojít například pokud je na trase ztrátovost či chybovost a protokol tak musí zajistit přenesení ztracených nebo poškozených dat znovu. Těmito aplikace mohou být například přenos souborů, emailových zpráv a nebo různé komunikační protokoly na vzdálenou správu.

Záhlaví segmentu se skládá z několika polí definujících přenášená data a aplikace účastnící se relace a doplňujících informace. [16]

- **Source Port** - číslo zdrojového portu.



Obr. 1.1: Struktura záhlaví protokolu TCP

- **Destination Port** - číslo cílového portu.
- **Sequence Number** - sekvenční číslo prvního oktetu dat v segmentu, pokud není vyplněné pole SYN, poté je inicializačním číslem a první datový oktet je na pozici +1.
- **Acknowledgment Number** - označuje následující sekvenční číslo, které odesílatel segmentu očekává pro přijetí.
- **Data Offset** - označuje začátek dat v segmentu.
- **Reserved** - rezervovaná pole pro další použití.
- **URG, ACK, PSH, RST, SYN, FIN** - jednobitové příznaky.
- **Window** - označuje „velikost okna“ (počet oktetů, které je schopný přijmout.
- **Checksum** - kontrolní součet.
- **Urgent Pointer** - označuje konec urgentních (vyslaných mimo pořadí) dat v segmentu.
- **Options** - další volby umožňující další nastavení způsobu přenosu.
- **Padding** - výplň, slouží pro vyplnění hlavičky, tak aby končila na hranici 32 bitů.

Před samotným přenosem dat protokol TCP nejprve navazuje spojení výměnou řídících zpráv s příznaky SYN a ACK to je tzv. **three-way handshake**. Po úspěšném navázání se každý blok přijatých dat potvrzuje zprávou s příznakem ACK a vyplněným sekvenčním číslem následujícího očekávaného bloku dat. Před ukončením spojení se opět vyměňují řídící zprávy, ale s příznaky FIN a ACK, tím si účastníci řeknou, že již nemají další data k poslání a relaci ukončí.

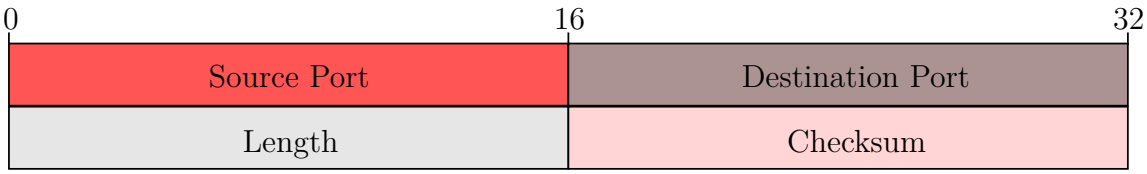
1.2 User Datagram Protocol (UDP)

Protokol UDP (User Datagram Protocol) je nespojově orientovaný protokol poprvé definovaný v RFC798 [12], který v roce 1980 vydal Jon Postel. Oproti TCP protokol UDP nenavazuje spojení, než začne vysílat data a nepotvrzuje úspěšný přenos. Je zde tedy menší režie relace.

UDP se používá především v aplikacích, u kterých není vyžadováno doručení zpráv ve správném pořadí nebo doručení všech odeslaných zpráv. Protože buď aplikace nevyžaduje takovou garanci a nebo si to řeší sama na aplikační úrovni. Díky tomuto mechanismu se používá především při přenosu videa a hlasu, kde uživateli nevadí, že vypadne malá část hovoru, ale je zde zapotřebí, aby data byly přenášena kontinuálně. Taktéž najde využití ve virtuálních privátních sítích, kde ztrátovost řeší právě tunelovací protokol.

Záhlaví datagramu UDP je podstatně menší než u protokolu TCP a to díky chybějícím polím zajišťujících režii a kontrolu relace. [12] Najdeme zde:

- **Source Port** - číslo zdrojového portu.
- **Destination Port** - číslo cílového portu.
- **Length** - délka dat datagramu včetně záhlaví v oktetech.
- **Checksum** - kontrolní součet.



Obr. 1.2: Struktura záhlaví protokolu UDP

1.3 Stream Control Transmission Protocol (SCTP)

SCTP (Stream Control Transmission Protocol) je ve srovnání s UDP nebo TCP velmi mladý protokol, který byl navržen skupinou SIGTRAN (Signaling Transport) standardizační organizace IETF (Internet Engineering Task Force) v roce 2000. Standardizovaný je v dokumentu RFC4960 zveřejněném v roce 2007 [15]. Jedná se o protokol, který se snaží spojit dvě nejlepší vlastnosti z protokolů TCP a UDP. Z TCP je to spolehlivost a správnost pořadí dat a z UDP přebírá přístup odesílaným datům jako k nezávislým zprávám.

Spojení (asociace v terminologii SCTP) je navázání několika paralelních proudů (stream), ve kterých lze přenášet data s garancí spolehlivosti v každém z nich. Přenášená data jsou rozdělena na jednotlivé kusy (chunk) s vlastním označením typu. Pokud se segment poškodí při přenosu, není potřeba posílat všechna data znovu. Potvrzuje se totiž číslo posledního správně přijatého chunk a čísla chybějících, proto se opětovně odesílá pouze část poškozených dat.

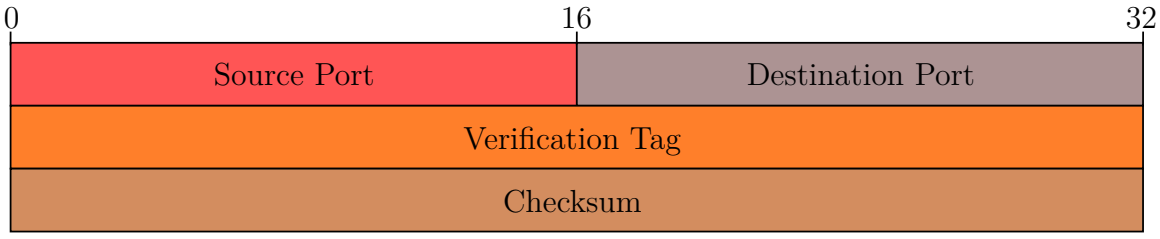
Velkou změnou u tohoto protoklu je tzv. **multi-homing**. Účastnící spojení si na začátku vyměňují všechny své IP (Internet Protocol) adresy, kdy jedna z nich je určena za primární a ostatní za sekundární. Pro opětovné odesílání dat se používá sekundární trasa, pokud je k dispozici. Protokol si také v době spojení udržuje informace o všech cestách a jejich dostupnosti, dojde-li k tomu, že primární cesta začne vykazovat vysokou chybovost a nebo přestane fungovat úplně, začnou se data posílat po sekundární trase trvale a je tedy označena za primární.

Záhlaví protokolu je velice prosté, skládající se pouze ze čtyř polí. [15]

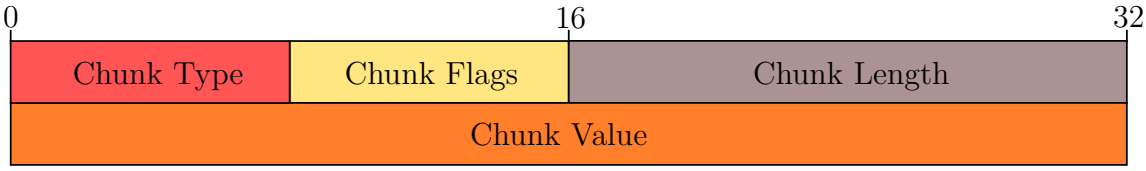
- **Source Port** - číslo zdrojového portu.
- **Destination Port** - číslo cílového portu.
- **Verification Tag** - pole sloužící k identifikaci odesílatele.
- **Checksum** - kontrolní součet.

Za společným záhlavím následují kusy (chunks), které mají také své záhlaví a skládají se z:

- **Chunk Type** - typ kusu. V RFC4960 je zatím definováno 12 typů a zbývající (244) typy jsou buď rezervované a nebo volné.
- **Chunk Flags** - příznak. Jeho využití závisí na konkrétním typu kusu.
- **Chunk Length** - velikost kusu v bajtech.
- **Chunk Value** - hodnota kusu. Závisí na typu například u typu „Payload Data“ jsou to samotná data určená k přenosu.



Obr. 1.3: Záhlaví protokolu SCTP



Obr. 1.4: Formát kusu (chunk) protokolu SCTP

Navázání spojení začíná tím, že klient zašle serveru žádost o navázání spojení (asociace) INIT, ten může obsahovat více IP adres ať už verze 4 nebo 6. Server vygeneruje Cookie jako otisk (hash) z přijatého INIT, chystaného INIT ACK a aktuálního času. Tento hash připojí k INIT ACK zprávě a pošle ji zpět klientovi. Klient poté potvrdí svou žádost další zprávou, ke které připojí Cookie, kterou obdržel od serveru. Tímto je zahájeno spojení a do této doby nebyly rezervované žádné prostředky na straně serveru. Díky tomuto mechanismu SCTP zabraňuje útoku odepření služeb využívaného proti protokolu TCP, kdy útočník zasíláním zpráv SYN vyčerpal prostředky serveru.

1.4 Srovnání TCP, UDP a SCTP

Protokoly TCP a SCTP navazují spojení před přenosem dat, což sice přidává určitou míru režie, ale také to dovoluje řízení toku a obnovu nepřenesených a poškozených dat. SCTP navíc přidává ověření klientské části a tím zvyšuje bezpečnost oproti TCP, taktéž dovoluje zaslání dat spolu s řídícími kusy, tím snižuje negativní vliv režie na přenos. Protokol UDP nenavazuje spojení, ale to je v aplikacích, pro které je určen, naopak výhodou. Velkou nevýhodou protokolu SCTP je, že je velice nový a zatím ne moc rozšířený, přesto je již implementovaný například v operačním systému Linux.

Protokol	Spolehlivý	Zachovává pořadí	Režie
TCP	Ano	Ano	Vysoká
UDP	Ne	Ne	Nízká
SCTP	Ano	Ano	Střední

Tab. 1.1: Srovnání přenosových protokolů

Výhody TCP:

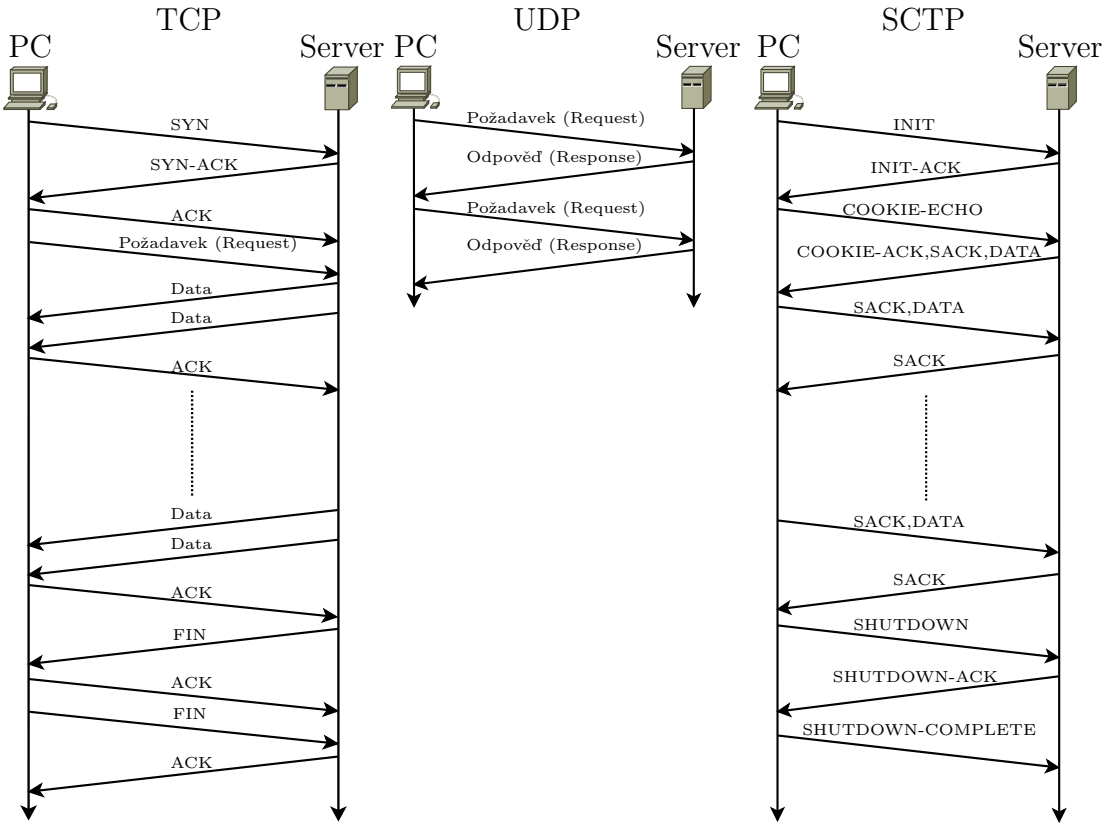
- Spolehlivost a garance správného pořadí dat.

Výhody UDP:

- Nízká režie.
- Kontinuální přenos dat.
- Jednoduchá implementace.

Výhody SCTP:

- Spolehlivost a garance správného pořadí dat.
- Paralelní doručování kusů (chunk) skrze nezávislé proudy dat.
- Multihoming - využití více přenosových tras ke koncové stanici.



Obr. 1.5: Srovnání průběhu spojení nejběžnějších přenosových protokolů

2 PROTOKOLY PŘENOSU WEBOVÝCH STRÁNEK

Protokoly pro přenos webových stránek fungují na sedmé (aplikační) vrstvě ISO/OSI modelu. Pracují na způsobu klient-server, kdy server je schopný poskytovat obsah několika klientům paralelně. S dnešním rozmachem informačních medií a hlavně přesunem veškerého multimediálního obsahu na web je snaha vylepšit tyto přenosové protokoly, aby se celý proces přenosu zefektivnil a zabezpečil.

2.1 Protokol HTTP 1.0

HTTP (Hypertext Transfer Protocol) verze 1.0 se plynule vyvinul z verze 0.9 a v roce 1996 vyšel dokument RFC (Request for Comments) [6] pouze informativního charakteru bez definice standardu. Je určený pro přenos dokumentů ve formátu HTML (HyperText Markup Language).

Protokoly z rodiny HTTP komunikují pomocí zpráv ve formátu dotaz-odpověď. Kdy server naslouchá na určitém portu (obvykle TCP/80), a čeká na dotaz od klienta, po obdržení zpracuje odpověď, a zašle ji zpět klientovi. Každá zpráva typu dotaz obsahuje také metodu. Verze 1.0 definuje tyto tři metody:

- **GET** - slouží jako požadavek na přesný zdroj. Tato metoda by měla být použita pouze na získávání dat.
- **POST** - žádá, aby server přijal odeslaná data. Slouží například pro odesílání formulářů.
- **HEAD** - obdobně jako GET slouží pro získávání informací. Tentokrát však pouze metadat obsažených v záhlaví, nepřenáší žádný jiný obsah.

2.2 Protokol HTTP 1.1

Verze 1.1 je první verzí, která byla vydána jako standard pro usnadnění jednotné implementace. Oficiálně zveřejněna byla v roce 1997 v dokumentu RFC2068. Rok předtím se však během krátké doby dočkala implementace v té době nepoužívanějších webových prohlížečích.

Verze 1.1 funguje, stejně jako verze 1.0, metodou zasílání zpráv obsahujících metody. K metodám obsažených ve verzi 1.0 přidává další metody, kterými jsou [1]

- **PUT** - požadavek na uložení souboru.
- **DELETE** - požadavek na smazání souboru.

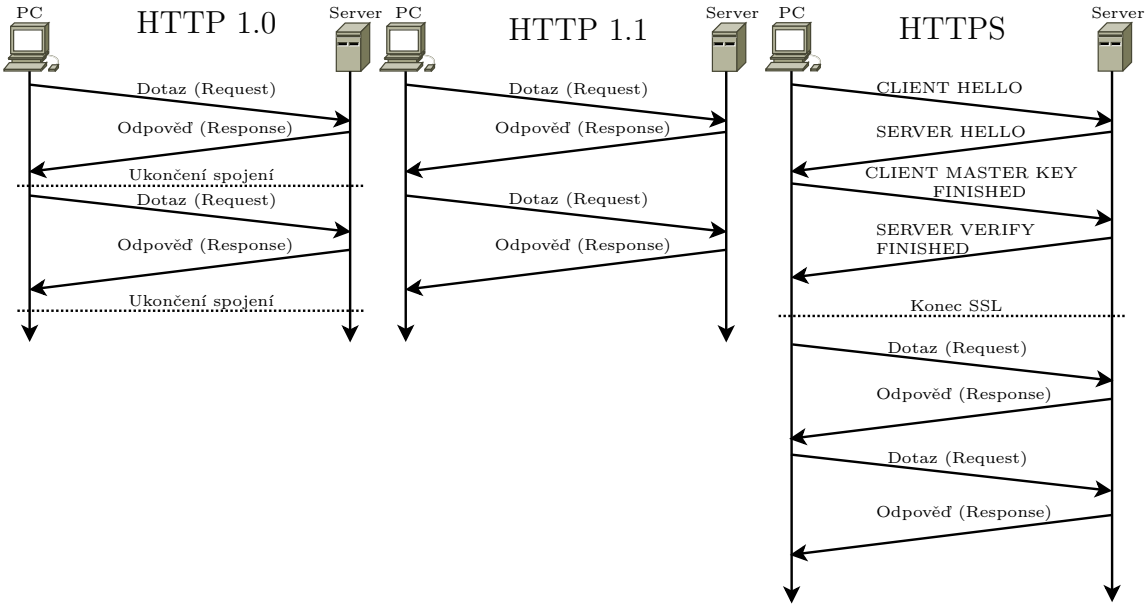
- **TRACE** - zobrazí vyslaní požadavek, takže klient ví, zda byly učiněny nějaké změny.
- **OPTIONS** - dotaz na metody, které server podporuje.
- **CONNECT** - přemění spojení na transparentní TCP/IP tunel.
- **PATCH** - aplikuje částečné úpravy na prostředek.

Protože protokol HTTP zasílá zprávy ve formátu čistého textu, je v síti Internet zranitelný vůči útokům a odcizení přenášených dat. Toto řeší nadstavba HTTPS (HTTP Secure) [13], která přidává šifrovanou SSL/TLS vrstvu. Tato vrstva definuje autentizaci pomocí certifikátu, jak serveru tak i klienta, ale ten většinou zůstává neautentizován a ověřuje se pouze server. Spojení se zapnutým HTTPS se provádí tak, že po připojování klienta k server, zašle server svůj certifikát. Klient vygeneruje náhodný klíč a ten zašifrovaný veřejným klíčem serveru zašle zpátky. Oba provedou matematické operace nad tímto klíčem a získají hlavní klíč, ze kterého vytvoří otisk pomocí hashovací funkce a tímto otiskem pak šifrují veškerý provoz asymetrickou šifrou. Na hashovací a šifrovací funkci se oba účastníci domlouvají na začátku spojení.

2.3 Srovnání HTTP 1.0 a 1.1

Největším rozdílem mezi verzí 1.0 a 1.1, který negativně ovlivňuje výkon verze 1.0, je, že verze 1.1 nenavazuje po každém přenosu nové spojení, ale udržuje stále stejné. Tím snižuje zátěž a zpoždění, protože se nemusí stále provádět TCP handshake. Tento fakt je vidět na obrázku 2.1 spolu se srovnáním navazování spojení HTTPS, kde se mimo zasílání dotazů a odpovědí ještě sjednávají klíče a ověřuje identita serveru, toto se však děje pouze jednou pro každé spojení. Verze 1.1 také nově vyžaduje záhlaví „Host“, která byla ve verzi 1.0 pouze volitelná. Díky tomuto záhlaví je možné využívat proxy servery (prostředník mezi klientem a serverem) a webový server je schopný rozlišit více požadavků na různé virtuální webové servery v jednom spojení. Ukončení spojení i nutnost využití záhlaví Host u verze 1.1 jsou znázorněny na následujících záznamech spojení.

HTTP 1.0 bez záhlaví Host: V tomto výstupu je ukázka spojení protokolem HTTP 1.0 s chybějícím záhlavím Host. Podle odpovědi „200 OK“ vidíme, že spojení bylo úspěšně navázáno. A také vidíme, že „Connection: close“ ukončuje spojení po výměně kombinace dotaz-odpověď.



Obr. 2.1: Srovnání průběhu spojení HTTP 1.0, 1.1 a HTTPS

```
moucka@test_machine:~$ telnet 192.168.121.239 80
GET /index.html HTTP/1.0
Accept-Charset: UTF-8,*

HTTP/1.1 200 OK <-- Spojení úspěšně navázáno
Date: Sun, 27 Nov 2016 10:21:43 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux) PHP/5.4.16
Last-Modified: Sun, 27 Nov 2016 10:20:02 GMT
ETag: "f91-54245b2b99e18"
Accept-Ranges: bytes
Content-Length: 3985
Connection: close <-- Ukončení spojení
Content-Type: text/html; charset=UTF-8
```

HTTP 1.1 bez záhlaví Host: Na následujícím výstupu vidíme, že pokud použijeme protokol HTTP ve verzi 1.1 a nevyplníme záhlaví Host dostaneme negativní odpověď „Bad Request“, což znamená neplatný požadavek. Jelikož spojení nebylo navázáno úspěšně, nechybí zde položka „Connection: close“.

```
moucka@test_machine:~$ telnet 192.168.121.239 80
GET /index.html HTTP/1.1
Accept-Charset: UTF-8,*

HTTP/1.1 400 Bad Request <-- Neplatný požadavek
Date: Sun, 27 Nov 2016 10:23:28 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux) PHP/5.4.16
Content-Length: 226
Connection: close <-- Ukončení spojení
Content-Type: text/html; charset=iso-8859-1
```

HTTP 1.1 se záhlavím Host: Zde vidíme správné navázání spojení při použití verze 1.1 a vyplněné záhlaví Host. Toto dokazuje odpověď „200 OK“, je zde také vidět, že se spojení neukončuje, pokud je úspěšně navázáno. Toto dokládá absence „Connection: close“.

```
moucka@test_machine:~$ telnet 192.168.121.239 80
GET /index.html HTTP/1.1
Host: 192.168.121.239
Accept-Charset: UTF-8,*

HTTP/1.1 200 OK <-- Navázání spojení bylo úspěšné
Date: Sun, 27 Nov 2016 10:24:05 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux) PHP/5.4.16
Last-Modified: Sun, 27 Nov 2016 10:20:02 GMT
ETag: "f91-54245b2b99e18"
Accept-Ranges: bytes
Content-Length: 3985
<-- Chybí položka „Connection: close" -->
Content-Type: text/html; charset=UTF-8
```

2.4 Protokol SPDY

SPDY byl protokol za jehož vývojem stála především společnost Google [8]. Vývoj začal roku 2012, ale neměl dlouhého trvání, protože po vydání verze 2 protokolu HTTP Google pozastavil jeho vývoj ve prospěch nově standardizovaného protokolu.

SPDY nenahrazuje HTTP jako takový, pouze upravuje přenos tak, aby bylo načítání stránek rychlejší. Tento protokol nepřenáší zprávy ve formátu čistého textu,

ale před opuštěním serveru jsou komprimované, a na straně klienta se zase dekomprimují. Tento mechanismus nejen zrychluje přenos, ale také snižuje využití dat, což byl další cíl nového protokolu, jelikož mobilní zařízení jsou dnes většinovou skupinou přistupující k webovým stránkám. Další změnou je, že server může vynutit přenos některých částí webové stránky dříve, než obdrží požadavek na jejich zaslání. Zabezpečení opět zajišťuje SSL/TLS, u tohoto protokolu již není volitelné, ale vynucené.

2.5 Protokol HTTP2

HTTP verze 2 je revizí starších protokolů rodiny HTTP (1.0 a 1.1) sémanticky zůstává stejný, liší se především ve způsobu přenášení dat počítačovou sítí. Standard byl předložen ke schválení roku 2014 a po schválení vydán v dokumentu RFC7540 v květnu roku 2015. Jeho podporu zařadila většina používaných prohlížečů do konce téhož roku. Dle webu w3techs.com [17] tohoto protokolu využívalo 10,8% webových stránek již v prosinci roku 2016.

Na rozdíl od protokolu SPDY je šifrování dat pouze doporučené, a ne vynucené. Přesto některé prohlížeče jako například Chrome a Firefox implementují HTTP2 pouze ve variantě se zapnutým šifrováním [14]. K zabezpečení přenosu se opět používá TLS.

2.5.1 Formát zpráv a způsob přenosu

Od svých předchůdců se liší především formátem zasílaných zpráv. HTTP ve verzích 1.0 a 1.1 zasílali zprávy jako otevřený text snadno čitelný pro člověka, verze 2 naopak zasílá data v binární komprimované podobě. Díky tomu se zmenší objem dat, který je potřeba přenést a s tím i vytížení linky a doba potřebná pro přenos. Což je nejhlavnějším cílem tohoto protokolu.

Zprávy jsou ve verzi 2 zasílány v jednom TCP spojení, jako tomu bylo u verze 1.1, ale jsou zasílána v paralelních proudech dat (streams), což je způsob se kterým se na transportní vrstvě můžeme setkat například u protokolu SCTP. Každý z těchto proudů je nezávislý, přesto lze závislost vynutit nastavením parametru spojení. V takovém případě jsou proudu přiděleny zdroje, jen pokud je-li proud na němž je závislý ukončen a nebo nemůže pokračovat. Každý z proudů umožňuje řízení toku, čímž tento protokol do jisté míry duplikuje vlastnosti protokolu SCTP. Jednotlivé pakety jsou označeny identifikátory, aby bylo poznat, do jakého proudy patří. Liché rozlišují proudy založené klientem a sudé serverem. Pokud má paket nulový identifikátor, znamená to, že má dopad na celé TCP spojení.

Zprávy se nově rozdělují do „rámců“. Každá zpráva je typicky složena z jednoho záhlaví (rámec HEADERS) a z několika rámců dat (DATA). V protokolu je definováno více typů rámců [7]

- **HEADERS** - záhlaví dotazu (request) klienta nebo odpovědi (response) serveru.
- **DATA** - vlastní přenášená data.
- **CONTINUATION** - další záhlaví, pokud přesáhnou velikost jednoho rámce.
- **SETTINGS** - nastavuje vlastnosti spojení.
- **PUSH_PROMISE** - oznamuje data, která budou odeslána bez jejich vyžádání.
- **RST_STREAM** - ukončení proudu.
- **PRIORITY** - změna váhy proudu.
- **WINDOW_UPDATE** - změna okna, slouží pro řízení toku dat.
- **GOAWAY** - zakazuje protější straně založit další proudy dat.
- **PING** - test spojení.

2.5.2 Vynucený přenos dat

V protokolu je také implementovaná funkce vynuceného přenosu dat serverem, která, obdobně jako u SPDY, slouží k přenosu zdrojů na dotazované stránce předtím, než o ně klient požádá. Jedná se především o obrázky nebo kaskádové styly CSS (Cascading Style Sheets). Tím se dosahuje rychlejšího načítání stránek, protože server nečeká, než o zdroje klient zažádá, a také jsou přenášena v paralelním proudu dat. Tento proud server navazuje rámcem „PUSH_PROMISE“ a klient reaguje pouze, pokud nemá zájem o data bez vyžádání a to rámcem „RST_STREAM“, kterým ukončí proud. Z důvodu, že by takto jednající server mohl klienta jednoduše přetížit, byla implementována také ochrana proti zahlcení. Klient může využít možnosti řídit celé TCP spojení nebo omezit velikost přenosového okna pouze pro určité proudy. [14]

2.5.3 Záhlaví

Jelikož každá přenesená zpráva obsahuje záhlaví, které může být v některých případech větší než přenášená data. Byl by takový přenos neekonomický, a tak je nově uvedena komprese záhlaví, která na rozdíl od protokolu SPDY, který využívá zranitelný kompresní mechanismus, využívá zcela nový kompresní mechanismus. Každý z účastníků si vede svou tabulku záhlaví pro celé TCP spojení. Každý odesílatel má pak na výběr ze tří možností, v jakém formátu bude záhlaví přenášeno. Pokud se

rozhodne poslat celé záhlaví, může nastavit příznak, zda si ho má příjemce uložit do své tabulky.

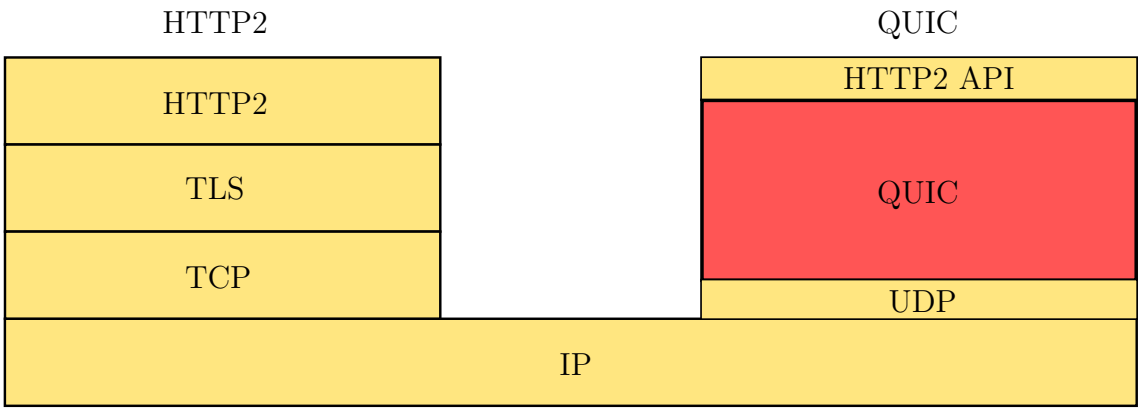
Možné způsoby přenosu záhlaví:

- **Plná komprese** - je přenášen pouze odkaz do tabulky záhlaví.
- **Částečná komprese** - přenáší se odkaz na jméno záhlaví a vlastní obsah.
- **Bez komprese** - přenáší se celé záhlaví.

2.6 Protokol QUIC

QUIC (Quick UDP Internet Connections) je protokol vyvíjený firmou Google, poprvé implementovaný byl v roce 2012 jako experimentální v prohlížeči Chromium (experimentální a otevřená varianta prohlížeče Chrome). V červnu roku 2015 byl podán návrh na schválení standardu ke standardizační organizaci IETF a pracovní skupina pro tento protokol byla sestavena v roce 2016. Jedná se tedy o velice mladý protokol, který přistupuje k přenosu stránek zcela novým způsobem. Nestaví totiž na transportním protokolu TCP jako ostatní, ale využívá UDP. Hlavním cílem tohoto protokolu je snížení odezvy a vytížení přenosových kanálu, stejně jako je tomu u SPDY a HTTP2.

Z pohledu referenčního modelu ISO/OSI se protokol QUIC špatně zařazuje, protože funkčně zasahuje do několika vrstev. Sice využívá protokol transportní vrstvy UDP, ale také implementuje některé vlastnosti této třídě vlastní. Jako je ochrana proti zahlcení, číslování rámců sekvenčními čísly a zajištění doručení všech dat. Z relační a prezentační vrstvy zahrnuje vlastní šifrování a kompresi dat. Do aplikační vrstvy zasahuje jen částečně, protože z části pro interpretaci dat, využívá protokol HTTP2. Jeho zařazení do ISO/OSI modelu je znázorněno na obrázku 2.2.



Obr. 2.2: Srovnání zařazení protokolu QUIC v modelu ISO/OSI oproti HTTP2

2.6.1 Spojení u protokolu QUIC

Navázání spojení u protokolu QUIC vyžaduje mnohem menší počet zaslaných zpráv, než u jiných protokolů určených pro přenos webových stránek. Přistupuje k tomuto kroku zcela novým způsobem. Místo zpráv pro sestavení TCP spojení, výměně certifikátů a sjednání šifrování, navázání spojení potřebuje pouze jednu až dvě zprávy. S ohledem na to, zda klient komunikuje s novým serverem a nebo již známým. Pokud sestavuje spojení s novým serverem, zašle nešifrovanou zprávu „zahájení spojení“ (CHLO), a server odpovídá zamítající zprávou (REJ), ve které sděluje klientovi, že musí navázat šifrované spojení. Zpráva také obsahuje žeton (token) zdrojové adresy a certifikát serveru. Při každém dalším navazování spojení s tímto serverem může klient použít uložená data a poslat zahajovací zprávu zašifrovanou, tím je zahájeno spojení a dochází okamžitě k výměně dat.

Jedno spojení QUIC protokolu se skládá z několika paralelních UDP proudů. Díky tomu, že UDP nezaručuje pořadí paketů, ztracený paket blokuje pouze ten proud, do kterého patří. Na rozdíl od HTTP2, kde je sice více paralelních proudů, ale v jednom TCP spojení, a když se ztratí paket patřící k TCP spojení, dojde k zablokování celého spojení, dokud se požadovaný paket nepřenese znovu.

2.6.2 Dopředná oprava chyb

Dopředná oprava chyb [9] neboli Forward Error Correction (FEC) je mechanismus, který zvyšuje přenášenou zátěž, ale dovoluje znovu sestavit ztracený paket na straně příjemce. Každý odeslaný paket obsahuje část dat obsažených v paketu jiném. To umožňuje příjemci tyto data vzít a ztracený paket znovu sestavit bez nutnosti jeho opětovného přenášení. V současné době je protokol nastaven tak, že z deseti UDP paketů jsme schopni sestavit jeden ztracený, což znamená o 10% větší zátěž, ale opětovné zaslání paketu trvá minimálně jeden cyklus, protože příjemce musí potvrdit chybějící paket a odesílatel ho znovu odeslat. Tento mechanismus má tedy za cíl snížit výslednou odezvu na úkor více přenášených dat.

2.6.3 Mobilita

Další novou funkcí toho protokolu je označování proudu unikátním číslem, který generuje klient na začátku spojení. Tímto se zvyšuje mobilita oproti spojení sestaveného protokolem TCP, kde k identifikaci spojení slouží čtyři parametry (zdrojová, cílová adresa, zdrojový a cílový port). Pokud se klientovi změnila adresa nebo směrovač vykonávající překlad adres smazal tabulku a spojení přiřadil jiný port, dojde k rozpadu spojení a musí být sestavené znovu. Díky vyšší mobilitě spojené s unikátním

identifikátorem se může klient pohybovat například mezi připojením přes mobilní a bezdrátovou síť. Nebo stahovat na dvou připojeních současně.

2.7 Srovnání HTTP2 a QUIC

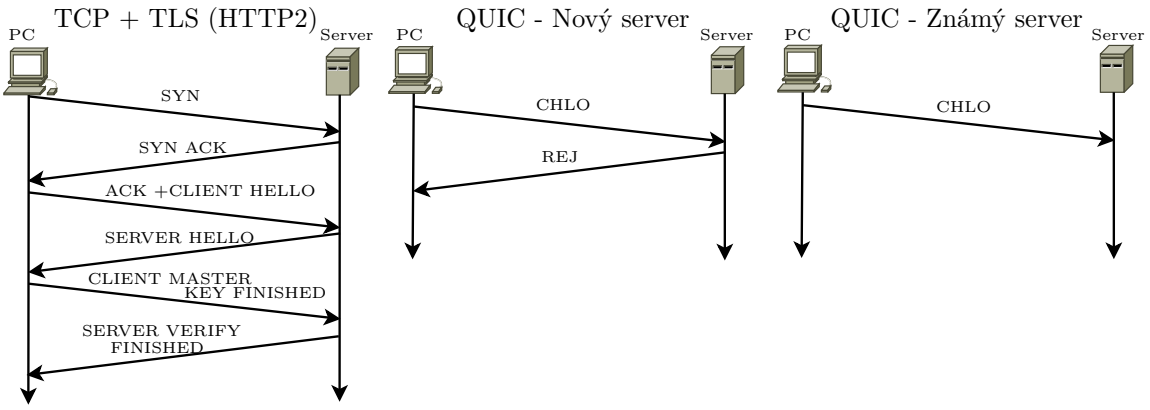
Velkou výhodou protokolu QUIC oproti HTTP2 je, že navázaní spojení trvá zlomek času, díky integraci šifrování a z části i transportní vrstvy. Zároveň se také jedná o nevýhodu, protože tím narušuje zaběhnutý systém a možnost variability. Další výhodou je multiplexing a mobilita koncového zařízení v průběhu trvajícího spojení.

Výhody protokolu QUIC:

- Multiplexing
- Mobilita
- Rychlost navázání spojení
- Vynucené šifrování

Výhody protokolu HTTP2:

- Variabilita protokolů nižší vrstvy
- Schválený standard
- Navazuje na již známé protokoly
- Vynucený přenos dat



Obr. 2.3: Srovnání sestavení spojení HTTP2 a QUIC

3 POUŽITÉ NÁSTROJE

Protože není k dispozici jeden webový server, který poskytuje podporu všech verzí HTTP a QUIC spolu s možností využití přenosového protokolu SCTP. Musíme pro testování použít dva různé servery Apache a Caddy. Tím není možnost přímého srovnání HTTP 1.1 nebo 1.0 přes SCTP a HTTP2 a QUIC. Jelikož výstupem práce mají být i laboratorní úlohy pro studenty, na kterých si budou moci prakticky ověřit vlastnosti jednotlivých protokolů, s výhodou využiji i nástroj pro usnadnění správy operačního systému Ansible, který nastaví laboratorní úlohu přesně dle zadání měření. Pro simulaci podmínek reálné sítě poslouží nástroj NetEm.

3.1 Webový server Apache

Jedná se o jeden z nejrozšířenějších webových serverů, jehož první verze s označením 0.6.2 byla vydána v roce 1995, aktuální verze je 2.4. Zdrojový kód je otevřený a server je určený pro operační systémy UNIX (včetně Linux) a Windows. Skládá se z jádra, které obsluhuje základní funkce a je možné ho rozšířit binárními moduly[1]. Jeho konfiguraci je možné rozdělit na virtuální servery a tím provozovat na jednom serveru více webových stránek.

3.2 Webový server Caddy

Caddy je webový server, který si klade za cíl zjednodušit poskytování webových stránek. Je primárně určen vývojářům, designérům a široké veřejnosti bez potřebných technických znalostí systémového administrátora. Je také zatím jediným serverem, který obsahuje experimentální podporu protokolu QUIC. Jeho kód je opět otevřený a byl poprvé uveřejněný na konci roku 2015[10] na portálu GitHub.

3.3 Simulátor NetEm

Emulační nástroj, s otevřeným zdrojovým kódem napsaný pro operační systém Linux, určený pro simulaci různých vlivů reálné sítě na přenos dat. Ovlivňuje pakety odcházející z vybraného síťového rozhraní na úrovni jádra. Jeho limitací může být zrnitost času, protože Linux nevykonává příkazy v reálném čase, ale až tehdy, když se jim dostane systémových prostředků.

Vlastnosti ovlivnitelné nástrojem NetEm[11]:

- **Delay** (zpoždění) - přidává nastavený čas k reálné době potřebné k doručení paketu.
- **Loss** (ztrátovost) - umožňuje nastavit procentuální hodnotu paketů, které budou zahozeny.
- **Corrupt** (narušení) - způsobuje chybovost na náhodném místě paketu pro specifikované procento paketů.
- **Duplicate** (duplikace) - nastavuje procento paketů, které budou duplikovány.
- **Reorder** (Změna pořadí) - určuje kolik procent paketů bude odesláno ve špatném pořadí.

3.4 Ansible

Je nástroj určený pro centralizovanou správu operačních systémů a služeb na nich bežících. Ke svému běhu potřebuje na koncové stanici pouze Python a funkční spojení přes SSH (Secure Shell). Nástroj má taktéž otevřený zdrojový kód a za jeho vývojem stojí jak komunita, tak firma Red Hat. V laboratorních úlohách má za cíl usnadnit studentům nastavení serveru a klienta pro jednotlivé úlohy, aby nedošlo ke zkreslení výsledků měření špatným nastavením prostředí. Mohou se tak více soustředit na samotné měření. Slouží také jako ukázka přístupu ke správě mnoha systému v dnešním firemním prostředí.

4 NÁVRH LABORATORNÍCH ÚLOH

Laboratorní měření se bude skládat ze dvou úloh, kdy každá prověří jednu skupinu protokolů přenosu webových stránek, podle podpory zvoleného serveru. Jelikož některé přenosové a aplikační protokoly mají pouze experimentální podporu, byly zvoleny dva servery, tak abychom mohli otestovat co nejširší spektrum protokolů při různých podmínkách. Sledovanými parametry budou například rychlost načtení stránky, doba potřebná k přenesení souboru a výsledný objem přenesených dat.

Obě měřené úlohy mají stejný postup a parametry, liší se tedy pouze srovnávanými protokoly. Toto nám umožňuje nejen porovnat mezi sebou protokoly, ale také srovnat vliv webového serveru na parametry přenosu.

4.1 Scénáře testování

Pro ověření všech vlastností testovaných protokolů využijeme několika scénářů a jejich kombinací. Každý z těchto scénářů má za cíl prověřit konkrétní vlastnost, přičemž bude spouštěn v ideálních podmínkách a v několika variantách uměle zhoršených podmínek. Také budeme sledovat chování se zapnutým a nebo vypnutým šifrováním (pokud to protokol dovoluje).

4.1.1 Ideální podmínky

Za ideální podmínky budeme pro následující úlohy považovat softwarový síťový most dvou virtuálních počítačů, kde jeden pracuje jako server a druhý jako klient. Měli bychom tedy dosáhnout ideálního zpoždění, téměř nulové ztrátovosti, bez špatného pořadí paketů. Tímto stanovíme chování protokolů v ideálních podmínkách a můžeme ho dále porovnat se zhoršenými podmínkami.

4.1.2 Zhoršené podmínky

Upravovat podmínky virtuální sítě budeme pomocí nástroje NetEm, tak abychom se přiblížili podmínkám reálné sítě, ve které budeme volit ztrátovost paketů v rozsahu 0,5%, 2,5% a 10%. Maximální ztrátovost volíme 10%, protože vyšší ztrátovost má fatální vliv na celý přenos. Dále budeme ověřovat, jaký vliv má vzdálenost (zpoždění), na parametry přenosu, a tak volíme hodnoty 100ms, 150ms a 200ms, což přibližně odpovídá přenosu mezi Evropou a Severní Amerikou. Pro nižší hodnoty zpoždění se ve zkušebních testech neprojevili žádné markantní rozdíly mezi protokoly. Budeme také ověřovat, jaký vliv má kolísavost zpoždění (jitter) na rychlost

přenosu, kde bude paket zpožděn o 10ms, 50ms a 150ms s pravděpodobností 25%. Tímto ověříme vliv dílčích negativních vlivů na protokol přenosu webových stránek.

4.1.3 Přenos velkého souboru

Při přenosu velkého souboru (v podobě obrázku) by se mělo projevit pouze navázání spojení, které je u protokolu HTTPS delší z důvodu vyjednávání šifrování, a také režie přenosového protokolu. Při ideálních podmínkách by se tedy výsledky pro jednotlivé protokoly neměly o mnoho lišit. Ve zhoršených podmínkách by již měla být situace jiná, protože se začnou projevovat mechanismy opětovného přenosu ztracených paketů.

4.1.4 Přenos více malých souborů

Pro větší množství malých souborů (tvořených malými obrázky) by se měla znatelně projevit režie použitého přenosového protokolu a komunikace potřebná k navázání spojení. Při zhoršených podmínkách by se měly pouze prohloubit rozdíly, které budou vidět při ideálních podmínkách a měl by být také vidět vliv stejných mechanismů jako v případě velkého souboru.

4.1.5 Kompletní webová stránka

Při přenosu webové stránky složené z různě velkých obrázků, textu a kaskádových stylů by se měly projevit mechanismy komprese dat, vynuceného přenosu a opět režie potřebná k navázání spojení. Jelikož se v tomto scénáři budou přenášet data, která jsou kompresní algoritmy schopné znatelně zmenšit, měly bychom vidět jasné výhody nových aplikačních protokolů, stejně tak zrychlení načítání stránky za pomoci vynuceného přenosu, ke kterému dochází u moderních protokolů viz kapitola 2. V případě vynuceného přenosu server zasílá klientovi data, která si nevyžádal, ale mohl by potřebovat.

4.2 Srovnání SPDY, HTTP2 a HTTP

Pro tuto úlohu volíme webový server Apache, pro který existuje experimentální záplata umožňující využití přenosového protokolu SCTP. Srovnávat budeme vlastnosti prokolů SPDY a všech verzí HTTP s využitím přenosových protokolů TCP a HTTP ve verzi 1.0 a 1.1 přes SCTP. Přistupovat k serveru budeme pomocí prohlížeče Chromium. V této úloze by se měly kladně projevit mechanismy komprese dat

použité pro SPDY a HTTP2, stejně jako mechanismy vynuceného přenosu. U protokolu SPDY není možné zakázat šifrování přenosu, to by se mělo negativně projevit při srovnání s ostatními protokoly, při testu s vypnutým šifrováním.

4.2.1 Manuální konfigurace serveru

Při manuální konfiguraci musíme měnit několik souborů a nastává tu možnost, že opomene změnit nějakou položku a test proběhne špatně. Pokud bychom nastavovali úlohu ručně, museli bychom absolvovat následující kroky:

1. Při měření HTTP a HTTPS verze 1.1, se musíme ujistit, že neběží proces Docker, který poskytuje obsah pro HTTP přes SCTP a proces Docker, který poskytuje obsah přes protokol SPDY. Poté se musíme ujistit, že nemáme povolený protokol HTTP2, tedy že neexistuje soubor „/etc/apache2/mods-available/http2.load“.
2. Při měření HTTP2 se musíme ujistit, že neběží ani jeden z procesů Docker, a že existuje soubor „/etc/apache2/mods-available/http2.load“.

Ukázka nastavení webového serveru (/etc/apache2/sites-enabled/000-default.conf):

```
<IfModule http2_module>
   Protocols h2
   <VirtualHost *:443>
       DocumentRoot /var/www/html/
       SSLEngine on
       SSLOptions +StrictRequire
       SSLCertificateFile /etc/ssl/certs/server.crt
       SSLCertificateKeyFile /etc/ssl/private/server.key
   </VirtualHost>
</IfModule>
<IfModule !http2_module>
   <VirtualHost *:80>
       DocumentRoot /var/www/html/
   </VirtualHost>
   <VirtualHost *:443>
       DocumentRoot /var/www/html/

       SSLEngine on
       SSLOptions +StrictRequire
```

```
        SSLCertificateFile /etc/ssl/certs/server.crt
        SSLCertificateKeyFile /etc/ssl/private/server.key
    </VirtualHost>
</IfModule>
```

4.2.2 Konfigurace serveru nástrojem Ansible

Pokud však zvolíme pro konfiguraci automatizační nástroj Ansible, můžeme předat ověřování jemu. Po zavedení abstrakce v podobě python scriptu, ovládajícího Ansible, vznikne uživatelsky přívětivé rozhraní neumožňující nesprávné nastavení serveru.

Po zadání příkazu „measurement_cli –server apache –protocol http2“ provede toto rozhraní následující kroky:

- Rozhraní podle zadaných parametrů spustí Ansible příkazem „ansible-playbook /home/student/ansible/playbooks/apache.yml –tags http2“
- Anisble podle toho spustí hrací plán (playbook), který obsahuje dvě role. První role se stará o zastavení všech běžících služeb a druhá se stará o nastavení požadované služby. Tyto role jsou reprezentovány soubory „/home/student/ansible/roles/services/apache/tasks/main.yml“ a „/home/student/ansible/roles/services/stop-all/tasks/main.yml“. Jejich obsah je zobrazen níže.

Obsah souboru role pro zastavení všech běžících služeb:

```
--
- name: Ensure all systemd services are stopped
  service:
    enabled: 'False'
    name: ' item '
    state: 'stopped'
  with_items:
    - 'apache2'
    - 'caddy_server_http'
    - 'caddy_server'
    - 'caddy_server_quic'
  tags:
    - 'http'
    - 'ssl'
    - 'http2'
```

```
    - 'spdy'
    - 'quic'
    - 'no-ssl'
    - 'sctp'
- name: Remove docker images if loaded
  docker_container:
    name: ' item '
    state: 'absent'
    force_kill: 'yes'
  with_items:
    - 'sctp-apache'
    - 'spdy-apache'
  tags:
    - 'http'
    - 'ssl'
    - 'http2'
    - 'spdy'
    - 'quic'
    - 'no-ssl'
    - 'sctp'
```

Obsah souboru role sloužící pro nastavení serveru Apache:

```
--
- name: Unload all modules
  command: 'a2dismod  item '
  with_items:
    - 'http2'
    - 'ssl'
  tags:
    - 'http'
    - 'ssl'
    - 'http2'
- name: Enable SSL
  command: 'a2enmod ssl'
  tags:
    - 'ssl'
    - 'http'
```

```
    - 'http2'
- name: Enable HTTP2
  apache2_module:
    name: 'http2'
    state: 'present'
  tags:
    - 'http2'
- name: Start Apache2 server
  service:
    enabled: 'False'
    name: 'apache2'
    state: 'started'
  tags:
    - 'http'
    - 'ssl'
    - 'http2'
```

4.3 Srovnání QUIC, HTTP2 a HTTP

V této úloze použijeme webový server Caddy, díky jeho experimentální podpoře QUIC, zde však chybí podpora SPDY. Budeme tedy porovnávat protokoly QUIC, který používá UDP jako přenosový protokol, a všechny verze HTTP přes TCP. V této úloze by se opět mohlo projevit vynucené šifrování protokolu QUIC obdobně jako u protokolu SPDY v předchozí úloze. Ve zhoršených podmínkách bude zajímavé sledovat mechanismus dopředné opravy chyb protokolu QUIC, především při větším zpoždění.

4.3.1 Konfigurace serveru Caddy

Protože server Caddy cílí na jednoduchou konfiguraci, nemusíme zde využívat Ansible. Konfigurace probíhá pomocí tzv. CaddyFile souboru, který může být společný pro všechny verze HTTP. To zda bude povolený protokol HTTP2 ovlivňuje parametr „-http2=false“ při spouštění serveru.

Ukázka souboru /usr/local/share/caddy_server/Caddyfile (nastavení serveru Caddy):

```
https://transport.protocols.org:443
tls /etc/ssl/certs/server.crt /etc/ssl/private/server.key
root /var/www/html/
```

Obsah role Ansible starající se o nastavení Caddy:

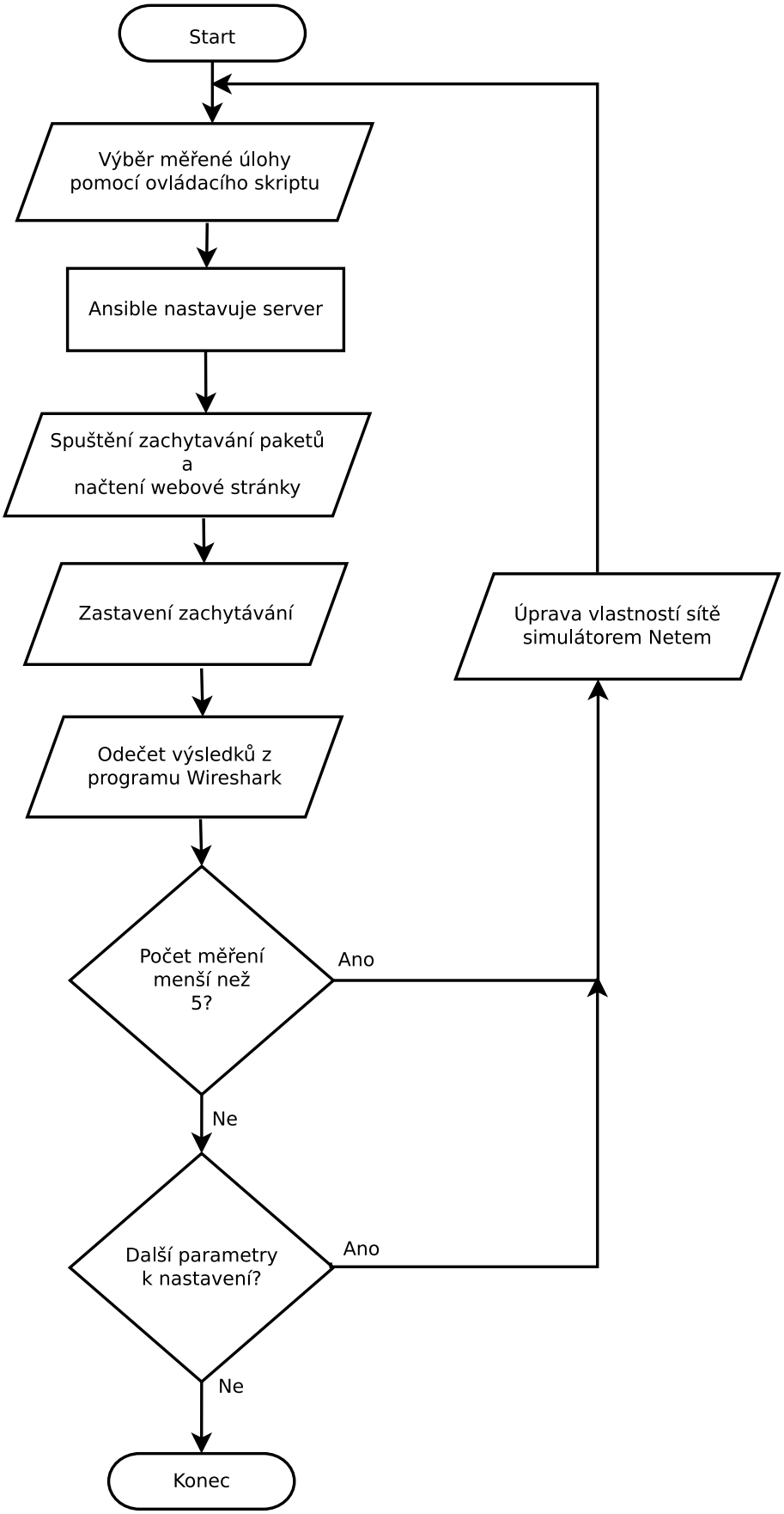
```
--
- name: Push Caddy config with SSL
  template:
    src: 'CaddyFile_ssl.j2'
    dest: '/usr/local/share/caddy_server/Caddyfile'
    owner: 'www-data'
    group: 'www-data'
    mode: '0644'
  tags:
    - 'ssl'
    - 'http2'
    - 'quic'
- name: Push Caddy config without SSL
  template:
    src: 'CaddyFile.j2'
    dest: '/usr/local/share/caddy_server/Caddyfile'
    owner: 'www-data'
    group: 'www-data'
    mode: '0644'
  tags:
    - 'no-ssl'
- name: Start Caddy server with HTTP2 support
  service:
    enabled: 'False'
    name: 'caddy_server'
    state: 'started'
  tags:
    - 'http2'
- name: Start Caddy server with QUIC support
  service:
    enabled: 'False'
    name: 'caddy_server_quic'
    state: 'started'
  tags:
    - 'quic'
- name: Start Caddy server without HTTP2/QUIC support
  service:
```

```
    enabled: 'False'
    name: 'caddy_server_http'
    state: 'started'
  tags:
    - 'ssl'
    - 'no-ssl'
```

4.4 Postup Měření

Postup měření každého scénáře se řídí stejnými kroky, liší se pouze nastavenými podmínkami sítě a nabízeným obsahem serveru. Obecný diagram postupu měření je zobrazený na obrázku 4.1. Měření se odehrává v následujících krocích:

- **Výběr úlohy** - výběr typu měřené úlohy a protokolu (například „Srovnání SPDY, HTTP2 a HTTP“ a protokol SPDY). Výběr se provádí v jednoduché aplikaci napsané v jazyce Python, která pomocí nástroje Ansible nastaví webový server umístěný na serveru.
- **Spuštění zachytávání paketů** - v programu Wireshark spustíme zachytávání paketů na rozhraní „enp0s8“.
- **Spuštění testu** - test se spouští otevřením webové stránky umístěné na serveru. Každý scénář má svoji stránku, která se liší počtem souborů a způsobem jejich umístění (přímé umístění nebo pomocí kaskádových stylů).
- **Odečet výsledků měření** - během testu můžeme sledovat průběh spojení v programu Wireshark. Po skončení načítání stránky zastavíme zachytávání paketů a z hlavní lišty vybereme „Statistics -> Captured file properties“ pro zobrazení počtu zachycených paketů, doby přenosu a přenesených dat.
- **Opakování měření** - z důvodu minimalizace chyby měření provádíme alespoň pět iterací, výsledkem je tedy nakonec aritmetický průměr dílčích výsledků.
- **Úprava vlastností sítě** - vlastnosti sítě se nastavují pomocí nástroje NetEm. Například příkaz „tc qdisc change dev enp0s8 root netem delay 150ms“ nastaví zpoždění 150ms.
- **Konec** - po otestování všech kombinací podmínek se test ukončí.



Obr. 4.1: Obecný diagram postupu měření

4.5 Měření na veřejném webovém serveru

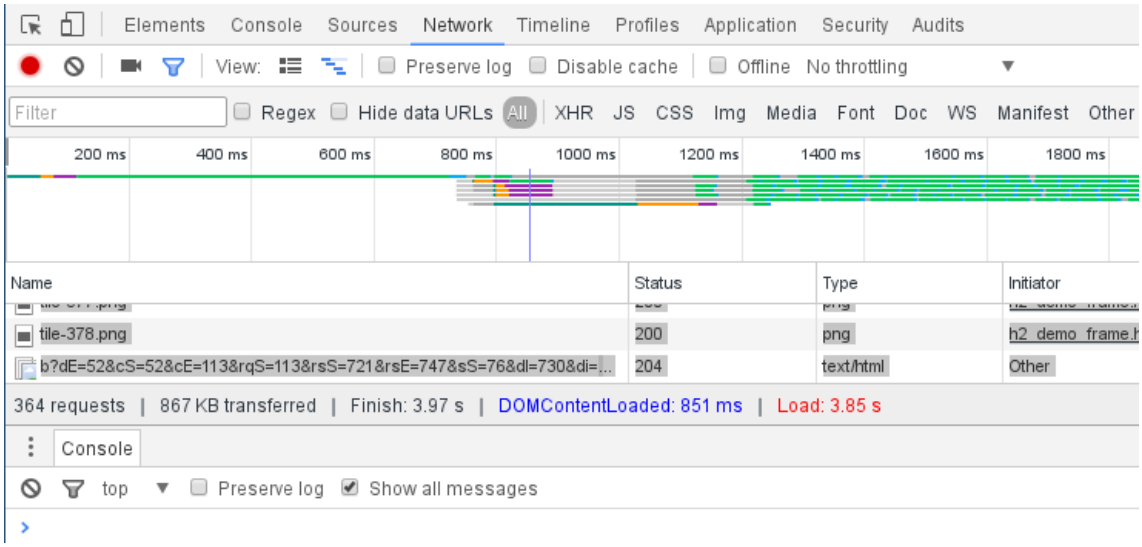
Společnost Akamai, poskytující služby doručování obsahu a cloudové služby, má na svých serverech umístěny dvě stránky poskytující testovací obsah (velký počet malých obrázků) pro porovnání protokolů HTTP 1.1 a HTTP2. Díky tomu, můžeme srovnat výkon těchto dvou protokolů nejen v rámci laboratorní sítě, ale i mezi laboratorní a veřejnou sítí.

Hodnoty v tabulce 4.1 jsou aritmetickým průměrem pěti měření. Můžeme zde pozorovat, že je protokol HTTP2 oproti protokolu HTTPS 1.1 v průměru rychlejší o přibližně 1,5 sekundy a také k přenosu potřebuje menší množství dat. Což je zapříčiněno paralelními proudy a vynuceným zasíláním dat od serveru ke klientovi popisovaným v kapitole 2.

Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTPS 1.1/TCP	1292	5,638	1243956
HTTP2/TCP	1484	4,113	977004

Tab. 4.1: Naměřené hodnoty přenosu oproti veřejnému webovému serveru

Na obrázku 4.2 je vidět, jak vypadá zobrazení průběhu spojení a přenesených dat ve webovém prohlížeči Chromium.



Obr. 4.2: Ukázka zobrazení průběhu měření v programu Chromium

4.6 Laboratorní prostředí

Laboratorní prostředí je tvořeno dvojicí virtuálních počítačů, které jsou spolu spojeny virtuální linkou, aby měření neovlivňovaly venkovní vlivy a abychom byli schopni jednoduše kontrolovat parametry této linky.

Z důvodu experimentální implementace protokolu HTTP přes SCTP, která není dostatečně ošetřena pro různé situace přenosu dochází během měření některých podmínek k pádu prohlížeče Firefox, dle chybové hlášky se jedná o neošetřený přístup do paměti. Protokol HTTP přes SCTP je tedy v tomto laboratorním prostředí obsažen pouze jako doplňkový, pro ukázku jiného transportního protokolu.

Průměrné parametry prostředí při ideálních podmínkách:

- **Zpoždění:** 0,310ms
- **Maximální rozdíl zpoždění (jitter):** 0,039ms
- **Ztrátovost:** 0%

Přenášená data:

- Jeden velký soubor o velikosti 1096869 Bytů (1,1MB) plus velikost HTML souboru 376 bytů.
- 379 malých souborů (zdroj Akamai [4]) o celkové velikosti 659800 Bytů plus velikost HTML souboru 21000 Bytů.
- Jedna webová stránka s obrázky, kaskádovými styly o celkové velikosti 164200 Bytů.

4.6.1 Klient

Klient je tvořen linuxovou distribucí Linux Mint. Na kterém se nachází skript napsaný v jazyce Python sloužící pro nastavení měřené úlohy. Tento skript je pouze abstraktní vrstvou nad nástrojem Ansible, který pomocí „hracích plánů (playbook)“ nastavuje služby na straně serveru. Z důvodu omezené experimentální implementace HTTP nad SCTP a chybějící implementace protokolu SPDY ve webovém prohlížeči Chromium a serveru Apache, je nutné pro spuštění měření využít Docker obrazy s prohlížečem Firefox pro protokol SCTP [5] a obrazy se starou verzí prohlížeče Chromium pro protokol SPDY [5].

Níže vidíme ukázku ovládacího skriptu, konkrétně se jedná o obsah funkce starající se o ošetření výběru úlohy. Díky této funkci dojde vždy k výběru správné role nástroje Ansible a tím k nastavení serveru podle měřené úlohy.

Ukázka ošetření výběru měřené úlohy:

```
if self.PROTOCOL == 'http':
    if self.SERVER == 'apache':
        self.runPlaybook(self.PLAYBOOKS_DIR+self.APACHE_PLAYBOOK, 'http')
    elif self.SERVER == 'caddy' and not self.ENCRYPTION:
        self.runPlaybook(self.PLAYBOOKS_DIR+self.CADDY_PLAYBOOK, 'no-ssl')
    elif self.SERVER == 'caddy':
        self.runPlaybook(self.PLAYBOOKS_DIR+
        self.CADDY_PLAYBOOK, 'ssl')
elif self.PROTOCOL == 'http2':
    if self.SERVER == 'apache':
        self.runPlaybook(self.PLAYBOOKS_DIR+self.APACHE_PLAYBOOK, 'http2')
    elif self.SERVER == 'caddy':
        self.runPlaybook(self.PLAYBOOKS_DIR+self.CADDY_PLAYBOOK, 'http2')
elif self.PROTOCOL == 'http_sctp':
    self.runPlaybook(self.PLAYBOOKS_DIR+self.APACHE_SCTP_PLAYBOOK, 'sctp')
elif self.PROTOCOL == 'spdy':
    self.runPlaybook(self.PLAYBOOKS_DIR+self.APACHE_PLAYBOOK, 'spdy')
elif self.PROTOCOL == 'quic':
    self.runPlaybook(self.PLAYBOOKS_DIR+self.CADDY_PLAYBOOK, 'quic')
```

4.6.2 Server

Server se skládá ze stejné verze linuxové distribuce jako klient. Na straně serveru během měření pouze měníme vlastnosti linky, kterou je klient a server propojen a to z toho důvodu, aby se v zachytávaných paketech na straně klienta objevily i jím odeslané pakety, které se během přenosu ztratily. Pokud bychom prováděli měření a zkreslení vlastností linky na stejné straně, neviděli bychom ztracené pakety a to by ovlivnilo výsledek měření. Služby zde poskytují webové servery Apache a Caddy, s výjimkou protokolů HTTP nad SCTP a SPDY. U nich došlo ke stejnému omezení jako na straně klienta, proto tyto protokoly poskytují Docker obrazy upraveného webového serveru Apache.

5 PRVNÍ ÚLOHA - SROVNÁNÍ PROTOKOLŮ SPDY, HTTP2 A HTTP

5.1 Úvod a cíle úlohy

V této úloze budou porovnávány protokoly SPDY, HTTP2 a HTTP 1.1 ve verzi se šifrováním a bez šifrování na přenosovém protokolu TCP a verze bez šifrování na přenosovém protokolu SCTP. Využití HTTP 1.1 s protokolem SCTP je pouze experimentální, a tak se měření ne vždy může podařit dokončit.

Protokol HTTP 1.1 je v dnešní době stále nejrozšířenější protokol pro přenos webových stránek, zprávy zasílá v podobě čistého textu, což v počítačových sítích může vést k snadnému odcizení přenášených dat. Z tohoto důvodu byl rozšířen nadstavbou HTTPS, která zavádí šifrování viz kapitola 2, obě dvě varianty běžně využívají přenosový protokol TCP. HTTP 1.1 přes transportní protokol SCTP byl uveden pouze jako experimentální, tato implementace se liší pouze v transportním protokolu. SCTP na rozdíl od TCP využívá především paralelních přenosových kanálů (stream) a více tras popisované v kapitole 1. SPDY je protokol z dílny společnosti Google, který využívá k rychlejšímu přenosu především komprimace přenášených dat a vynuceného přenosu, popisované v kapitole 2. Protokol HTTP2 také využívá komprimace dat, vynuceného přenosu a paralelních proudů viz kapitola 2.

Student se zde setká také s možností automatizace konfigurace linuxového serveru, která zde slouží k odstranění chyb při konfiguraci a k u rychlení práce, aby se mohl student soustředit pouze na zadaný úkol.

Cílem úlohy je porovnat výše zmíněné protokoly v ideálních podmínkách, při zvyšujícím se zpoždění, kolísáním zpoždění (jitter) a ztrátovostí na přenosové lince. Vše zmíněné se porovnává při přenosu jednoho velkého souboru, velkého množství menších souborů a kompletní webové stránky. Na těchto scénářích bude student dokazovat, či vyvracet nebo doplňovat teoretické poznatky o výše zmíněných protokolech.

5.2 Příprava a seznámení s laboratorním prostředím

1. Spusťte virtuální počítače s názvy „LinuxMint Client“ a „LinuxMint Server“.

- Na počítači „Client“ se provádějí měření a slouží k ovládání Serveru přes kontrolní skript, který na pozadí využívá automatizační nástroj Ansible pro jeho nastavení.
- Server slouží jako zdroj služeb a také zde v pozdější aplikujeme parametry uměle zhoršující vlastnosti přenosového prostředí.
- Pro přihlášení používejte jméno **student** a heslo **student**. Heslo k účtu **root** je **vutbrno**.

2. Otevřete program Wireshark a seznamte se s jeho ovládáním.
   - Veškerá měření provádějte na rozhraní „enp0s8“.
   - Měřená data jako počet zachycených paketů, doba přenosu a počet zachycených dat najdeme v nabídce „Statistics -> Capture File Properties“. Pokud se vám tato možnost nezobrazuje, označte libovolný zachycený paket a dvakrát zmáčkněte zkratku **Ctrl + D**.
3. Ovládání serveru a výběr úlohy.
   - Otevřete terminál.
   - Výběr úlohy se provádí příkazem „measurement_cli“.
     - Přepínač „–protocol“ slouží k výběru měřeného protokolu.
     - Přepínač „–encryption“ v této úloze nebude potřeba.
     - A u přepínače „–server“ vždy vybíráme možnost **apache**.
   - Po zadaní příkazu například „measurement_cli –server apache – protocol http“ se spustí Ansible a nastaví server tak, aby poskytoval obsah na webovém serveru Apache a byl dostupný přes protokol HTTP.
4. Nastavení negativních vlivů na lince.
   - Spusťte terminál a pomocí příkazu „su“ se přihlaste jako root.
   - Při prvním nastavení negativních vlivů se požívá funkce **add** programu NetEM. Například nastavení zpoždění „tc qdisc **add** dev enp0s8 root netem delay 100ms“.
   - Každá další změna se provádí pomocí funkce **change**. Například změna na ztrátovost 0,5% „tc qdisc **change** dev enp0s8 root netem loss 0.5%“
   - Ovládání zpoždění se provádí pomocí
     „tc qdisc add dev enp0s8 root netem delay **$ČAS$**ms“
   - Ovládání jitter - následující příkaz nastaví zpoždění linky na 1 ms a jitter na zvolenou hodnotu $ČAS$
     „tc qdisc change dev enp0s8 root netem delay 1ms **$ČAS$**ms 25%“
   - Ovládání ztrátovosti
     „tc qdisc change dev enp0s8 root netem loss **$ZTRÁTOVOST$**%“
5. Přes začátkem měření se ujistěte, že je vypnuta podpora následujících funkcí zadáním níže uvedených příkazů do terminálu:
   - Generic Segment Offload - příkazem

„**sudo ethtool –offload enp0s8 gso off**“

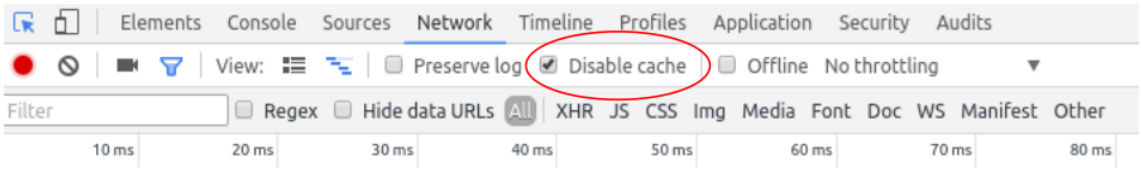
- Generic Reassembly Offload - příkazem
  „**sudo ethtool –offload enp0s8 gro off**“
- TCP Segment Offload - příkazem
  „**sudo ethtool –offload enp0s8 tso off**“
- Tyto funkce zkreslují výsledky protokolů využívajících transportní protokol TCP. Některé funkce protokolu TCP jsou totiž přesunuty až do ovladače síťové karty, proto je program Netem ani Wireshark nevidí. Bez vypnutí těchto funkcí síťové karty bychom viděli pakety o velikosti větší, než 1500 bytů, což je běžně hodnota MTU (Maximum Transmission Unit) nastavená na síťové kartě.

6. **Všechna měření provádějte alespoň 5x a do tabulky vkládejte aritmetický průměr výsledků.**

5.3 Postup měření

5.3.1 Protokol HTTP a HTTPS verze 1.1

1. Vyberte úlohu příkazem „measurement_cli –server apache – protocol http“
2. Otevřete programy Wireshark a Chromium Web Browser z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network** viz obrázek 5.1.



Obr. 5.1: Zakázané ukládání dat do cache - webový prohlížeč Chromium

4. Měření na přenosu jednoho velkého souboru.
   (a) Zapněte odchytávání paketů v programu Wireshark.
   (b) V programu Chromium otevřete stránku „http://192.168.150.1/one_big“ pro měření na protokolu HTTP a nebo „https://192.168.150.1/one_big“ pro měření se zapnutým šifrováním.
   (c) Během přenosu sledujte chování přenosového protokolu.
   (d) Po skončení načítání stránky vypněte zachytávání paketů.

(e) Odečtěte počet přenesených paketů, dobu přenosu a objem přenesených dat.

(f) Postupně nastavujte zpoždění na hodnoty 100, 150 a 200ms, kolísavé zpoždění (jitter) 10ms, 50ms a 150ms, ztrátovost v hodnotách 0,5%, 2,5% a 10%. Po každé změně negativních vlivů opakujte body (a) až (e).

5. Měření na přenosu velkého množství malých souborů.
   (a) V programu Chromium otevřete stránku „http://192.168.150.1/small_objects“ pro HTTP a nebo „https://192.168.150.1/small_objects“ pro HTTPS.
   (b) Postup měření je stejný jako v bodě 4.
6. Přenos kompletní webové stránky.
   (a) V programu Chromium otevřete stránku „http://192.168.150.1/full_page“ pro HTTP a nebo „https://192.168.150.1/full_page“ pro HTTPS.
   (b) Postup měření je stejný jako v bodě 4.
7. Naměřené údaje graficky zpracujte a porovnejte s ostatními protokoly.

5.3.2 Protokol HTTP2

1. Vyberte úlohu příkazem „measurement_cli –server apache – protocol http2“
2. Otevřete programy Wireshark a Chromium Web Browser z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network**.
4. Používejte pouze adresu „https://192.168.150.1/“.
5. Postup měření je stejný jako v sekci 5.3.1 v bodech 4 až 6.
6. Naměřené údaje graficky zpracujte a porovnejte s ostatními protokoly.

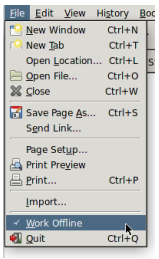
5.3.3 Protokol SPDY

1. Vyberte úlohu příkazem „measurement_cli –server apache – protocol spdy“
2. Otevřete programy Wireshark a Chromium SPDY z hlavní plochy a po otevření terminálu zadejte heslo **student**.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network** viz obrázek
4. Používejte pouze adresu „http://192.168.150.1/“.
5. Postup měření je stejný jako v sekci 5.3.1 v bodech 4 až 6.

6. Naměřené údaje graficky zpracujte a porovnejte s ostatními protokoly.

5.3.4 Protokol HTTP přes SCTP

1. Vyberte úlohu příkazem „measurement_cli –server apache – protocol http_sctp“
2. Otevřete programy Wireshark a Firefox SCTP z hlavní plochy a po otevření terminálu zadejte heslo **student**.
3. V programu Firefox vypněte práci v režimu offline, viz obrázek 5.2.
   - Odškrtněte položku „File -> Work Offline“.



Obr. 5.2: Vypnutí práce v režimu offline - prohlížeč Firefox

4. Po každém měření nezapomeňte vymazat cache.
   - Klávesová zkratka **Ctrl + Shift + Del** a kliknout na možnost „Clear Private Data Now“.
5. Používejte pouze adresu „http://192.168.150.1/“.
6. Postup měření je stejný jako v sekci 5.3.1 v bodech 4 až 6.
7. Naměřené údaje graficky zpravujte a porovnejte s ostatními protokoly.

5.3.5 Doplňující otázky

1. Došlo u některého z protokolů ke kompresi? Pokud ano, dokažte.
2. Na jakém z měřených scénářů můžeme nejlépe dokázat paralelní přenos dat a u jakých protokolů k němu dochází?
3. Odhadněte z naměřených údajů, který z protokolů se hodí pro jaké použití. Svůj výběr zdůvodněte.
4. Na přenosu velkého počtu souborů otestujte jaký vliv na parametry přenosu mají následující kombinace negativních vlivů:
   - Ztrátovost paketů 2,5% a zpoždění 100ms.
     „tc qdisc change dev enp0s8 root netem loss 2.5% delay 100ms“
   - Ztrátovost paketů 2,5% a jitter 10ms.
     „tc qdisc change dev enp0s8 root netem loss 2.5% delay 1ms 10ms 25%“

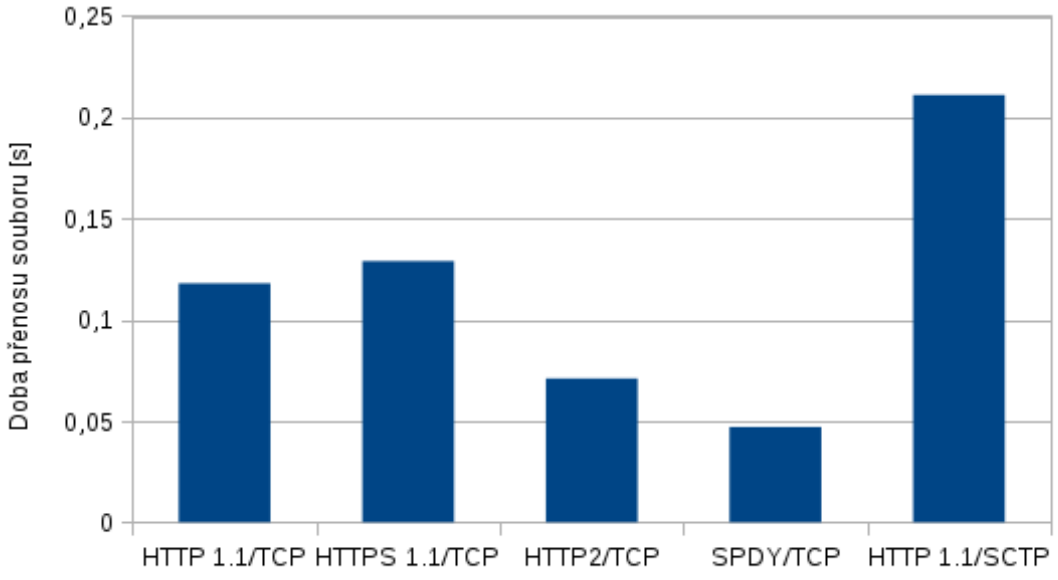
5.4 Vzorové vypracování

5.4.1 Přenos velkého souboru

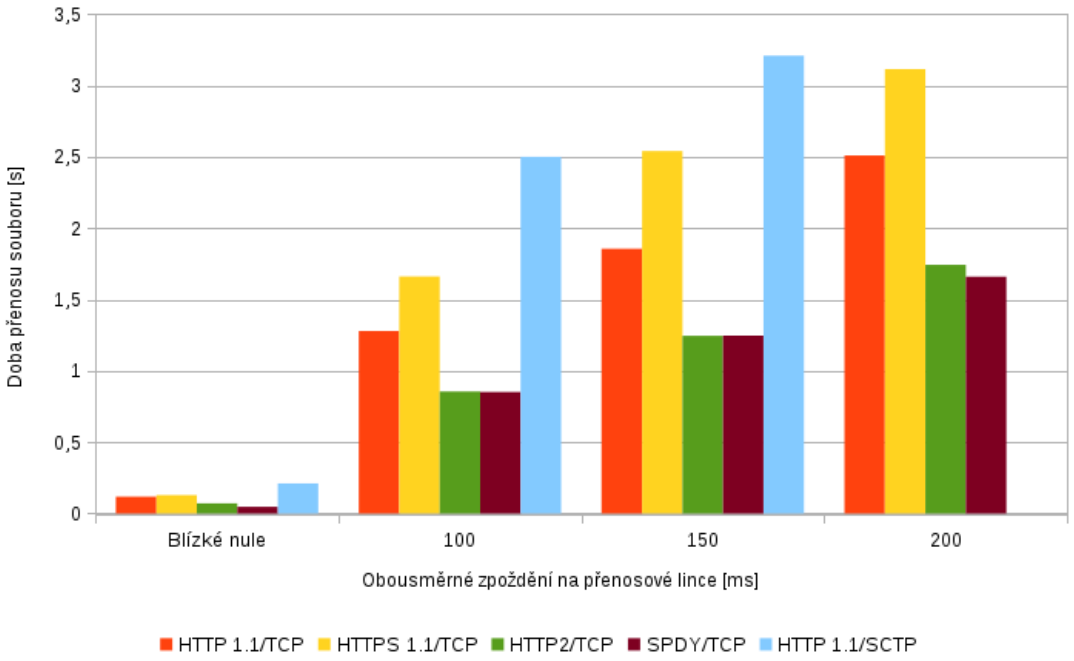
Ideální průběh Při ideálním stavu linky se jako nejlepší projevil protokol SPDY, pomocí kterého přenos trval 47ms. Druhým nejrychlejším byl v tomto testu protokol HTTP2 následovaný protokoly HTTP a HTTPS verze 1.1, jak můžeme vidět z grafu na obrázku 5.3. HTTP 1.1 přes transportní protokol SCTP vyšlo z testu nejhůře, což je zapříčiněno přílišnou režií, protože tento protokol použil k přenesu dat o téměř 50% více paketů než ostatní protokoly.

Protokol	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	842	0,118	1154233
HTTPS 1.1/TCP	913	0,129	1163219
HTTP2/TCP	802	0,071	1152916
SPDY/TCP	808	0,047	1152851
HTTP 1.1/SCTP	1224	0,211	1176156

Tab. 5.1: Naměřené hodnoty na webovém serveru Apache při přenosu jednoho souboru - bez uměle zhoršených přenosových podmínek (průměr pěti měření)



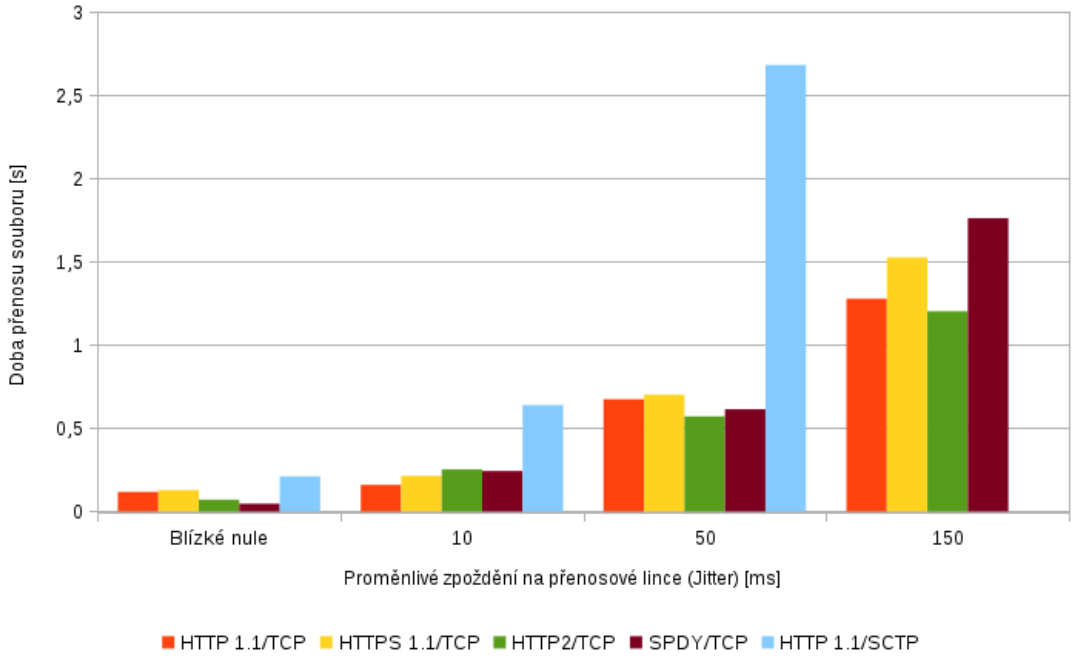
Obr. 5.3: Graf doby přenosu protokolů na webovém serveru Apache při ideálním stavu linky (půměr pěti měření)



Obr. 5.4: Graf doby přenosu protokolů na webovém serveru Apache ovlivněné zpožděním na lince (průměr pěti měření)

Vliv zpoždění Zvýšení zpoždění se pro každý z protokolů projevilo nárůstem doby přenosu. Z grafu na obrázku 5.4 je patrné, že protokoly SPDY a HTTP2 reagují na zvyšující se zpoždění velice podobně a vykazují téměř stejné parametry přenosu. Je zde také vidět, že má zpoždění větší vliv na protokoly HTTP a HTTPS 1.1, což je zapříčiněno neschopností paralelního přenosu dat, se kterým se setkáváme až u protokolů HTTP2 a SPDY, popisovaným v kapitole 2. U protokolu HTTP1.1 přes SCTP se bohužel nepodařilo měření dokončit z důvodu programové chyby a pádů webového prohlížeče, toto je pravděpodobně zapříčiněno nesprávně ošetřenou experimentální implementací protokolu SCTP. Protokol HTTP 1.1 v tomto měření vykazoval čistě lineární nárůst doby přenosu v závislosti na umělém zpoždění přenosové trasy.

Jitter Proměnlivost zpoždění se u všech měřených protokolů projevila především nárůstem počtu duplicitních paketů souvisejících s opakovanými přenosy. Hraniční měřená hodnota jitter (150ms) se projevila nejhůře na protokolu SPDY, kde došlo k markantnímu nárůstu doby přenosu a byl tedy pomalejší než ostatní měřené protokoly. viz obrázek 5.5. Verze 1.1 protokolu HTTP ve variantě bez šifrování a se šifrováním měly podobný průběh, kde varianta bez šifrování byla mírně rychlejší do hodnoty jitter 150ms, při ní byl rozdíl již patrnější. Měření se bohužel opět nepodařilo dokončit celé u protokolu HTTP1.1 přes SCTP, z tohoto důvodu chybí v



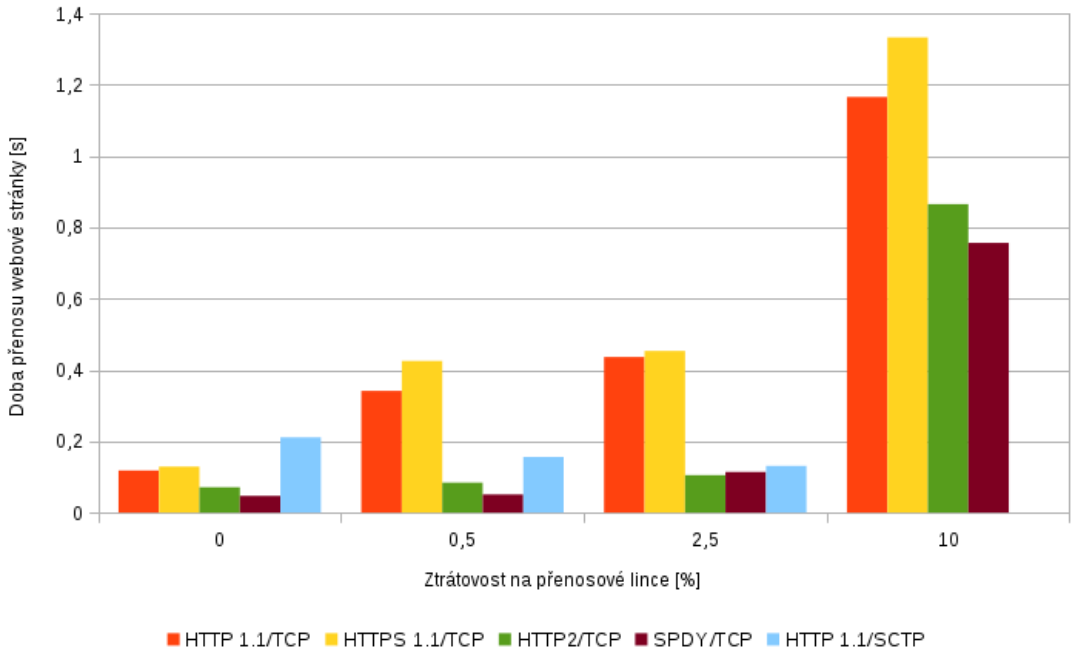
Obr. 5.5: Graf doby přenosu protokolů na webovém serveru Apache na lince s proměnlivým zpožděním (průměr pěti měření)

grafech.

```
349 0.248357342   192.168.150.2   192.168.150.1   TCP        86 [TCP Window Update] 49892 → 443 [ACK] Seq=1503 Ac…
350 0.248411430   192.168.150.1   192.168.150.2   TCP      4410 [TCP Out-Of-Order] 443 → 49892 [ACK] Seq=1068002 …
351 0.248418722   192.168.150.2   192.168.150.1   TCP        86 [TCP Dup ACK 347#1] 49892 → 443 [ACK] Seq=1503 Ac…
352 0.248557861   192.168.150.1   192.168.150.2   TCP     14546 [TCP Out-Of-Order] 443 → 49892 [ACK] Seq=1083930 …
353 0.248572531   192.168.150.2   192.168.150.1   TCP        86 [TCP Dup ACK 347#2] 49892 → 443 [ACK] Seq=1503 Ac…
354 0.248864695   192.168.150.1   192.168.150.2   TCP      5858 [TCP Out-Of-Order] 443 → 49892 [ACK] Seq=1078138 …
355 0.248881058   192.168.150.2   192.168.150.1   TCP        78 [TCP Dup ACK 347#3] 49892 → 443 [ACK] Seq=1503 Ac…
356 0.252528930   192.168.150.1   192.168.150.2   TLSv1.2 10202 [TCP Fast Retransmission] Ignored Unknown Record
```

Obr. 5.6: Projev proměnlivého zpoždění při přenosu velkého souboru protokolem HTTP 1.1

Ztrátovost Ztrátovost 0,5% a 2,5% se u protokolů SPDY a HTTP2 projevila pouze mírným nárůstem doby přenosu, při hodnotě ztrátovosti 10% se doba přenosu skokově zvýšila, jak můžeme pozorovat na obrázku 5.7. Pro protokoly HTTP 1.1 a HTTPS 1.1 se zvýšení ztrátovosti z 0% na 0,5% projevilo zvýšením doby přenosu až na téměř trojnásobek, viz tabulka A.2, mezi hodnotami ztrátovosti 0,5% a 2,5% došlo pouze k minimálnímu nárůstu a na hodnotě ztrátovosti 10% došlo opět k nárůstu až na trojnásobek doby přenosu, to souvisí s náhodností výběru zahozeného paketu. U protokolu HTTP 1.1 přes SCTP dokonce jejím snížením, což může být způsobeno chybou implementace, která je v tomto případě pouze experimentální. Měření se opět nepodařilo pro tento protokol dokončit, protože se přenos při hodnotě ztrátovosti paketů 10% pokaždé zasekl.



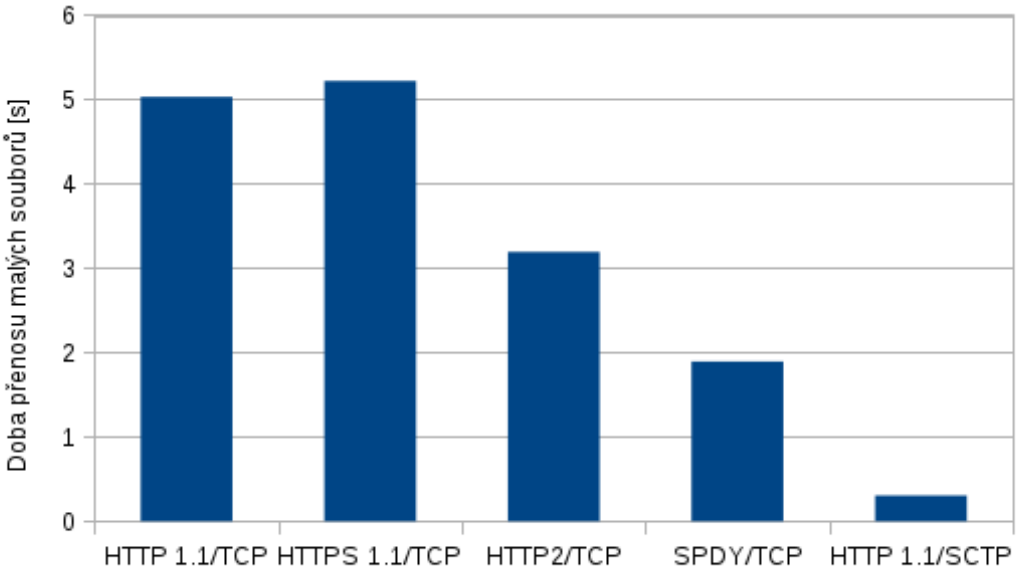
Obr. 5.7: Graf doby přenosu protokolů na webovém serveru Apache při zvýšené ztrátovosti (průměr pěti měření)

5.4.2 Velký počet malých souborů

Ideální průběh Přenos velkého počtu souborů bez negativních vlivů na lince zvládl nejlépe protokol HTTP 1.1 přes SCTP, který byl o 4,9 vteřiny rychlejší než nejpomalejší protokol (HTTP 1.1 přes TCP) a o 1,6 vteřiny rychlejší než druhý nejrychlejší protokol SPDY, viz 5.8. Protokoly HTTP 1.1 a HTTPS 1.1 se projevily jako velice vyrovnané, kde protokol HTTPS byl rychlejší s rozdílem přibližně 4%, což může být zapříčiněno nepřesností měření. U protokolu HTTP2 byl potřeba k přenosu nejmenší počet paketů a přeneseno nejméně dat z důvodu komprese, jak můžeme vidět v tabulce 5.2.

Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTP 1.1/TCP	1619	5,029	1019004
HTTPS 1.1/TCP	2610	5,218	1180963
HTTP2/TCP	1101	3,191	789415
SPDY/TCP	1221	1,892	793271
HTTP 1.1/SCTP	1829	0,305	1047804

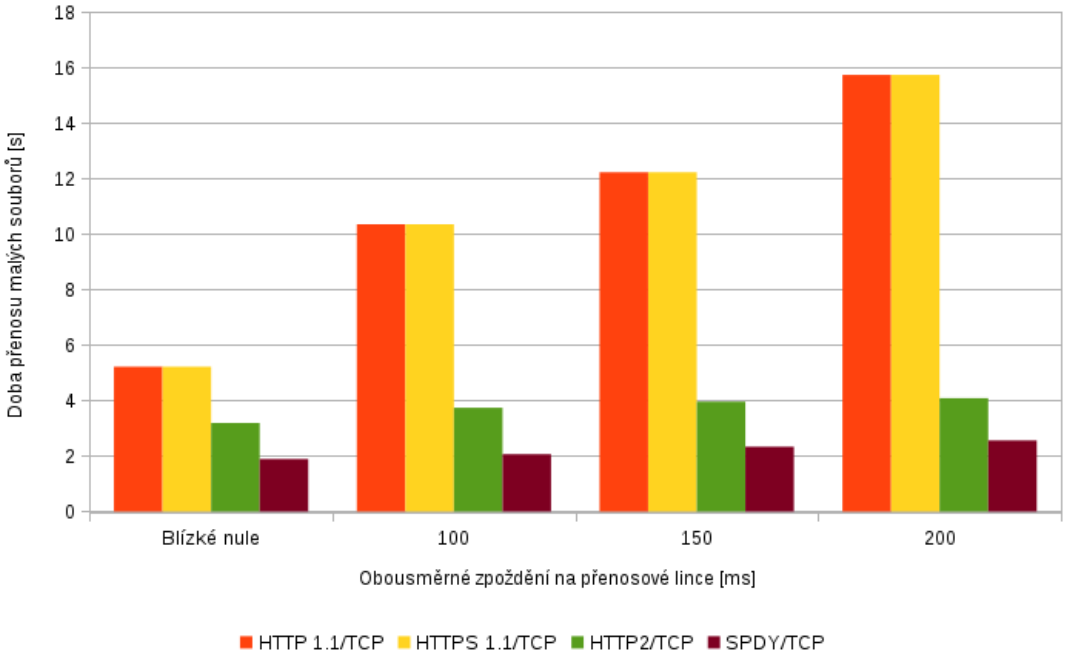
Tab. 5.2: Naměřené hodnoty na webovém serveru Apache při přenosu velkého množství souborů - bez uměle zhoršených přenosových podmínek (průměr pěti měření)



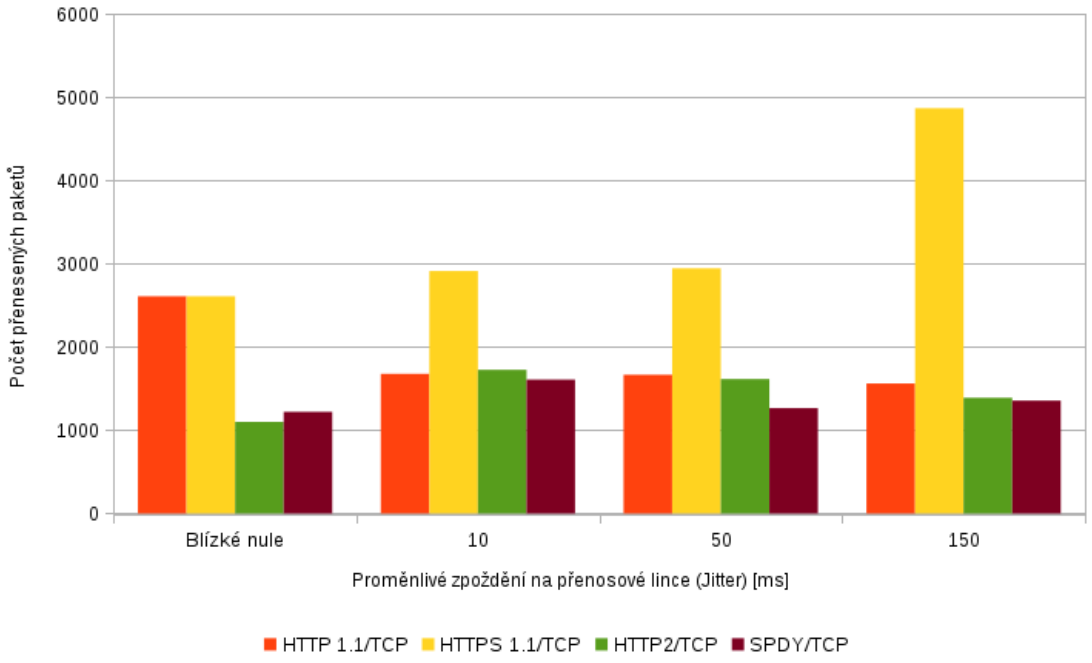
Obr. 5.8: Graf doby přenosu (velký počet souborů) protokolů na webovém serveru Apache při ideálním stavu linky (průměr pěti měření)

Zpoždění na lince Zvyšující se zpoždění se u protokolů využívajících transportní protokol TCP projevilo snížením počtu přenášených paketů, viz tabulka A.4. To bylo zapříčiněno snahou protokolu o snížení počtu potvrzovacích zpráv, protože právě čekání na potvrzení má při vysokém zpoždění na lince největší vliv na zpoždění celého přenosu. Z grafu na obrázku 5.9 je vidět, že protokoly HTTP2 a SPDY vykázaly v tomto měření pouze minimální nárůst přenosové doby, což je zapříčiněno využitím paralelního přenosu dat popisovaného v kapitole 2. U protokolů HTTP 1.1 a HTTPS 1.1, které nemají podporu paralelního přenosu, vedlo zvyšování zpoždění k lineárnímu nárůstu doby přenosu. Měření nebylo možné provést pro protokol SCTP z důvodu pádu webového prohlížeče.

Proměnlivost zpoždění Zajímavý je v tomto testu rozdíl mezi chováním protokolu HTTP, u něhož došlo ke snížení počtu přenesených paketů, oproti protokolu HTTPS, kde došlo k razantnímu nárůstu při hodnotě jitter 150ms, jak můžeme pozorovat na obrázku 5.10, což vedlo také k velkému rozdílu přenosové doby. Při této hodnotě přenos protokolem HTTP trval 8 sekund, ale protokolem HTTPS 11s, rozdíl pro nižší hodnoty jitter byl však vždy okolo jedné vteřiny. Je tedy zřejmé, že jitter okolo hodnoty 150ms má již pro protokol HTTPS v tomto scénáři fatální důsledky. Měření se opět nepodařilo provést u protokolu SCTP.

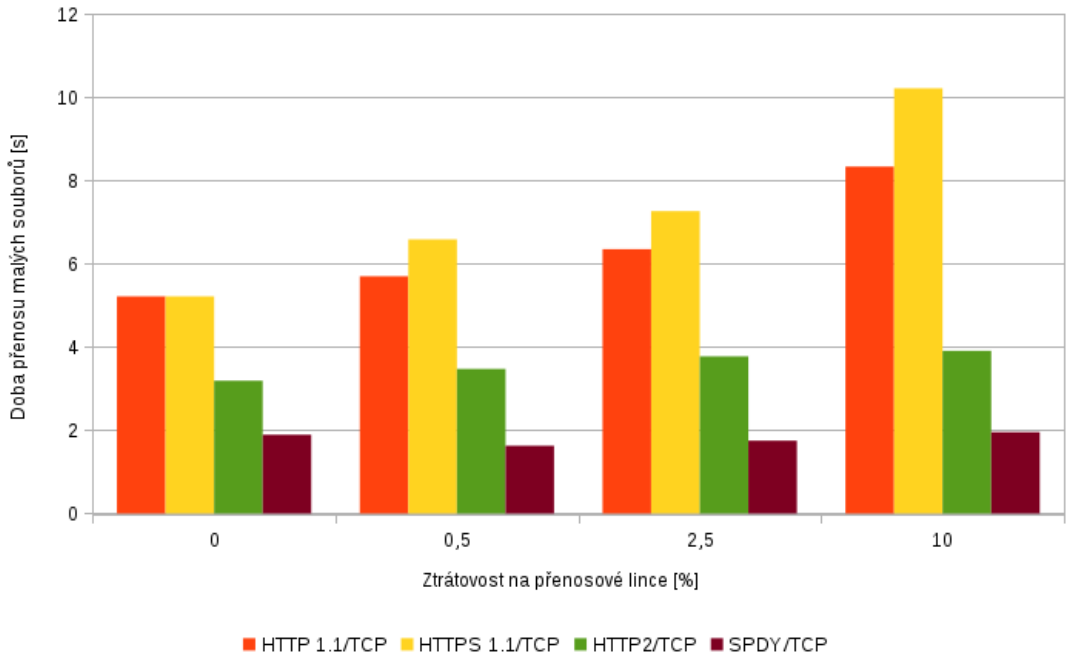


Obr. 5.9: Graf doby přenosu (velký počet malých souborů) protokolů na webovém serveru Apache při zpoždění na lince (průměr pěti měření)



Obr. 5.10: Graf počtu přenesených paketů (velký počet malých souborů) protokolů na webovém serveru Apache při proměnlivém zpoždění (průměr pěti měření)

Ztrátovost Vliv ztrátovosti je mnohem mírnější pro protokoly HTTP a HTTPS, než to mu bylo při přenosu velkého souboru, viz Obr. 5.11. Nezvyklým úkazem při tomto měření je pokles doby přenosu při ztrátovosti 0,5% a 2,5% protokolu SPDY, což je pravděpodobně způsobeno chybou měření. Doba přenosu tímto protokolem se pohybovala kolem stejné hodnoty, projevil se tedy jako velice stabilní a pokud bychom započítali chybu měření, tak můžeme prohlásit, že prokázal konstantní průběh vztaženo na dobu přenosu. U protokolu SCTP opět nebylo možné měření uskutečnit.



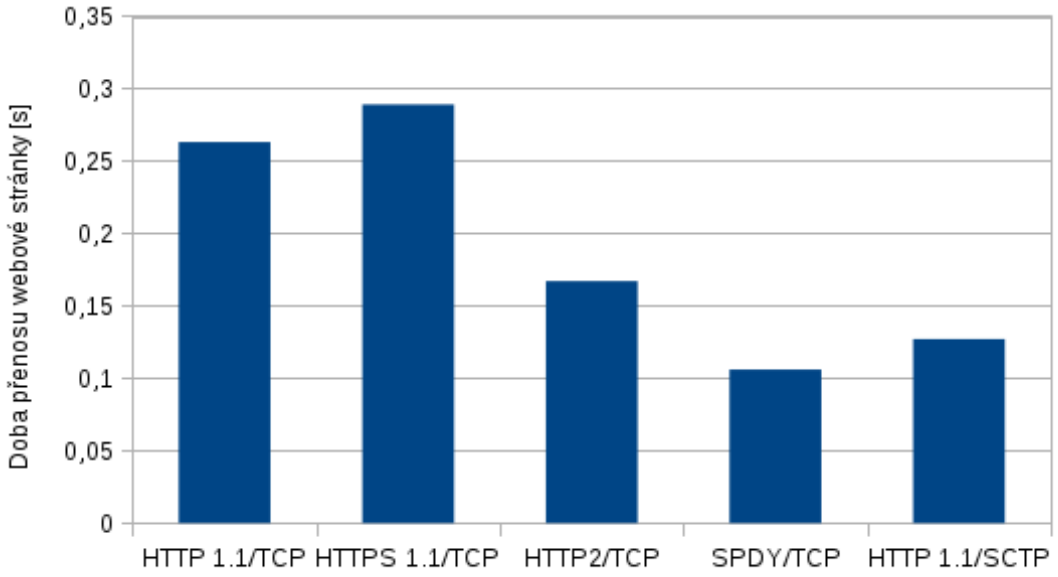
Obr. 5.11: Graf doby přenosu (velký počet malých souborů) protokolů na webovém serveru Apache při ztrátovosti (půměr pěti měření)

5.4.3 Kompletní webová stránka

Ideální průběh Při přenosu v ideálních podmínkách si nejlépe vedl protokol SPDY následovaný protokolem HTTP přes SCTP s rozdílem 21ms. to odpovídá nárůstu o 20%. Protokoly z HTTP a HTTPS verze 1.1 na transportním protokolu TCP se pohybovaly v mezi hodnotami 260ms až 290ms, což je více než dvojnásobná hodnota oproti protokolu SPDY. Protokolu HTTP2 trval přenos průměrně 167ms s nejnižším počtem přenesených paketů. Grafické znázornění rozdílu mezi protokoly je zobrazeno na obrázku 5.12.

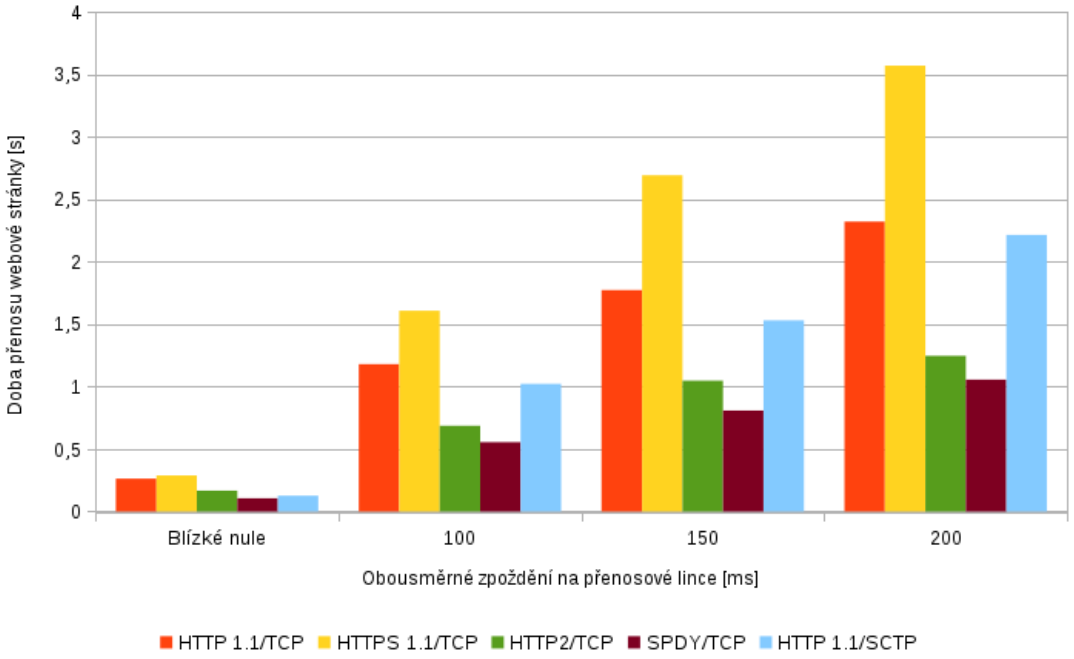
Protokol	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	215	0,263	180472
HTTPS 1.1/TCP	352	0,289	202411
HTTP2/TCP	177	0,167	173201
SPDY/TCP	192	0,106	171689
HTTP 1.1/SCTP	223	0,127	189812

Tab. 5.3: Naměřené hodnoty na webovém serveru Apache při přenosu kompletní webové stránky - bez uměle zhoršených přenosových podmínek (průměr pěti měření)



Obr. 5.12: Graf doby přenosu (kompletní stránka) protokolů na webovém serveru Apache při ideálním stavu linky (průměr pěti měření)

Zpoždění V tomto měření se neprojevilo snížení počtu přenášených paketů u protokolu HTTPS 1.1, jak tomu bylo v předchozím měření, což nasvědčuje, že se tato vlastnost neprojevuje při krátkém přenosu. Negativní podmínky naopak zapříčinily nárůst přenášených paketů u protokolů HTTP 1.1 a HTTPS 1.1. Na protokoly SPDY a HTTP2 mělo narůstající zpoždění vliv v podobně lineárního nárůstu doby přenosu, viz graf na obrázku 5.13, což je rozdíl oproti předchozímu měření, při kterém tyto protokoly zachovaly téměř konstantní dobu přenosu.



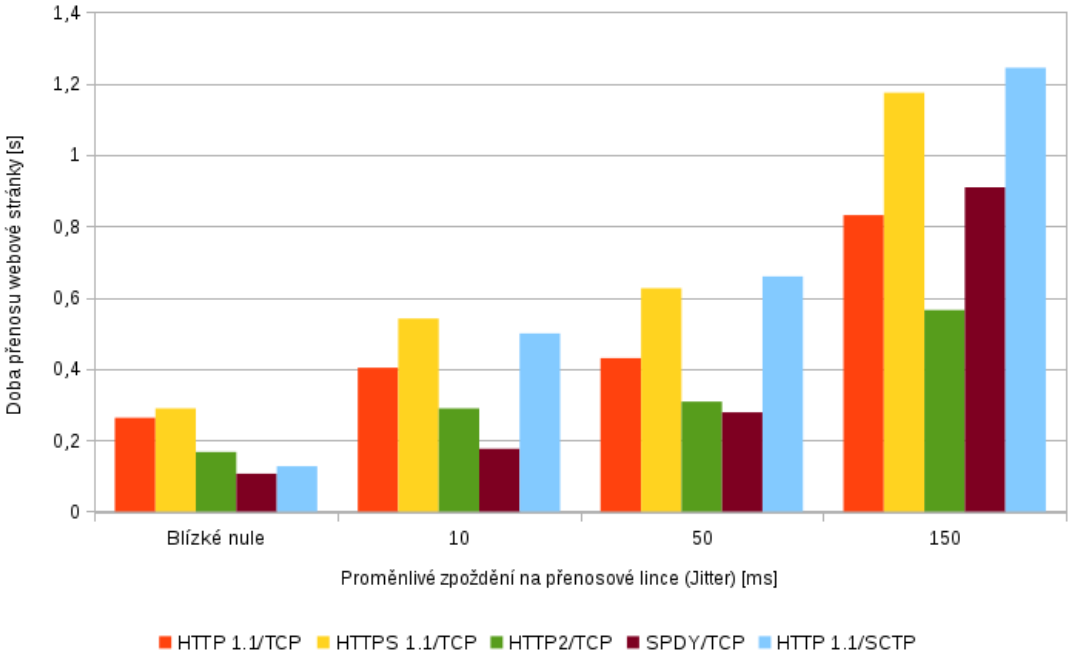
Obr. 5.13: Graf doby přenosu (kompletní stránka) protokolů na webovém serveru Apache při zpoždění na lince (průměr pěti měření)

Proměnlivé zpoždění Jitter se projevil na přenosu stejně jako v předchozích měřeních a to především nárůstem znovu poslaných paketů a duplicitních potvrzení, které vedly ke zvýšení počtu přenesených paketů a nárůstu času, potřebného pro přenos dat. Na protokol HTTP přes SCTP mělo proměnlivé zpoždění (jitter) vliv v podobě lineárního nárůstu zpoždění, jak můžeme vidět na obrázku 5.15. Protokol SPDY se v tomto případě choval stejně jako při přenosu jednoho souboru a to tak, že při hodnotě jitter 150ms došlo ke rapidnímu nárůstu doby přenosu.

```
2065 0.635728610   192.168.150.1       192.168.150.2       SCTP     1470 DATA (retransmission)
2066 0.635754178   192.168.150.2       192.168.150.1       SCTP       70 SACK
2067 0.635770189   192.168.150.1       192.168.150.2       SCTP     1470 DATA
2068 0.635979763   192.168.150.1       192.168.150.2       SCTP     1470 DATA (retransmission)
2069 0.635997801   192.168.150.2       192.168.150.1       SCTP       66 SACK
2070 0.638756863   192.168.150.1       192.168.150.2       SCTP     1470 DATA (retransmission)
2071 0.638805228   192.168.150.2       192.168.150.1       SCTP       66 SACK
```

Obr. 5.14: Vliv jitter 10ms na transportní protokol SCTP

Ztrátovost Pro hodnoty ztrátovosti paketů 0,5% a 2,5% si protokoly SPDY a HTTP2 opět zachovávají téměř konstantní dobu přenosu, avšak při hodnotě ztrátovosti 10% již dochází ke zvýšení doby přenosu na téměř dvojnásobek pro oba protokoly, viz obrázek 5.16. Protokoly HTTP a HTTPS verze 1.1 vykazují téměř lineární nárůst doby přenosu. U protokolu HTTP přes SCTP dochází opět k mírnému

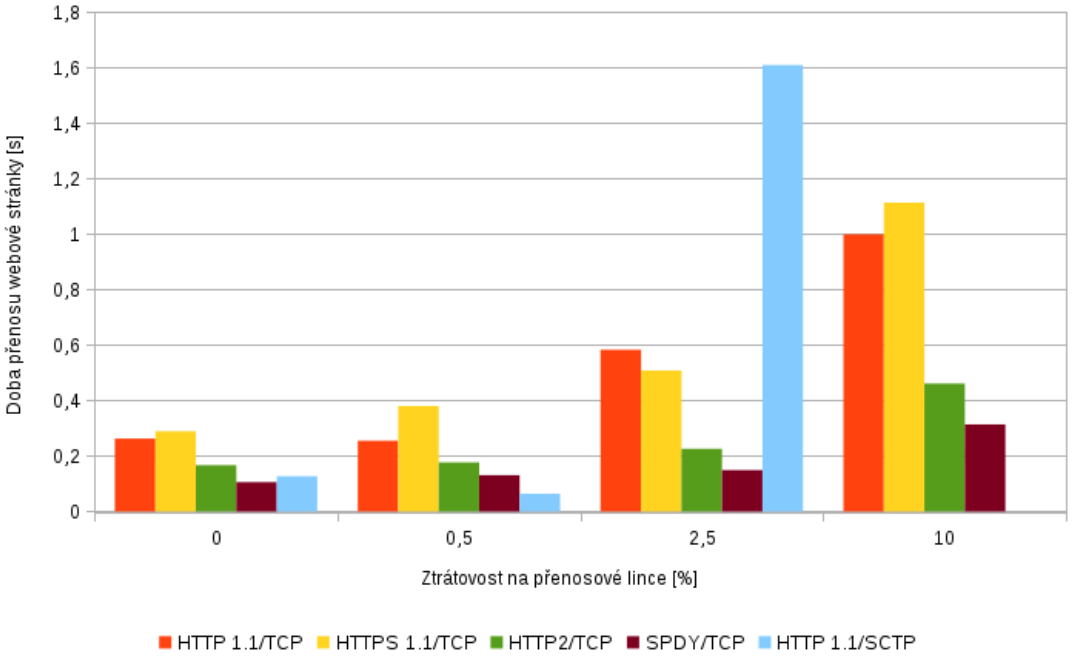


Obr. 5.15: Graf doby přenosu (kompletní stránka) protokolů na webovém serveru Apache při proměnlivém zpoždění (půměr pěti měření)

snížení přenosové doby na hodnotě ztrátovosti 0,5%, při 2,5% však dochází ke skokovému nárůstu a na 10% ztrátovosti se již nepodařilo pro tento protokol dokončit měření.

5.4.4 Doplňující otázky

1. Došlo u některého z protokolů ke kompresi? Pokud ano, dokažte.
   - Ne, u žádného z měřených protokolů se prokazatelně neprojevila komprese.
2. Na jakém z měřených scénářů můžeme nejlépe dokázat paralelní přenos dat a u jakých protokolů k němu dochází?
   - Při přenosu velkého počtu souborů a hlavně při narůstajícím zpoždění, kde je vidět, že nárůst přenosové doby je mnohonásobně pomalejší, než u protokolů HTTP a HTTPS ve verzi 1.1. K paralelnímu přenosu by mělo docházet u protokolů HTTP2, SPDY a HTTP přes SCTP, bohužel u SCTP se nepodařilo dokončit ani jedno z měření se zhoršenými podmínkami, pro přenos velkého počtu souborů. Při ideálních podmínkách si však vedl nejlépe, proto by se dalo předpokládat, že k paralelnímu přenosu také dochází. Nutno také podotknout, že u SPDY a HTTP2 se o paralelní přenos starají aplikační protokoly, kdežto u HTTP přes SCTP



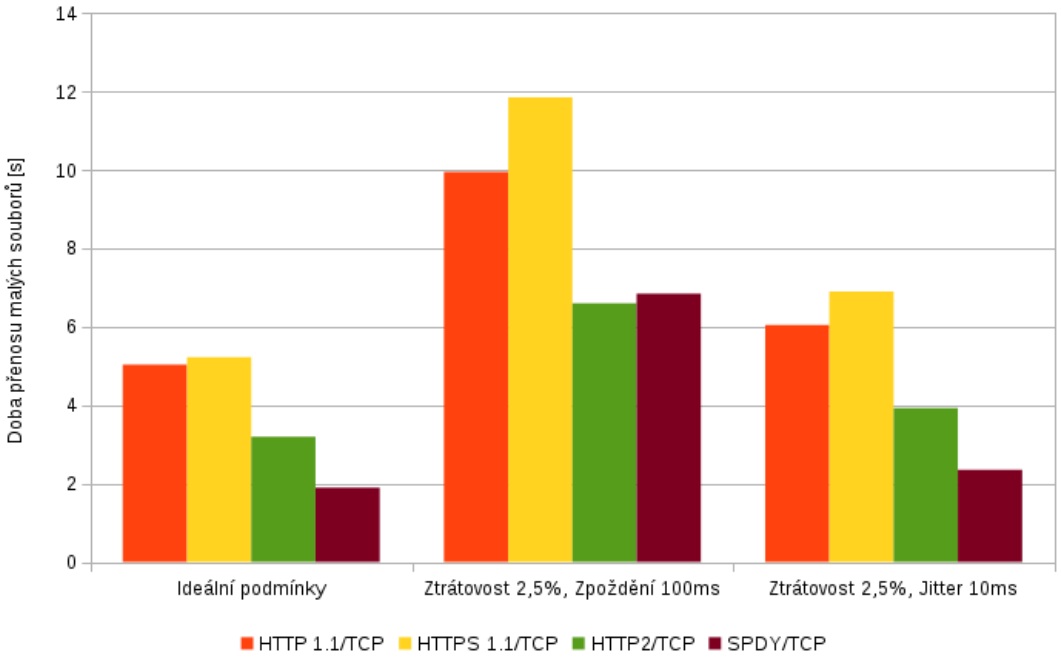
Obr. 5.16: Graf doby přenosu (kompletní stránka) protokolů na webovém serveru Apache při ztrátovosti (průměr pěti měření)

se o něj stará transportní protokol.

3. Odhadněte z naměřených údajů, který z protokolů se hodí pro jaké použití. Svůj výběr zdůvodněte.
   - Z měřených protokolů se jako nejvhodnější jeví protokol SPDY, jehož vývoj byl však ukončen. Jako další protokol vhodný pro univerzální použití se jeví protokol HTTP2. Pro přenos velkého souboru, jsou stále dobré protokoly HTTP a HTTPS ve verzi 1.1 a to hlavně díky své jednoduchosti implementace. Tyto protokoly však ztrácí při přenosu velkého počtu souborů, a to především díky chybějící podpoře vynuceného přenosu.
4. Na přenosu velkého počtu souborů otestujte, jaký vliv na parametry přenosu mají kombinace negativních vlivů.
   - Zatímco kombinace zpoždění a ztrátovosti má na protokoly HTTP a HTTPS verze 1.1 vliv stejný jako součet těchto vlivů individuálně, u protokolů HTTP2 a SPDY dochází k markantnímu nárůstu přenosové doby, jak můžeme pozorovat v grafu na obrázku 5.17. Při přenosu protokolem HTTP2 došlo k nárůstu až o 3 vteřiny více což je zhruba 85%, avšak u negativních vlivů ztrátovosti a zpoždění došlo pouze k nárůstu přenosové doby o zhruba 14% u každého. U protokolu SPDY byl tento jev ještě větší.
   - Kombinace jitter a ztrátovost paketů se u všech protokolů projevila jako

součet dílčích vlivů.

- Měření protokolu HTTP přes SCTP se nepodařilo dokončit z důvodu pádů prohlížeče.



Obr. 5.17: Graf doby přenosu (velkého počtu malých souborů) protokolů na webovém serveru Apache při kombinaci negativních vlivů (půměr pěti měření)

6 DRUHÁ ÚLOHA - SROVNÁNÍ PROTOKOLŮ QUIC, HTTP2 A HTTP

6.1 Úvod a cíle úlohy

V této úloze budou porovnávány protokoly QUIC, HTTP2 a HTTP 1.1 ve verzi se šifrováním a bez šifrování. QUIC je velice nový protokol z dílny firmy Google, v této úloze je využíván v experimentální implementaci ve webovém prohlížeči Caddy. Jedná se o lehký webový prohlížeč kompletně napsaný v jazyce Go.

Protokol HTTP 1.1 je v dnešní době stále nejrozšířenější protokol pro přenos webových stránek, zprávy zasílá v podobě čistého textu, což v počítačových sítích může vést k snadnému odcizení přenášených dat. Z tohoto důvodu byl rozšířen nadstavbou HTTPS, která zavádí šifrování viz kapitola 2, obě dvě varianty běžně využívají přenosový protokol TCP. Protokol HTTP2 je novější variantou těchto protokolů a odlišuje se především přenosem komprimovaných dat v binární podobě, paralelním přenosem a vynucených přenosem částí webové stránky serverem popsanými v kapitole 2. QUIC je nový protokol z laboratoří společnosti Google. Odlišuje se především využitím transportního protokolu UDP oproti ostatním protokolů, které využívají protokol TCP. Tímto se snaží dosáhnout efektivnějšího přenosu. Některé části pro zajištění spolehlivého přenosu z protokolu TCP implementuje na aplikační úrovni. Dále se například odlišuje dopřednou opravou chyb, popsanou v kapitole 2.

Student se zde setká také s možností automatizace konfigurace linuxového serveru, která zde slouží k odstranění chyb při konfiguraci a k u rychlení práce, aby se mohl student soustředit pouze na zadaný úkol.

Cílem úlohy je porovnat výše zmíněné protokoly v ideálních podmínkách, při zvyšujícím se zpoždění, kolísáním zpoždění (jitter) a ztrátovostí na přenosové lince. Vše zmíněné se porovnává při přenosu jednoho velkého souboru, velkého množství menších souborů a kompletní webové stránky. Na těchto scénářích bude student dokazovat, či vyvracet nebo doplňovat teoretické poznatky o výše zmíněných protokolech.

6.2 Příprava a seznámení s laboratorním prostředím

1. Spusťte virtuální počítače s názvy „LinuxMint Client“ a „LinuxMint Server“.

- Na počítači „Client“ se provádějí měření a slouží k ovládání Serveru přes kontrolní skript, který na pozadí využívá automatizační nástroj Ansible pro jeho nastavení.
- Server slouží jako zdroj služeb a také zde aplikujeme negativní vlivy ovlivňující přenosovou linku.
- Pro přihlášení používejte jméno **student** a heslo **student**. Heslo k účtu **root** je **vutbrno**.

2. Otevřete program Wireshark a seznamte se s jeho ovládáním.
   - Veškerá měření provádějte na rozhraní „enp0s8“.
   - Měřená data jako počet zachycených paketů, doba přenosu a počet zachycených dat najdeme v nabídce „Statistics -> Capture File Properties“. Pokud se vám tato možnost nezobrazuje, označte libovolný zachycený paket a dvakrát zmáčkněte zkratku **Ctrl + D**.
3. Ovládání serveru a výběr úlohy.
   - Otevřete terminál.
   - Výběr úlohy se provádí příkazem „measurement_cli“.
     - Přepínač „–protocol“ slouží k výběru měřeného protokolu.
     - Přepínač „–encryption“ slouží k výběru šifrované verze protokolu HTTP, tedy HTTPS.
     - A pro u přepínače „–server“ v této úloze vždy vybíráme možnost **caddy**.
   - Po zadaní příkazu například „measurement_cli –server caddy – protocol quic“ se spustí Ansible a nastaví server, poskytoval obsah na serveru Caddy a byl dostupný přes protokol QUIC.
4. Nastavení negativních vlivů na lince.
   - Spusťte terminál a pomocí příkazu „su“ se přihlaste jako root.
   - Při prvním nastavení negativních vlivů se požívá funkce **add** programu NetEM. Například nastavení zpoždění „tc qdisc **add** dev enp0s8 root netem delay 100ms“.
   - Každá další změna se provádí pomocí funkce **change**. Například změna na ztrátovost 0,5% „tc qdisc **change** dev enp0s8 root netem loss 0.5%“
   - Ovládání zpoždění se provádí pomocí
     „tc qdisc add dev enp0s8 root netem delay **$ČAS$**ms“
   - Ovládání jitter - následující příkaz nastaví zpoždění linky na 1 ms a jitter na zvolenou hodnotu $ČAS$
     „tc qdisc change dev enp0s8 root netem delay 1ms **$ČAS$**ms 25%“
   - Ovládání ztrátovosti
     „tc qdisc change dev enp0s8 root netem loss **$ZTRÁTOVOST$**%“
5. Přes začátkem měření se ujistěte, že je vypnuta podpora následujících funkcí

zadáním níže uvedených příkazů do terminálu:

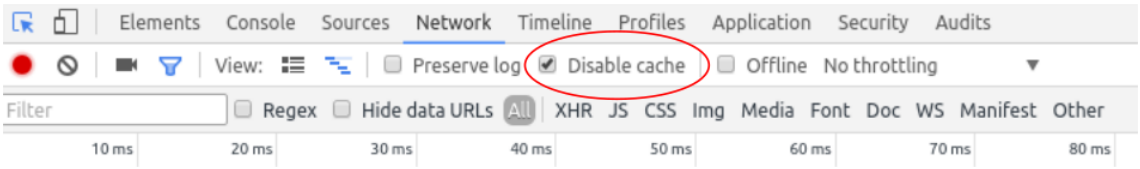
- Generic Segment Offload - příkazem „**sudo ethtool –offload enp0s8 gso off**“
- Generic Reassembly Offload - příkazem „**sudo ethtool –offload enp0s8 gro off**“
- TCP Segment Offload - příkazem „**sudo ethtool –offload enp0s8 tso off**“
- Tyto funkce zkreslují výsledky protokolů využívajících transportní protokol TCP. Některé funkce protokolu TCP jsou totiž přesunuty až do ovladače síťové karty, proto je program Netem ani Wireshark nevidí. Bez vypnutí těchto funkcí síťové karty bychom viděli pakety o velikosti větší, než 1500 bytů, což je běžně hodnota MTU (Maximum Transmission Unit) nastavená na síťové kartě.

6. **Všechna měření provádějte alespoň 5x a do tabulky vkládejte aritmetický průměr výsledků.**

6.3 Postup měření

6.3.1 Protokol HTTP verze 1.1

1. Vyberte úlohu příkazem „measurement_cli –server caddy – protocol http“
2. Otevřete programy Wireshark a Chromium Web Browser z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network** viz obrázek 5.1.



Obr. 6.1: Zakázané ukládání dat do cache - webový prohlížeč Chromium

4. Měření na přenosu jednoho velkého souboru.
   (a) Zapněte odchytávání paketů v programu Wireshark.
   (b) V programu Chromium otevřete stránku „http://transport.protocols.org/one_big“.
   (c) Během přenosu sledujte chování přenosového protokolu.
   (d) Po skončení načítání stránky vypněte zachytávání paketů.

(e) Odečtěte počet přenesených paketů, dobu přenosu a velikost přenesených dat.

(f) Postupně nastavujte zpoždění na hodnoty 100, 150 a 200ms, kolísavé zpoždění (jitter) 10ms, 50ms a 150ms, ztrátovost v hodnotách 0,5%, 2,5% a 10%. Po každé změně negativních vlivů opakujte body (a) až (e).

5. Měření na přenosu velkého množství malých souborů.
   (a) V programu Chromium otevřete stránku „http://transport.protocols.org/small_objects“.
   (b) Postup měření je stejný jako v bodě 4.
6. Přenos kompletní webové stránky.
   (a) V programu Chromium otevřete stránku „http://transport.protocols.org/full_page“.
   (b) Postup měření je stejný jako v bodě 4.
7. Naměřené údaje graficky zpravujte a porovnejte s ostatními protokoly.

6.3.2 Protokol HTTPS verze 1.1

1. Vyberte úlohu příkazem „measurement_cli –server caddy – protocol http – encryption“
2. Otevřete programy Wireshark a Chromium Web Browser z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network**.
4. Používejte adresu „https://transport.protocols.org/“.
5. Postup měření je stejný jako v sekci 6.3.1 v bodech 4 až 6.
6. Naměřené údaje graficky zpravujte a porovnejte s ostatními protokoly.

6.3.3 Protokol HTTP2

1. Vyberte úlohu příkazem „measurement_cli –server caddy – protocol http2“
2. Otevřete programy Wireshark a Chromium Web Browser z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network**.
4. Používejte adresu „https://transport.protocols.org/“.
5. Postup měření je stejný jako v sekci 6.3.1 v bodech 4 až 6.
6. Naměřené údaje graficky zpravujte a porovnejte s ostatními protokoly.

6.3.4 Protokol QUIC

1. Vyberte úlohu příkazem „measurement_cli –server caddy – protocol quic“
2. Otevřete programy Wireshark a Chromium Quic z hlavní plochy.
3. V programu Chromium se ujistěte, že máte zablokované ukládání obrázků do cache.
   - Zkratka **Ctrl + Shift + C** a výběr možnosti **Disable Cache** v záložce **Network**.
4. Používejte adresu „https://transport.protocols.org/“.
5. Postup měření je stejný jako v sekci 6.3.1 v bodech 4 až 6.
6. Naměřené údaje graficky zpravujte a porovnejte s ostatními protokoly.

6.3.5 Doplňující otázky

1. Zhodnoťte zda se vlastnosti transportního protokolu TCP implementované na úrovni aplikačního protokolu QUIC vyrovnají vlastnostem protokolu TCP na transportní úrovni.
2. Jaký z měřených protokolů využívá vynucený přenos dat? Zdůvodněte.
3. Jaký je váš názor na budoucnost protokolů pro přenos webových stránek, jaký protokol z dvojice HTTP2 a QUIC podle vás bude používanější a proč?
4. Na přenosu velkého počtu souborů otestujte jaký vliv na parametry přenosu mají následující kombinace negativních vlivů:
   - Ztrátovost paketů 2,5% a zpoždění 100ms.
     „tc qdisc change dev enp0s8 root netem loss 2.5% delay 100ms“
   - Ztrátovost paketů 2,5% a jitter 50ms.
     „tc qdisc change dev enp0s8 root netem loss 2.5% delay 1ms 10ms 25%“

6.4 Vzorové vypracování

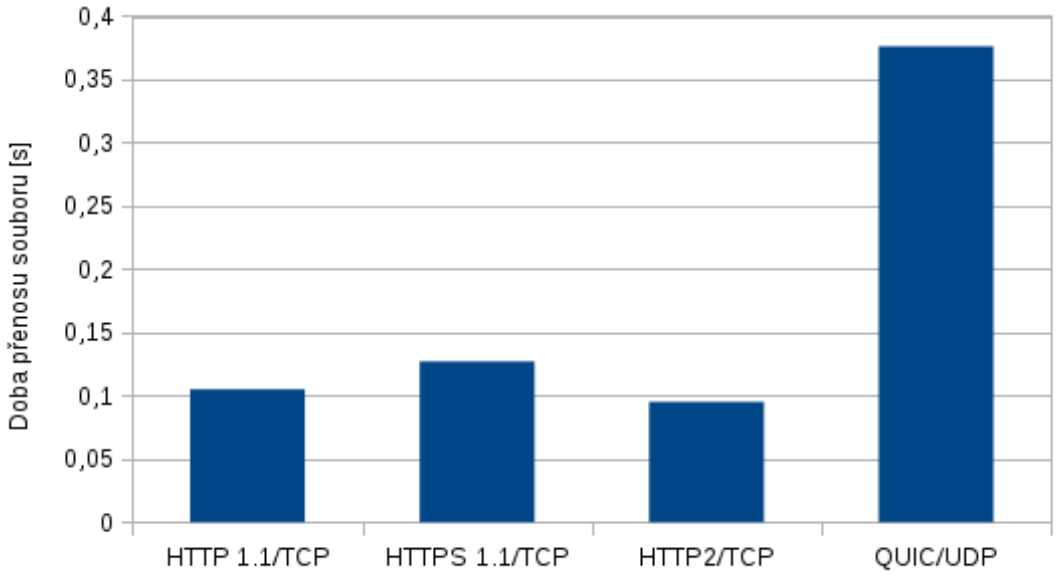
6.4.1 Přenos velkého souboru

Ideální průběh Protokoly rodiny HTTP v tomto testu byly více než třikrát rychlejší než protokol QUIC a k přenosu potřebovaly o téměř 50% paketů méně, jak můžeme vidět na obrázku 6.2, přesto se rozdíl přenesených dat pohyboval pouze okolo 3 procent. To ukazuje, že protokol QUIC k přenosu dat využívá velkého počtu malých paketů a díky tomu vyžaduje přenos vyšší režii.

Zpoždění Z grafu na obrázku 6.3 je patrné, že od hodnoty zpoždění narůstá u všech protokolů doba přenosu lineární, avšak u protokolu QUIC je strmost nižší v

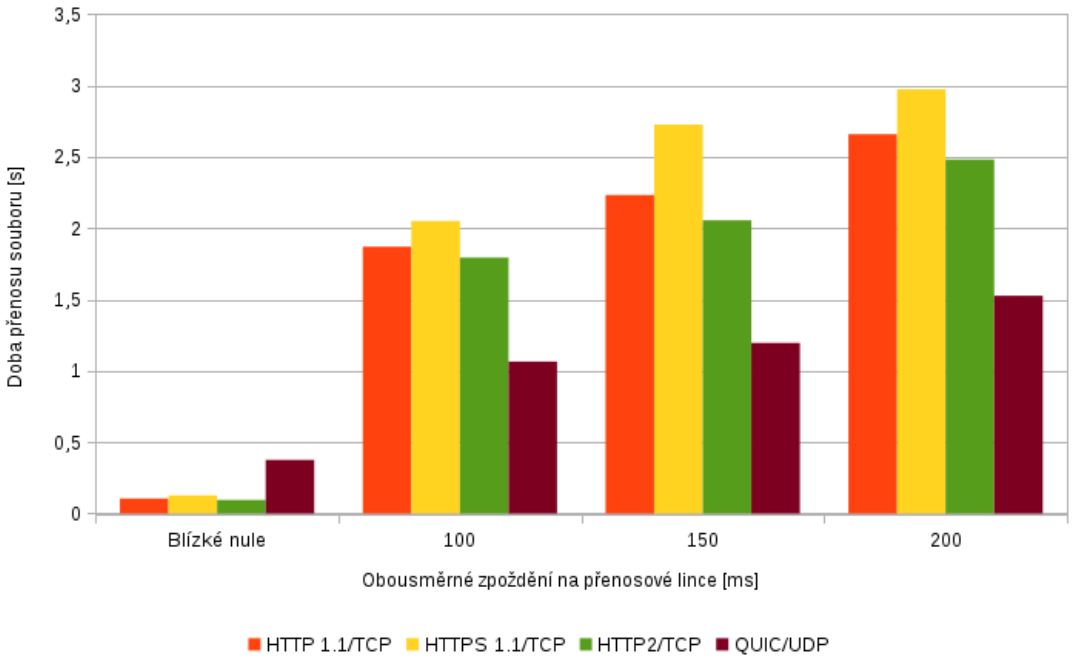
Protokol	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	862	0,105	1155570
HTTPS 1.1/TCP	872	0,127	1158202
HTTP2/TCP	894	0,095	1162488
QUIC/UDP	1217	0,376	1186503

Tab. 6.1: Naměřené hodnoty na webovém serveru Caddy při přenosu jednoho souboru - bez uměle zhoršených přenosových podmínek (průměr pěti měření)



Obr. 6.2: Graf doby přenosu protokolů na webovém serveru Caddy při ideálním stavu linky (průměr pěti měření)

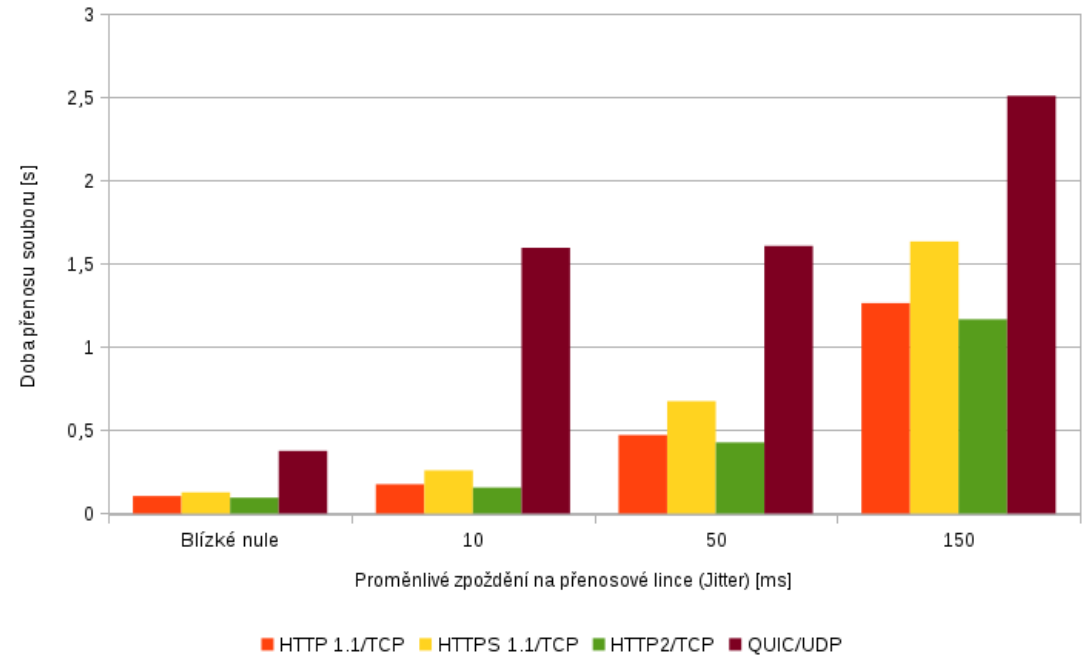
porovnání s ostatními protokoly. Tento protokol si také vedl v testu nejlépe, následovaný protokolem HTTP2. Z testu je patrné, že režie přidaná šifrováním dat u protokolu HTTPS oproti protokolu HTTP má znatelný vliv na prodloužení doby přenosu při zvýšeném zpoždění na lince.



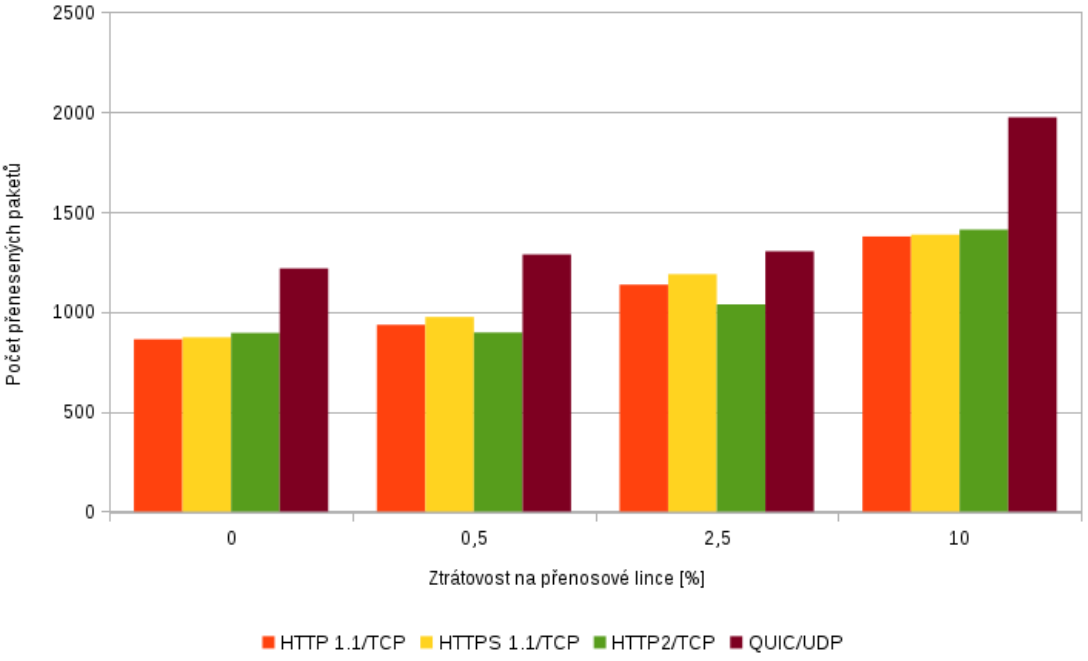
Obr. 6.3: Graf doby přenosu protokolů na webovém serveru Caddy při zpoždění (půměr pěti měření)

Jitter V tomto testu vyšel protokol QUIC jako nejhorší ve zvládání proměnlivého zpoždění (jitter), viz 6.4. Jitter způsobuje především špatnou posloupnost přenášených dat, což mají ostatní protokoly ošetřeno již na transportní vrstvě díky protokolu TCP. Protokol QUIC však toto řeší až na vrstvě aplikační, jak bylo popsané v kapitole 2.

Ztrátovost Při přenosu velkého souboru prokazatelně nedošlo k dopředné opravě chyb u protokolu. Z grafu na obrázku 6.5 je ale patrné, že počet přenesených paketů protokolem QUIC je téměř konstantní v rozmezí ztrátovosti 0,5% a 2,5%, zatímco u ostatních protokolů dochází k lineárnímu nárůstu. U všech měřených protokolů dochází k nárůstu přenosové doby se zvyšující se ztrátovostí, nejkratší dobu přenosu zaznamenal protokol HTTP2 pro všechny měřené hodnoty ztrátovosti paketů, avšak nejnižší nárůst přenosové doby se projevil u protokolu QUIC, viz tabulka B.3.



Obr. 6.4: Graf doby přenosu protokolů na webovém serveru Caddy při proměnlivém zpoždění (průměr pěti měření)



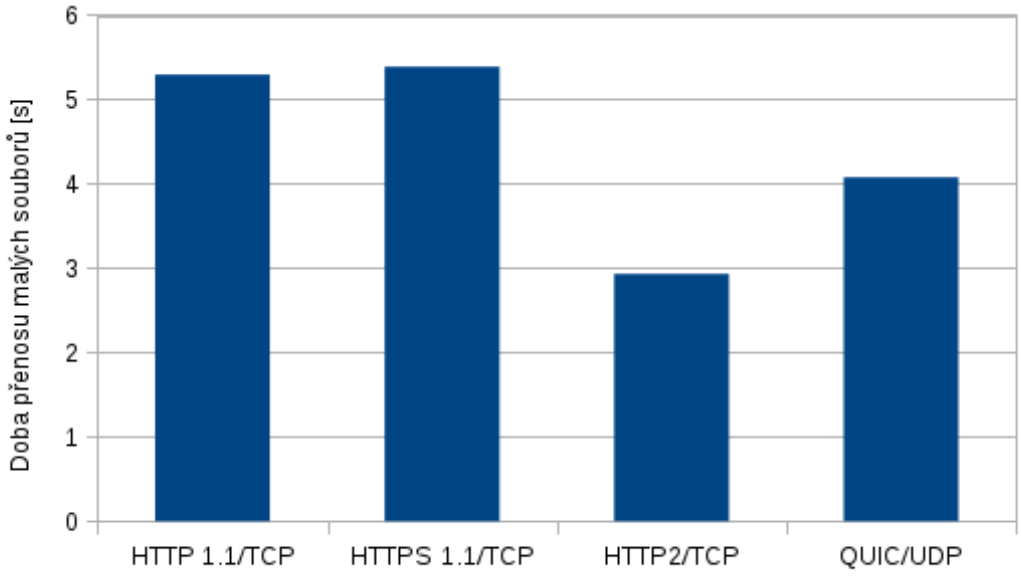
Obr. 6.5: Graf počet přenesených paketů protokolů na webovém serveru Caddy při ztrátovosti (průměr pěti měření)

6.4.2 Velký počet malých souborů

Ideální průběh Protokol HTTP2 byl při přenosu velkého počtu souborů nejrychlejší z měřených protokolů, celý přenos tímto protokolem trval v průměru 2,9 vteřiny. Druhým nejrychlejším protokolem byl QUIC s časem přenosu 4 sekundy, ten ale potřeboval k přenosu o více než 140% počet paketů než protokol HTTP2. Jak můžeme vidět v grafu na obrázku 6.6, protokoly HTTP a HTTPS verze 1.1 měly téměř totožnou dobu přenosu, lišící se pouze o 93ms.

Protokol	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1488	5,293	1010889
HTTPS 1.1/TCP	1618	5,386	1036074
HTTP2/TCP	977	2,931	789606
QUIC/UDP	2206	4,077	893198

Tab. 6.2: Naměřené hodnoty na webovém serveru Caddy při přenosu velkého počtu souborů - bez uměle zhoršených přenosových podmínek (půměr pěti měření)



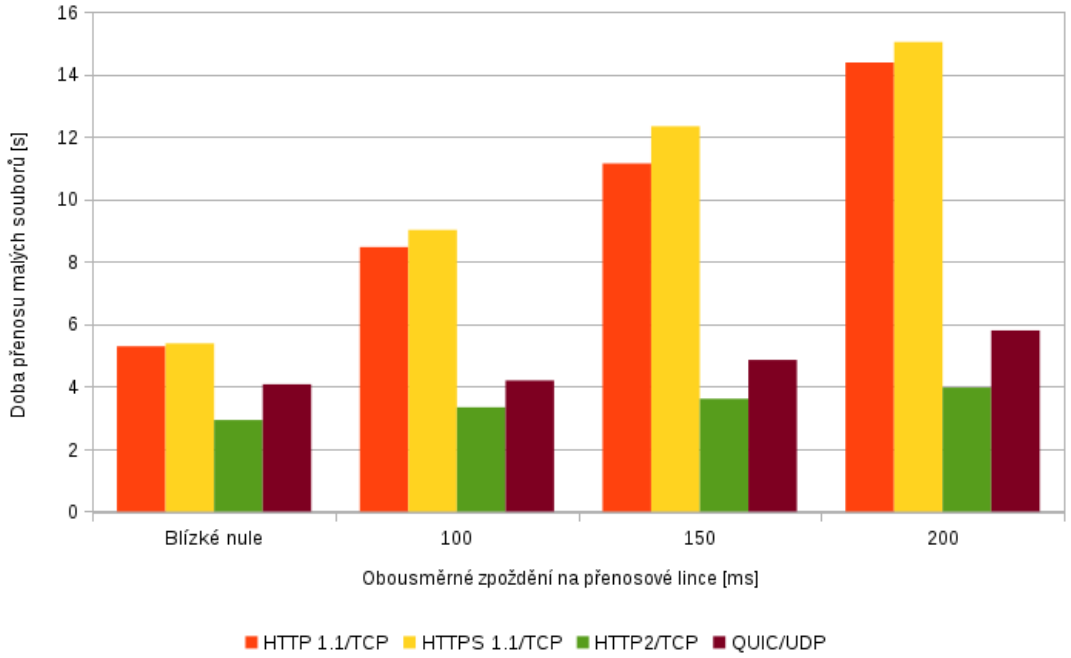
Obr. 6.6: Graf doby přenosu (velkého počtu souborů) protokolů na webovém serveru Caddy při ideálním stavu linky (půměr pěti měření)

Zpoždění V tomto měření se opět projevil pokles přenášených paketů, obdobně jako v měření na webovém serveru Apache. Z grafu na obrázku 6.8 je patrné, že u

všech protokolů se narůstající zpoždění projevilo lineárním nárůstem doby přenosu. Nejvíce se tento vliv projevil u protokolů HTTP 1.1 a HTTPS 1.1, naopak nejmenší nárůst zaznamenal protokol HTTP2.

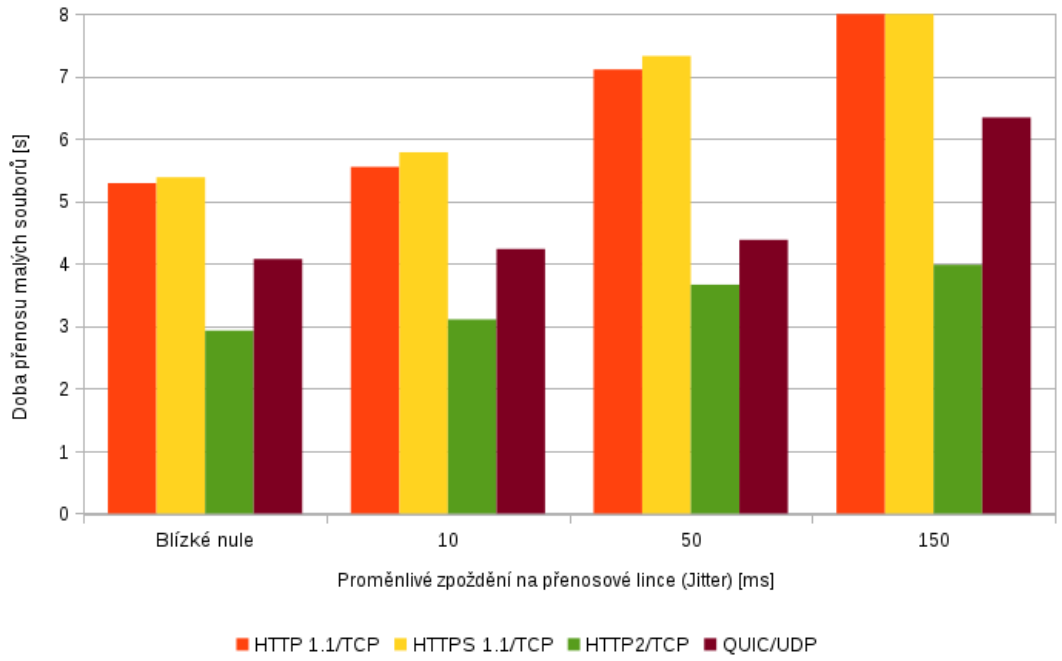
1	0.000000000	192.168.150.2	192.168.150.1	QUIC	111 Payload (Encrypted), CID: 9082959406529914236, Seq: 191
2	0.010879390	192.168.150.1	192.168.150.2	QUIC	1392 Payload (Encrypted), Seq: 8429
3	0.010904172	192.168.150.1	192.168.150.2	QUIC	1392 Payload (Encrypted), Seq: 8428
4	0.010906984	192.168.150.1	192.168.150.2	QUIC	211 Payload (Encrypted), Seq: 8427
5	0.010910283	192.168.150.1	192.168.150.2	QUIC	1392 Payload (Encrypted), Seq: 8430
6	0.011553286	192.168.150.2	192.168.150.1	QUIC	84 Payload (Encrypted), CID: 9082959406529914236, Seq: 192
7	0.011664399	192.168.150.2	192.168.150.1	QUIC	81 Payload (Encrypted), CID: 9082959406529914236, Seq: 193
8	0.011929834	192.168.150.2	192.168.150.1	QUIC	78 Payload (Encrypted), CID: 9082959406529914236, Seq: 194
9	0.022385865	192.168.150.1	192.168.150.2	QUIC	1392 Payload (Encrypted), Seq: 8431
10	0.022409162	192.168.150.1	192.168.150.2	QUIC	1392 Payload (Encrypted), Seq: 8432
11	0.022617717	192.168.150.2	192.168.150.1	QUIC	81 Payload (Encrypted), CID: 9082959406529914236, Seq: 195

Obr. 6.7: Označení proudu (CID) protokolem QUIC



Obr. 6.8: Graf doby přenosu (velkého počtu souborů) protokolů na webovém serveru Caddy při zpoždění (průměr pěti měření)

Proměnlivé zpoždění Vliv jitter se na protokolech HTTP 1.1 a HTTPS 1.1 projevil podobně a na hodnotě proměnlivého zpoždění 150ms dokonce dosahovaly totožné přenosové doby, viz obrázek 6.9. Dá se tedy předpokládat, že šifrování nemá vliv na parametry přenosu při jitter na přenosové lince. Protokoly QUIC se jevil jako stabilní do hodnoty jitter 150ms, kde došlo k rapidnímu nárůstu přenosové doby. Protokol HTTP2 oproti tomu vykazoval pouze mírný nárůst doby přenosu se zvyšující se hodnotou proměnlivého zpoždění.

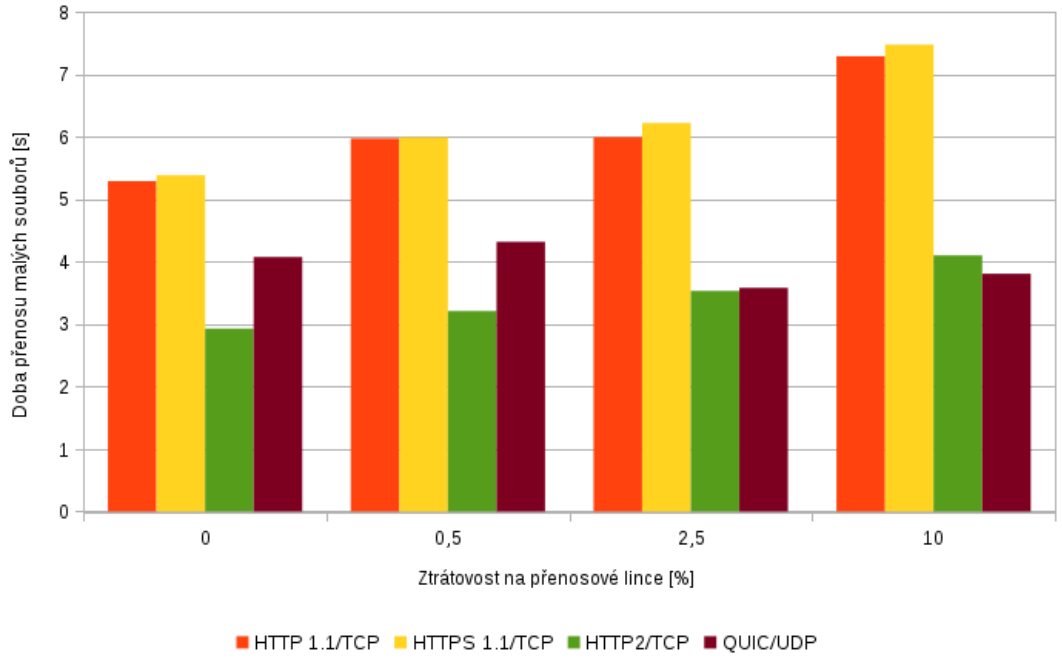


Obr. 6.9: Graf doby přenosu (velkého počtu souborů) protokolů na webovém serveru Caddy při proměnlivém zpoždění (průměr pěti měření)

Ztrátovost Vliv zvyšující se ztrátovosti na lince vedl ke zvyšování přenosové doby u všech protokolů z rodiny HTTP. Avšak u protokolu QUIC došlo k poklesu doby potřebné pro přenos při změně ztrátovosti z 0,5% na hodnotu 2,5% a to až pod úroveň naměřenou při ideálním stavu linky. Ani při 10% ztrátovosti nepřesáhla doba přenosu hodnotu při měření na ideální lince, jak můžeme vidět na obrázku 6.10. To jednoznačně prokazuje schopnost protokolu QUIC vyrovnat se se ztrátovostí dopřednou opravou chyb, popisovanou v kapitole 2.

6.4.3 Kompletní webová stránka

Ideální průběh Při ideálních podmínkách dopadly v tomto scénáři všechny protokoly velice podobně, až na protokol HTTPS 1.1, který dosáhl průměrné doby přenosu 268ms, což je téměř o 20% více než protokol HTTP stejné verze, viz 6.11. Protokoly HTTP2 a QUIC se téměř nelišily ani velikostí přenesených dat, kde rozdíl činil v průměru méně než 1%.

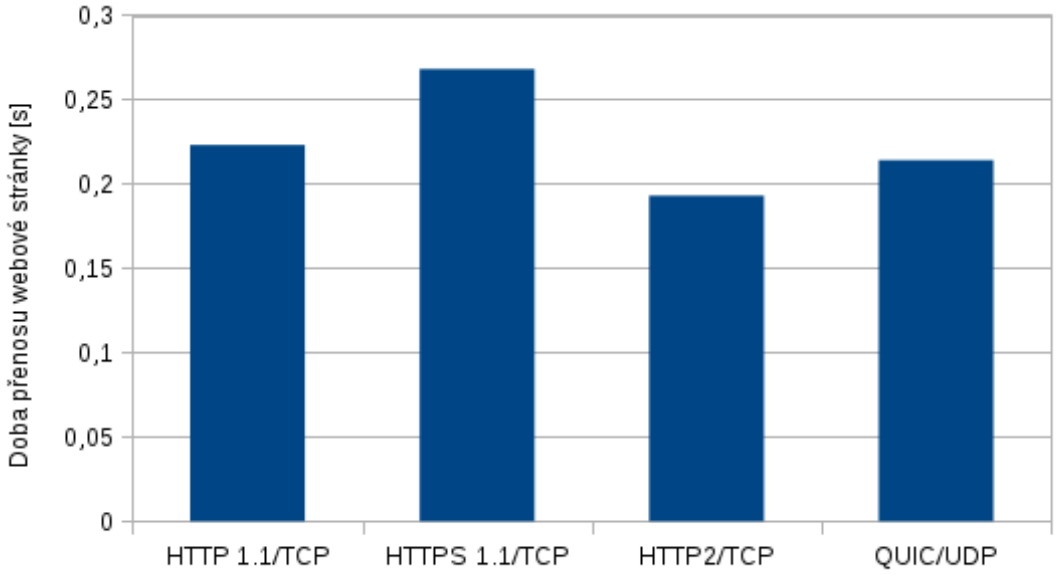


Obr. 6.10: Graf doby přenosu (velkého počtu souborů) protokolů na webovém serveru Caddy při ztrátovosti (průměr pěti měření)

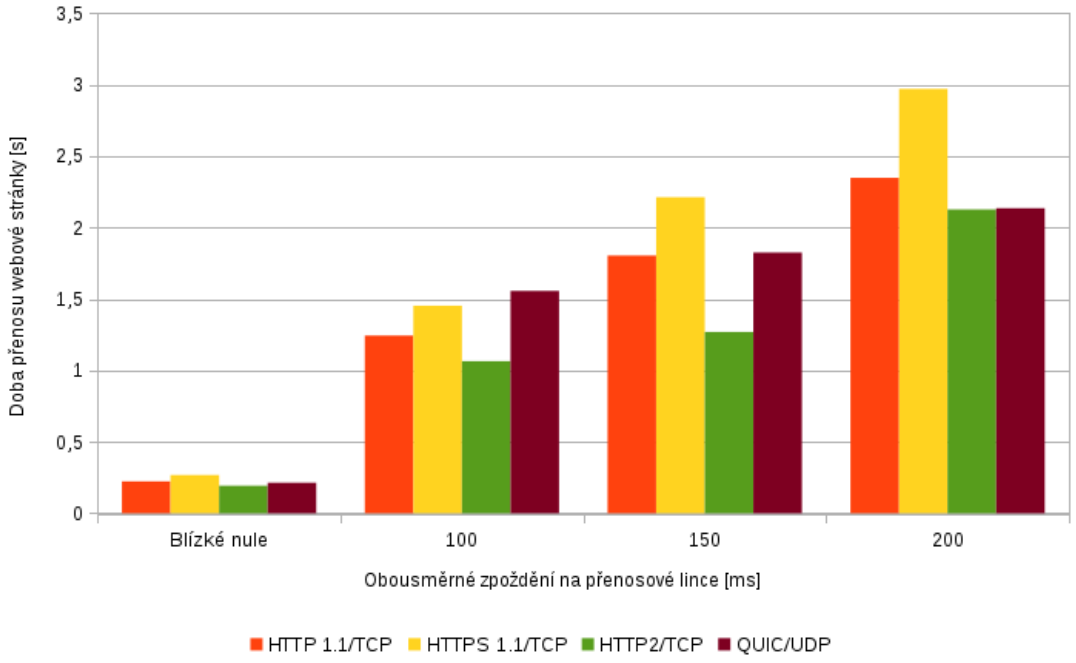
Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTP 1.1/TCP	177	0,223	185318
HTTPS 1.1/TCP	191	0,268	187513
HTTP2/TCP	210	0,193	183322
QUIC/UDP	241	0,214	183470

Tab. 6.3: Naměřené hodnoty na webovém serveru Caddy při přenosu webové stránky - bez uměle zhoršených přenosových podmínek (průměr pěti měření)

Zpoždění Protokoly HTTP a HTTPS se v tomto měření nechovaly stejně jako tomu bylo u předchozích měření vlivu zvyšujícího se zpoždění. V předchozích případech docházelo ke snižování počtu přenášených paketů oproti ideální lince, zde ale dochází k opačnému jevu. U protokolů QUIC a HTTP2 se sice trend zachoval, ale v tak malé míře, že může jít pouze o chybu měření z důvodu krátké měření a malého objemu přenášených dat. Zajímavé také je, že ačkoli si protokol vedl nejlépe při hodnotách zpoždění 100 a 150ms, při hodnotě zpoždění 200ms byla jeho doba přenosu téměř totožná s dobou přenosu u protokolu QUIC, jak můžeme pozorovat v grafu na obrázku 6.12.



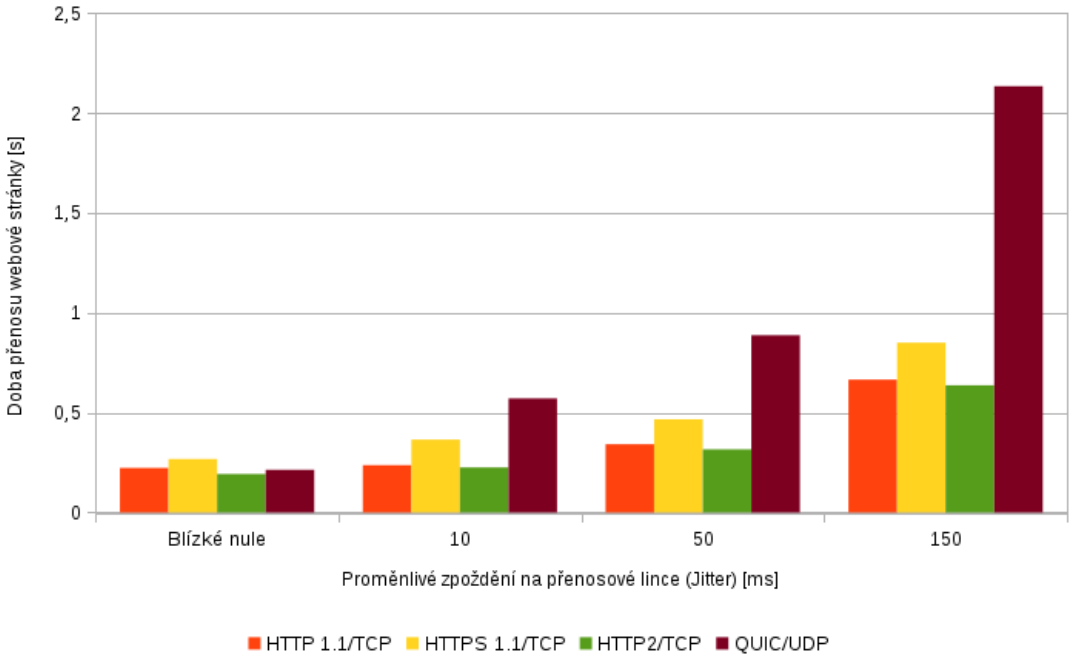
Obr. 6.11: Graf doby přenosu (kompletní webová stránka) protokolů na webovém serveru Caddy při ideálním stavu linky (půměr pěti měření)



Obr. 6.12: Graf doby přenosu (kompletní webová stránka) protokolů na webovém serveru Caddy při zpoždění (půměr pěti měření)

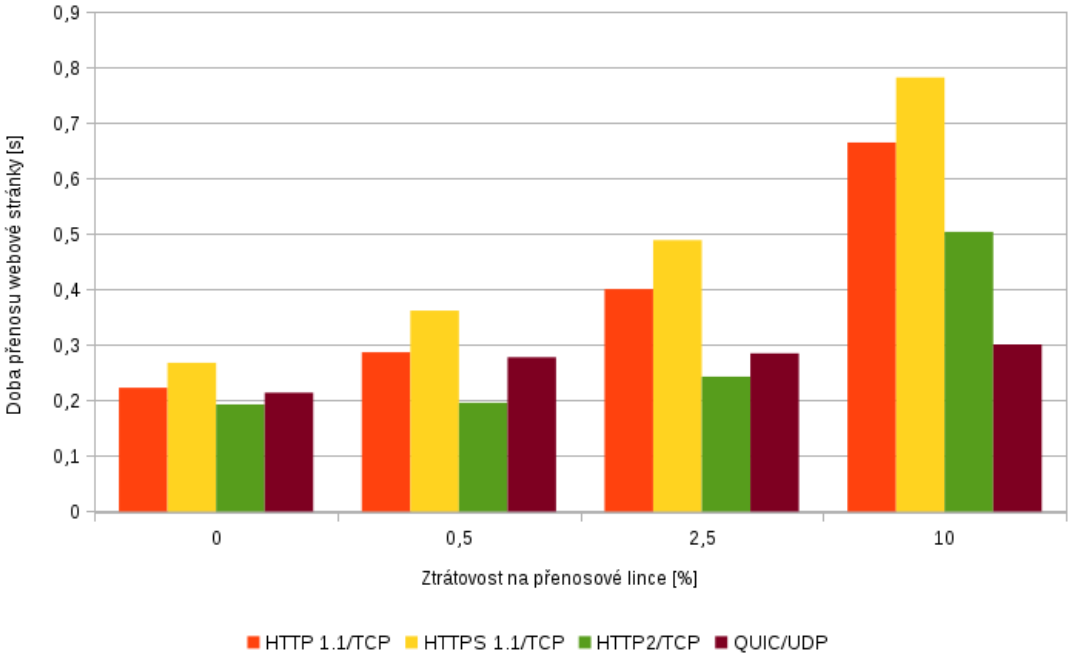
Jitter Z grafu na obrázku 6.13 je patrné, že proměnlivé zpoždění mělo při tomto měření téměř totožný vliv na protokoly HTTP a HTTPS. Nejhůře se s tímto ne-

gativním vlivem vypořádal protokol QUIC, u kterého se zvyšující se jitter způsobil strmý nárůst přenosové doby na téměř desetinásobek přenosové doby při ideálních podmínkách.



Obr. 6.13: Graf doby přenosu (kompletní webová stránka) protokolů na webovém serveru Caddy při proměnlivém zpoždění (průměr pěti měření)

Ztrátovost U protokolu QUIC se při tomto měření projevila zvyšující se ztrátovost nárůstem přenosové doby pouze přibližně 8% mezi hodnotami ztrátovosti od 0,5% do 10%. To opět dokazuje vlastnost protokolu dopředné opravy chyb, jako v předchozím scénáři. U ostatních měřených protokolů měla ztrátovost za příčinu nárůst přenosové doby, avšak u protokolu HTTP2 byl tento nárůst mnohem nižší než u protokolů HTTP a HTTPS, jak můžeme pozorovat na obrázku 6.14.



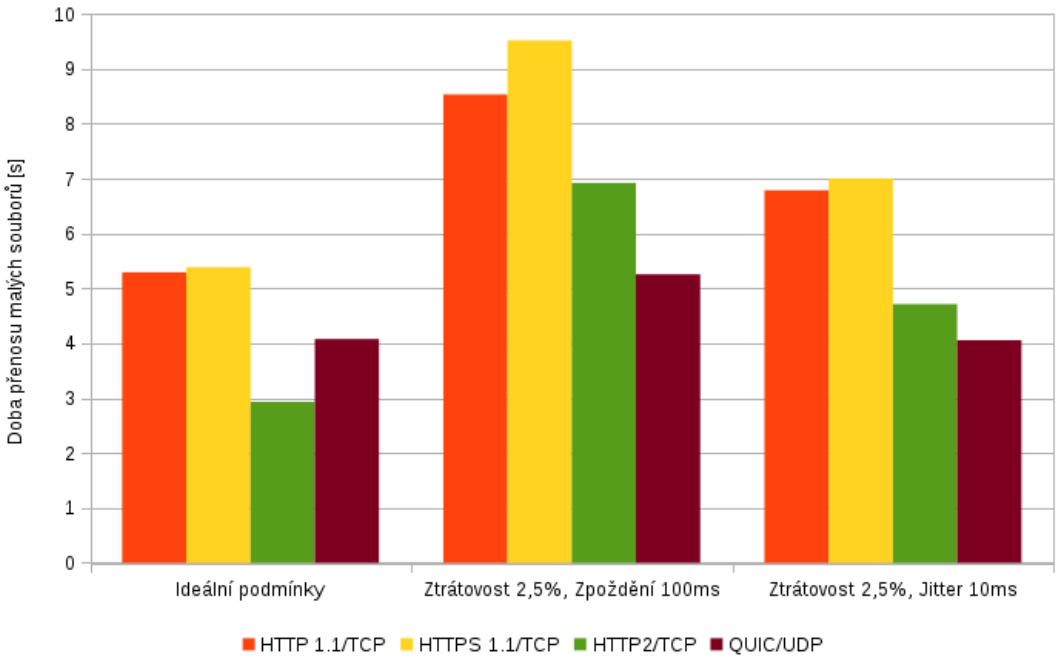
Obr. 6.14: Graf doby přenosu (kompletní webová stránka) protokolů na webovém serveru Caddy při ztrátovosti (průměr pěti měření)

6.4.4 Doplňující otázky

1. Zhodnoťte zda se vlastnosti transportního protokolu TCP implementované na úrovni aplikačního protokolu QUIC vyrovnají vlastnostem protokolu TCP na transportní úrovni.
    - Nevyrovnají, protože z měření jasně vyplývá, že protokol QUIC ztrácí na protokoly používající transportní protokol TCP především při proměnlivém zpoždění. Jitter má vliv především na pořadí doručených paketů, což protokol TCP ošetřuje. Z měření ale také vyplývá, že protokol QUIC má implementovaný mechanismus plovoucího okna, protože na zpoždění reaguje jeho úpravou a tím snížením počtu přenášených paketů.
2. Jaký z měřených protokolů využívá vynucený přenos dat? Zdůvodněte.
    - Protokoly HTTP2 a QUIC, tento mechanismus můžeme pozorovat při přenosu velkého počtu souborů, kde je protokol HTTP2 mnohem rychlejší než ostatní protokoly. U protokolu QUIC je tato vlastnost nejlépe vidět až při zpoždění na lince, kde se tento negativní vliv projevuje pouze mírným nárůstem doby přenosu, viz obrázek 6.8.
3. Jaký je váš názor na budoucnost protokolů pro přenos webových stránek, jaký protokol z dvojice HTTP2 a QUIC podle vás bude používanější a proč?
    - Z provedených měření se zdá být protokol HTTP2 rychlejší a vhodnější

ve všech situacích než protokol QUIC. Je ale třeba podotknout, že protokol QUIC byl testovaný v experimentální implementaci, a tak by mohl vykazovat lepší vlastnosti v budoucnu, kdy jeho implementace bude finální.

4. Na přenosu velkého počtu souborů otestujte, jaký vliv na parametry přenosu mají kombinace negativních vlivů.
   - Kombinace ztrátovosti a zpoždění se projevila u protokolů HTTP 1.1 a HTTPS 1.1 jako součet dílčích negativních vlivů. U protokolu HTTP2, obdobně jako v měření na webovém serveru Apache, došlo k markantnímu nárůstu přenosové doby v porovnání s dílčími jevy. V grafu na obrázku 6.15 je vidět, že protokol QUIC se s kombinací těchto vlivů vypořádal nejlépe. Došlo u něho sice ke stejnému jevu jako u protokolu HTTP2, kdy se nejedná pouze o součet dílčí negativních vlivů, ale nárůst přenosové doby nebyl tak vysoký.
   - Při kombinaci jitter a ztrátovosti dokonce dosahoval protokol QUIC podobných hodnot jako při ideálních podmínkách, což zapříčinila jeho vlastnost dopředné opravy chyb popisovaná v kapitole 2.



Obr. 6.15: Graf doby přenosu (velkého počtu souborů) protokolů na webovém serveru Caddy při kombinaci negativních vlivů (průměr pěti měření)

7 ZÁVĚR

Cílem této práce bylo srovnání různých protokolů přenosu webových stránek a možnosti jejich praktické otestování při přenosu různých typů souborů a při různých podmínkách na přenosové lince. Protože je implementace některých protokolů pouze experimentální, volil jsem dva různé webové servery, abych obsáhl co nejširší spektrum jejich variací. Také jsem zjistil, že ne všechny protokoly je možné otestovat na všech přenosových protokolech, protože některé z nich jsou návrhem vázány na jeden konkrétní transportní protokol a nebo taková implementace neexistuje na straně webového serveru. Právě díky experimentální implementaci nebylo možné některá měření dokončit, především u protokolu SCTP, kde často stávalo, že webový prohlížeč spadl z důsledku špatného přístupu do paměti počítače. Díky měření stejným postupem na dvou různých webových serverech, můžeme odstranit vliv webového serveru na parametry přenosu a to nám umožňuje bezpečně určit chování protokolu při různých scénářích. U všech společných protokolů docházelo ke konstantnímu vlivu webového serveru, protože poměrový vliv negativních podmínek byl stejný. I přesto, že jsem měřil protokol QUIC v experimentální verzi, byl jsem schopný prokázat jeho vlastnosti uvedené v teorii. Pravě srovnání protokolů HTTP2 a QUIC je velkou hodnotou pro studenty, pro které jsou určeny navržené úlohy. Tyto dva protokoly totiž představují nástupce dnes nejvíce využívaných protokolů.

LITERATURA

[1] AULDS, Charles. *Linux - administrace serveru Apache. Praha: Grada, 2003. Profesional. ISBN 80-247-0640-7.*

[2] JEŘÁBEK, J. *Pokročilé komunikační techniky. Skriptum FEKT Vysoké učení technické v Brně, 2016. s. 1-193.*

[3] SOSINSKY, Barrie A. *Mistrovství - počítačové sítě: [vše, co potřebujete vědět o správě sítí]. Brno: Computer Press, 2010. ISBN 978-80-251-3363-7.*

[4] Akamain *Turn-on HTTP/2 today!* [online]. Poslední aktualizace v roce 2016, [cit. 14. 05. 2017]. Dostupné z URL: <`https://http2.akamai.com/`>.

[5] AUGUSTIN, S. *Netlab-HBRS Docker repository* [online]. Poslední aktualizace v roce 2015, [cit. 14. 05. 2017]. Dostupné z URL: <`https://hub.docker.com/u/netlab/`>.

[6] BERNERS-LEE, T. - FIELDING, R. - FRYSTYK, H. - MIT/LCS - UC Irvine *Hypertext Transfer Protocol – HTTP/1.0* [online]. 1996, [cit. 26. 11. 2016]. Dostupné z URL: <`https://tools.ietf.org/html/rfc1945`>.

[7] BELSHE, M. - BitGo - Google, Inc - Mozilla - PEON, R. - THOMSON, Ed. *RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)* [online]. 2015, [cit. 02. 12. 2016]. Dostupné z URL: <`https://tools.ietf.org/html/rfc7540`>.

[8] BELSHE, M. - PEON, R. *SPDY Protocol - Draft 1* [online]. [cit. 02. 12. 2016]. Dostupné z URL: <`https://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1`>.

[9] Google, Inc *QUIC, a multiplexed stream transport over UDP* [online]. [cit. 04. 12. 2016]. Dostupné z URL: <`https://www.chromium.org/quic`>.

[10] HOLT, M. *Caddy - README* [online]. Poslední aktualizace 28. 10. 2016, [cit. 02. 12. 2016]. Dostupné z URL: <`https://github.com/mholt/caddy`>.

[11] LUDOVICI, F. - PFEIFER, Hagen P. *NetEm - Network Emulator* [online]. 2011, [cit. 26. 11. 2016]. Dostupné z URL: <`http://man7.org/linux/man-pages/man8/tc-netem.8.html`>.

[12] POSTEL, J. *RFC 768 - User Datagram Protocol* [online]. 1980, [cit. 26. 11. 2016]. Dostupné z URL: <`https://tools.ietf.org/html/rfc768`>.

[13] RESCORLA, E. - RTFM, Inc. *HTTP Over TLS* [online]. 2000, [cit. 26. 11. 2016]. Dostupné z URL: <https://tools.ietf.org/html/rfc2818>.

[14] STENBERG, D. *Http2 explained, online e-book* [online]. Poslední aktualizace 4. 5. 2016, [cit. 02. 12. 2016]. Dostupné z URL: <https://daniel.haxx.se/http2/>.

[15] STEWART, R. *RFC 4960 - Stream Control Transmission Protocol* [online]. 2007, [cit. 26. 11. 2016]. Dostupné z URL: <https://tools.ietf.org/html/rfc4960>.

[16] USC Information Sciences Institute *RFC 793 - Transmission Control Protocol* [online]. 1981, [cit. 26. 11. 2016]. Dostupné z URL: <https://tools.ietf.org/html/rfc793>.

[17] w3techs *Usage of HTTP/2 for websites* [online]. Poslední aktualizace 1. 12. 2016, [cit. 02. 12. 2016]. Dostupné z URL: <https://w3techs.com/technologies/details/ce-http2/all/all>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CSS	Cascading Style Sheets
FEC	Forward Error Correction
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
MTU	Maximum Transmission Unit
OSI	Open Systems Interconnection
QUIC	Quick UDP Internet Connections
RFC	Request for Comments
SCTP	Stream Control Transmission Protocol
SIGTRAN	Signaling Transport
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SEZNAM PŘÍLOH

A TABULKY NAMĚŘENÝCH HODNOT - SROVNÁNÍ PROTOKOLŮ SPDY, HTTP2 A HTTP

A.1 Přenos jednoho souboru

Protokol	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	911	1,279	1158835	987	1,856	1163851	1034	2,509	1166953
HTTPS 1.1/TCP	968	1,661	1166106	1061	2,540	1172244	1083	3,114	1173696
HTTP2/TCP	891	0,855	1181589	888	1,245	1158684	913	1,743	1160307
SPDY/TCP	906	0,851	1159318	907	1,246	1159384	906	1,660	1159319
HTTP 1.1/SCTP	1243	2,499	1176514	1244	3,208	1176440			

Tab. A.1: Naměřené hodnoty při přenosu jednoho souboru (webový server Apache) - vliv zpoždění

Protokol	Jitter 10ms			Jitter 50ms			Jitter 150ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1542	0,16	1249261	1545	0,674	1237207	1557	1,278	1233491
HTTPS 1.1/TCP	1568	0,215	1238638	1552	0,702	1227554	1599	1,526	1260704
HTTP2/TCP	1565	0,253	1245221	1542	0,571	1234024	1541	1,203	1231524
SPDY/TCP	1547	0,243	1228029	1604	0,614	1280830	1604	1,762	1280235
HTTP 1.1/SCTP	2071	0,639	1604428	2032	2,682	1579910			

Tab. A.2: Naměřené hodnoty při přenosu jednoho souboru (webový server Apache) - vliv proměnlivého zpoždění

Protokol	Ztrátovost 0,5%			Ztrátovost 2,5%			Ztrátovost 10%		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	898	0,341	1157897	1068	0,436	1171821	1359	1,165	1193122
HTTPS 1.1/TCP	1061	0,425	1159463	1295	0,453	1189896	1405	1,332	1202627
HTTP2/TCP	821	0,084	1153024	977	0,105	1164574	1352	0,865	1215675
SPDY/TCP	821	0,051	1153733	1200	0,114	1179997	1352	0,756	1192289
HTTP 1.1/SCTP	1231	0,156	1176604	1274	0,131	1178676			

Tab. A.3: Naměřené hodnoty při přenosu jednoho souboru (webový server Apache) - vliv ztrátovosti

A.2 Přenos velkého množství souborů

Protokol	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1542	0,16	1249261	1545	0,674	1237207	1557	1,278	1233491
HTTPS 1.1/TCP	1568	0,215	1238638	1552	0,702	1227554	1599	1,526	1260704
HTTP2/TCP	1565	0,253	1245221	1542	0,571	1234024	1541	1,203	1231524
SPDY/TCP	1547	0,243	1228029	1604	0,614	1280830	1604	1,762	1280235
HTTP 1.1/SCTP	2071	0,639	1604428	2032	2,682	1579910			

Tab. A.4: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Apache) - vliv zpoždění

Protokol	Jitter 10ms			Jitter 50ms			Jitter 150ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1678	5,558	1024473	1667	5,683	1020266	1562	7,942	1012220
HTTPS 1.1/TCP	2913	6,207	1202027	2945	6,356	1209597	4868	10,938	1444294
HTTP2/TCP	1726	3,508	932848	1614	3,771	893558	1390	4,313	884070
SPDY/TCP	1609	2,182	896378	1266	2,382	875440	1356	4,654	892389
HTTP 1.1/SCTP									

Tab. A.5: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Apache) - vliv proměnlivého zpoždění

Protokol	Ztrátovost 0,5%			Ztrátovost 2,5%			Ztrátovost 10%		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1630	5,7	1018777	1610	6,352	1018235	1441	8,336	1009915
HTTPS 1.1/TCP	2952	6,585	1217000	3010	7,265	1230223	4708	10,214	1455543
HTTP2/TCP	1112	3,473	788653	1161	3,776	794009	1231	3,908	800700
SPDY/TCP	1243	1,628	798790	1220	1,749	793673	1303	1,955	802844
HTTP 1.1/SCTP									

Tab. A.6: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Apache) - vliv ztrátovosti

A.3 Přenos webové stránky

Protokol	Ztrátovost2,5%, Zpoždění 100ms			Ztrátovost 2,5%, Jitter 10ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1369	9,945	999572	1663	6,041	1023548
HTTPS 1.1/TCP	2143	11,85	1135193	5065	6,896	1478667
HTTP2/TCP	1023	6,596	780243	1392	3,923	865558
SPDY/TCP	1066	6,84	782431	1576	2,348	894251

Tab. A.7: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Apache) - kombinace vlivů

Protokol	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1360	9,169	997544	1343	11,766	996422	1299	14,485	993518
HTTPS 1.1/TCP	1989	10,355	1133475	1493	12,226	1052154	1521	15,734	1053995
HTTP2/TCP	1003	3,737	770377	982	3,956	769014	1005	4,088	775810
SPDY/TCP	1189	2,066	789632	1278	2,333	795506	1302	2,562	797084
HTTP 1.1/SCTP									

Tab. A.8: Naměřené hodnoty při přenosu webové stránky (webový server Apache) - vliv zpoždění

Protokol	Jitter 10ms			Jitter 50ms			Jitter 150ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	298	0,403	188790	294	0,43	195066	296	0,831	193890
HTTPS 1.1/TCP	462	0,541	217450	493	0,626	221507	462	1,174	216609
HTTP2/TCP	258	0,289	190723	307	0,308	219577	248	0,565	180755
SPDY/TCP	300	0,176	191674	309	0,278	216445	276	0,909	194347
HTTP 1.1/SCTP	264	0,499	214294	345	0,659	248480	352	1,244	255022

Tab. A.9: Naměřené hodnoty při přenosu webové stránky (webový server Apache) - vliv proměnlivého zpoždění

Protokol	Ztrátovost 0,5%			Ztrátovost 2,5%			Ztrátovost 10%		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	220	0,255	180729	223	0,583	180927	270	0,999	181099
HTTPS 1.1/TCP	312	0,38	196431	390	0,508	205645	410	1,113	207100
HTTP2/TCP	190	0,177	171019	208	0,226	175699	226	0,461	177155
SPDY/TCP	194	0,131	172274	242	0,149	175679	248	0,314	175818
HTTP 1.1/SCTP	225	0,064	188864	226	1,609	189194			

Tab. A.10: Naměřené hodnoty při webové stránky (webový server Apache) - vliv ztrátovosti

B TABULKY NAMĚŘENÝCH HODNOT - SROVNÁNÍ PROTOKOLŮ QUIC, HTTP2 A HTTP

B.1 Přenos jednoho souboru

	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTP 1.1/TCP	866	1,87	1155738	913	2,232	1158322	917	2,658	1159200
HTTPS 1.1/TCP	905	2,05	1157652	924	2,725	1160417	931	2,973	1161537
HTTP2/TCP	879	1,794	1161437	844	2,055	1157773	840	2,481	1157532
QUIC/UDP	1275	1,064	1185468	1304	1,195	1211214	1289	1,526	1185434

Tab. B.1: Naměřené hodnoty při přenosu jednoho souboru (webový server Caddy) - vliv zpoždění

	Jitter 10ms			Jitter 50ms			Jitter 150ms		
Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTP 1.1/TCP	1532	0,176	1229770	1578	0,471	1242522	1589	1,264	1255136
HTTPS 1.1/TCP	1571	0,259	1252929	1557	0,675	1238543	1544	1,634	1228817
HTTP2/TCP	1533	0,156	1231024	1542	0,427	1232110	1532	1,167	1225553
QUIC/UDP	1880	1,596	1547884	2612	1,608	1655317	2125	2,509	1523087

Tab. B.2: Naměřené hodnoty při přenosu jednoho souboru (webový server Caddy) - vliv proměnlivého zpoždění

	Ztrátovost 0.5%			Ztrátovost 2.5%			Ztrátovost 10%		
Protokol	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**	**Počet paketů**	**Celkový čas přenosu [s]**	**Přenesená data [B]**
HTTP 1.1/TCP	935	0,228	1160356	1136	0,575	1175686	1377	1,135	1192772
HTTPS 1.1/TCP	974	0,264	1166986	1188	0,633	1184574	1386	1,298	1196198
HTTP2/TCP	896	0,143	1163809	1037	0,178	1171568	1411	0,415	1202193
QUIC/UDP	1287	0,339	1186114	1303	0,424	1188036	1973	0,592	1859746

Tab. B.3: Naměřené hodnoty při přenosu jednoho souboru (webový server Caddy) - vliv ztrátovosti

B.2 Přenos velkého množství souborů

Protokol	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1342	8,473	995973	1316	11,149	994257	1326	14,384	994917
HTTPS 1.1/TCP	1371	9,023	1025067	1364	12,342	1024605	1361	15,04	1024407
HTTP2/TCP	818	3,339	778477	728	3,613	770321	777	3,968	781422
QUIC/UDP	2182	4,196	886245	1936	4,855	870468	1765	5,799	861239

Tab. B.4: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Caddy) - vliv zpoždění

Protokol	Jitter 10ms			Jitter 50ms			Jitter 150ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1623	5,552	1031722	1634	7,112	1021626	1572	8,556	1014619
HTTPS 1.1/TCP	1665	5,787	1041500	1685	7,328	1050436	1585	8,836	1028526
HTTP2/TCP	1213	3,109	839349	1277	3,665	862739	1228	3,982	860662
QUIC/UDP	2451	4,238	1105689	1953	4,384	1177502	2017	6,343	1300019

Tab. B.5: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Caddy) - vliv proměnlivého zpoždění

Protokol	Ztrátovost 0.5%			Ztrátovost 2.5%			Ztrátovost 10%		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1600	5,975	1013465	1635	5,997	1024353	1654	7,292	1021883
HTTPS 1.1/TCP	1654	5,993	1032767	1713	6,223	1055113	1726	7,477	1049722
HTTP2/TCP	982	3,212	792679	1012	3,533	799770	1082	4,103	796629
QUIC/UDP	2199	4,319	887503	2247	3,584	892340	2141	3,808	888026

Tab. B.6: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Caddy) - vliv ztrátovosti

B.3 Přenos webové stránky

Protokol	Ztrátovost2,5%, Zpoždění 100ms			Ztrátovost 2,5%, Jitter 10ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	1299	8,532	994568	1685	6,786	1031555
HTTPS 1.1/TCP	1505	9,517	1031475	1837	7,004	1066793
HTTP2/TCP	945	6,921	784266	1280	4,713	836758
QUIC/UDP	1778	5,256	861950	2214	4,056	1054435

Tab. B.7: Naměřené hodnoty při přenosu velkého počtu souborů (webový server Caddy) - kombinace vlivů

Protokol	Zpoždění 100ms			Zpoždění 150ms			Zpoždění 200ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	277	1,243	191998	276	1,804	191932	276	2,347	191932
HTTPS 1.1/TCP	290	1,453	198692	292	2,212	198911	294	2,969	198985
HTTP2/TCP	208	1,064	182594	179	1,269	179082	215	2,126	183050
QUIC/UDP	218	1,556	181976	223	1,825	182316	229	2,135	182764

Tab. B.8: Naměřené hodnoty při přenosu webové stránky (webový server Caddy) - vliv zpoždění

Protokol	Jitter 10ms			Jitter 50ms			Jitter 150ms		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	320	0,237	209612	311	0,342	199478	302	0,665	195760
HTTPS 1.1/TCP	345	0,365	206509	350	0,467	208275	347	0,85	206849
HTTP2/TCP	315	0,226	204664	271	0,316	196242	296	0,637	201886
QUIC/UDP	459	0,571	294169	433	0,887	318980	458	2,134	330251

Tab. B.9: Naměřené hodnoty při přenosu webové stránky (webový server Caddy) - vliv proměnlivého zpoždění

Protokol	Ztrátovost 0.5%			Ztrátovost 2.5%			Ztrátovost 10%		
	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]	Počet paketů	Celkový čas přenosu [s]	Přenesená data [B]
HTTP 1.1/TCP	205	0,287	189488	234	0,401	189160	253	0,665	190848
HTTPS 1.1/TCP	266	0,362	198037	301	0,489	200129	325	0,782	203841
HTTP2/TCP	213	0,196	183689	225	0,243	180424	276	0,504	185498
QUIC/UDP	269	0,278	194779	253	0,285	184392	267	0,301	187786

Tab. B.10: Naměřené hodnoty při webové stránky (webový server Caddy) - vliv ztrátovosti

C OBSAH PŘILOŽENÉ SD KARTY

- *xmouck02_diploma_thesis.pdf* - elektronická verze této práce.
- *Laboratorni_prostedi.ova* - Appliance pro virtualizační prostředí VirtualBox, obsahující oba virtuální počítače, na kterých se provádí popisované úlohy. S nainstalovanými potřebnými webovými servery, konfiguračními skripty a včetně ovládacího skriptu a rolí pro Ansible.