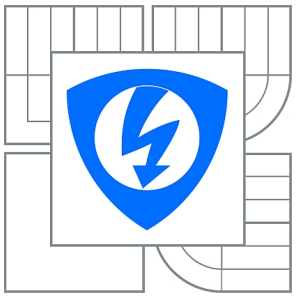
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VLIV PROTOKOLU HTTP 2.0 NA MODERNÍ DATOVÉ SÍTĚ

IMPACT OF HTTP 2.0 ON MODERN DATA NETWORKS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

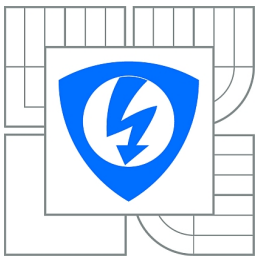
AUTOR PRÁCE — LENKA MAUREROVÁ

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADKO KRKOŠ

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Studentka: Lenka Maurerová ***ID:*** 146901
Ročník: 3 ***Akademický rok:*** 2013/2014

NÁZEV TÉMATU:

Vliv protokolu HTTP 2.0 na moderní datové sítě

POKYNY PRO VYPRACOVÁNÍ:

Popište chování a principy protokolů Gopher, HTTP 1.0, HTTP 1.1, HTTP 1.2, SPDY a HTTP 2.0 pro přenos hypertextových dokumentů. Tyto protokoly charakterizujte a porovnejte, přičemž se zaměřte zejména na vliv použití těchto protokolů na přístupové a transportní datové sítě. V popisu zahrňte pro jednotlivé protokoly formát kódování a strukturu zpráv, poskytované metody a prováděné operace, řešení chybových stavů, stavový model či podporu zabezpečeného přenosu. Popište také identifikaci zdrojů a koncept REST. Zrealizujte experiment, který bude demonstrovat rozdíly v chování a vlastnostech protokolů HTTP 1.1 a HTTP 2.0 při využívání služeb a výsledky diskutujte.

DOPORUČENÁ LITERATURA:

[1] PUŽMANOVÁ, Rita. TCP/IP v kostce. 1. vyd. České Budějovice: Kopp, 2004, 607 s. ISBN 80-723-2236-2.
[2] GOURLEY, David a Brian TOTTY. HTTP: The Definitive Guide. 1st ed. Sebastopol, CA: O'Reilly, 2002, xviii, 635 p. ISBN 15-659-2509-2.
[3] draft-ietf-httpbis-http2-06. Hypertext Transfer Protocol version 2.0. Marked for implementation. RFC: HTTPbis Working Group, 2013. Dostupné z: http://tools.ietf.org/pdf/draft-ietf-httpbis-http2-06.txt

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 4.6.2014

Vedoucí práce: Ing. Radko Krkoš
Konzultanti bakalářské práce:

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

ABSTRAKT

Tato bakalářská práce pojednává o protokolech Gopher, HTTP 1.0, HTTP 1.1, HTTP 1.2, SPDY, HTTP 2.0 a HTTPS, které slouží pro přenos hypertextových dokumentů a architektuře REST a vzájemně tyto protokoly prakticky porovnává.

KLÍČOVÁ SLOVA

Gopher, HTTP 1.0, HTTP 1.1, HTTP 1.2, SPDY, HTTP 2.0, HTTPS, REST

ABSTRACT

This thesis deals with the Gopher protocol, HTTP 1.0, HTTP 1.1, HTTP 1.2 SPDY HTTP 2.0 and HTTPS, which used to transfer hypertext documents, a REST architecture and the comparison of the protocols in practice.

KEYWORDS

Gopher, HTTP 1.0, HTTP 1.1, HTTP 1.2, SPDY, HTTP 2.0, HTTPS, REST

MAURER OVÁ, Lenka *Název: Vliv protokolu HTTP 2.0 na moderní datové sítě*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 53 s. Vedoucí práce byl Ing. Radko Krkoš,

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Název: Vliv protokolu HTTP 2.0 na moderní datové sítě“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu bakalářské práce panu Ing. Radku Krkošovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci a také Lukáši Andrysíkovi za poskytnutí serveru, na kterém byla tato práce testována.

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 bakalářské 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

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Pojem Internet je v dnešní době známý pro každého z nás. Jedná se o propojení velkého množství počítačových sítí komunikující přes TCP/IP. Internet obsahuje velké množství služeb, ale ta nejznámější se nazývá World Wide Web, která běžně používá protokol HTTP.

Hypertext Transfer Protocol slouží pro přenos hypertextových dokumentů. Předchůdce protokolu HTTP byl Gopher, který otevřel bránu hypertextovým dokumentům a vlastně i bránu komunikaci, zábavě a informacím po Internetu. Protokol HTTP se postupně vyvíjel a prošel velkými změnami.

První verze, která byla vypuštěna po Gopheru, byla HTTP 0.9, ale ta po čase nebyla dostatečná, jelikož byly nároky čím dál vyšší, proto přišly na řadu další verze: HTTP 1.0, HTTP 1.1, HTTP 1.2. Každá verze přinesla něco nového a užitečného. Nejnovější verzí je HTTP 2.0, která ještě není rozšířená a je ve fázi testování, ale měla by být revoluční a také by měla být přínosem pro mobilní zařízení, která v dnešní době ovládla svět. Tuto verzi doplnil protokol SPDY.

Vzhledem k tomu, že je HTTP přenos nešifrovaný, tak bylo potřeba, aby se vynalezl protokol, který bude šifrovaný, jelikož se postupně začaly objevovat webové stránky různých bankovních společností a také eshopy apod., kde lidé nakupují a je zde potřeba zadávat osobní údaje, které by mohly být zneužity. Proto byl vyvinut protokol HTTPS. Pokud se mluví o HTTP, tak pojem, který k tomuto protokolu neodmyslitelně patří je architektura REST, která je navržená pro distribuované systémy, jako je World Wide Web.

Informace obsažené v této práci jsem čerpala v první řadě z internetových zdrojů a také ze zdrojů knižních, které jsou uvedeny v sekci literatura.

1 TEORETICKÝ ÚVOD DO PROBLEMATIKY JEDNOTLIVÝCH PROTOKOLŮ

1.1 Gopher

Psal se rok 1991 a na Minesotské univerzitě byl vytvořen textový protokol zvaný Gopher, definovaný v RFC 1436 a používající protokol TCP na portu 70. Byl určen pro organizaci a lokalizaci informací v distribuovaném síťovém systému.

Tento protokol pracuje na aplikační vrstvě modelem klient - server viz obr. 1.1 a je bezstavový, tzn., že při každém dotazu musí klient se serverem navázat nové spojení. Pokud přijde další dotaz na server od stejného klienta, bere ho server jako úplně nový a nedává si ho do žádné souvislosti s dotazem předchozím. Což může být někdy výhodné, protože Gopher šetří kapacitu serveru - neudržuje se stálé spojení. Dá se snadno implementovat a je odolný. Jistou nevýhodou může být plýtvání přenosové kapacity, jelikož dotaz musí obsahovat přesnou informaci o tom, co klient požaduje.

Obr. 1.1: Klient - server

Základem Gopheru je menu. Pokud se uživatel připojil na server, tak obdržel jisté menu, ve kterém se pohyboval a vybíral z položek, ze kterých se dostal na nové menu. Takto bylo možné brouzdat celým Internetem, jelikož se mohlo stát, že položka vedla na další server. Server při dotazu odesílá informace o každém řádku z menu ve formě viz obr. 1.2.

Obr. 1.2: Princip Gopheru

Položka typ se zapisuje jednou číslicí, či jedním písmenem. Toto značení určuje o jaký typ dat se jedná viz tab. 1.1. Jméno je název položky, kterou chce uživatel zobrazit, selektor je jakési ID pro data na serveru, takže pokud uživatel sdělí selektor, dostane informace. Server a port udávají informace o tom, na jaký server se snažíme dostat.

Tab. 1.1: Typ dat

0	textový soubor
1	adresář (menu)
2	PH (CSO) telefonní seznam
4	hqx soubor pro Macintosh
7	vyhledávání
8	Telnet seance
9	binární soubor
s	zvuk
h	WWW stránka

Po tomto protokolu byla vydána i další vylepšená verze s názvem Gopher+, ale vzhledem k nástupu HTTP se přestal používat. [1]

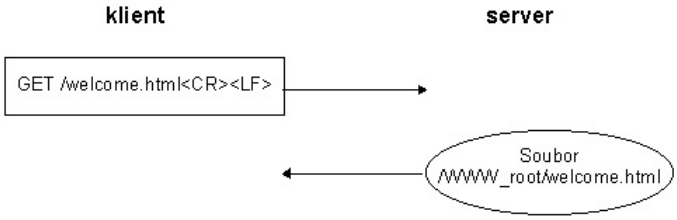
1.2 HTTP 0.9

Ve stejném roce, jako byl vynalezen Gopher, se objevil i nový protokol HTTP verze 0.9 a nadneseně řečeno - odstartoval éru World Wide Web.

HTTP, používající TCP a port 80, opět funguje na principu klient - server. Klient zasílá požadavek na server a server vrací odpověď.

U verze 0.9 je formát požadavku poměrně jednoduchý, jelikož obsahuje pouze jeden řádek - metodu GET viz obr. 1.3, za kterou je uvedena cesta k danému dokumentu, o který klient žádá. Pokud se takto odeslal požadavek na server a server požadovaný dokument našel, odeslal ho zpět klientovi jako dokument ve formě HTML. Pokud tento dokument nenašel, odeslal chybové hlášení taktéž v HTML.

Bohužel po nějaké době se zjistilo, že pouze metoda GET nestačí, jelikož uživatel ani nevěděl, jak veliký je požadovaný soubor, jakého je typu (to mohl zjistit pouze z přípony) apod., proto bylo potřeba tento protokol vylepšit. Vylepšení nastalo již v další verzi, která nesla označení HTTP 1.0. [2] [3]



Obr. 1.3: Struktura HTTP 0.9

1.3 HTTP 1.0

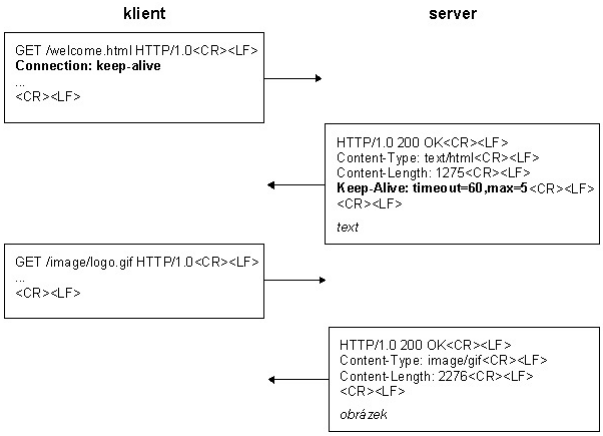
V roce 1996 byla vypuštěna do světa verze HTTP 1.0 popsána v RFC 1945.

Aby klient mohl získat co nejvíce informací o požadovaném souboru, přidaly se do požadavku a odpovědi serveru textové hlavičky, které byly ve tvaru: Jméno: hodnota - pro každou hlavičku vždy na novém řádku. Používání hlaviček je dobrovolné.

V této verzi se objevuje opět metoda GET, o které jsem se zmínila v předchozí kapitole a také dvě nové metody a to HEAD a POST.

Metoda HEAD umožňuje získat klientovi hlavičky zaslané serverem a z těchto hlaviček může klient zjistit např. kdy byl naposledy požadovaný soubor změněn apod.

Metoda POST slouží k zaslání odpovědi na formulář serveru. Hlavička je ukončena vždy prvním prázdným řádkem. V této verzi je možné volitelně použít udržované spojení (hlavička - Connection: Keep - Alive) viz obr. 1.4. [3]



Obr. 1.4: Struktura HTTP 1.0 a udržované spojení

Další poměrně důležitou novinkou byly stavové kódy. Stavový kód se skládá ze tří číslic a tyto číslice udávají, jak byl požadavek splněn. Každý stavový kód má

přiřazené stavové hlášení, které oznámí klientovi, jak byl jeho požadavek zpracován. Přehled stavových kódů viz obr. 1.5. [2]

Kód	Popis
	1xx -- Informační kódy (nepoužívá se)
	2xx -- Úspěšné vyřízení požadavku
200 OK	Požadavek byl úspěšně zpracován
201 Created	Výsledkem požadavku je nově vytvořený objekt
200 Accepted	Požadavek byl přijat, ale dosud není zpracován
200 No content	Požadavek byl úspěšně zpracován, ale jeho výsledkem nejsou žádná data pro klienta
	3xx -- Přesměrování
301 Moved Permanently	Požadovaný objekt byl trvale přemístěn na jinou adresu
302 Moved Temporarily	Požadovaný objekt byl dočasně přemístěn na jinou adresu
304 Not Modified	Objekt nebyl změněn (odpověď při podmíněném požadavku pomocí hlavičky **If-Modified- Since**)
	4xx -- Chyba klienta
400 Bad Request	Špatná syntaxe dotazu
401 Unauthorized	Objekt je dostupný pouze po autorizaci
403 Forbidden	Požadavek je v pořádku, ale server nemá povoleno jej vykonat
404 Not Found	Požadovaný objekt nebyl na serveru nalezen
	5xx -- Chyba na straně serveru
500 Internal Server Error	Serveru se něco stalo a nemůže vyplnit požadavek
501 Not Implemented	Server nepodporuje metodu uvedenou v požadavku
502 Bad Gateway	Server, pracující jako gateway, dostal špatnou odpověď od dalšího serveru
503 Service Unavailable	Služba je nedostupná (přetížení, údržba serveru)

Obr. 1.5: Stavové kódy HTTP 1.0

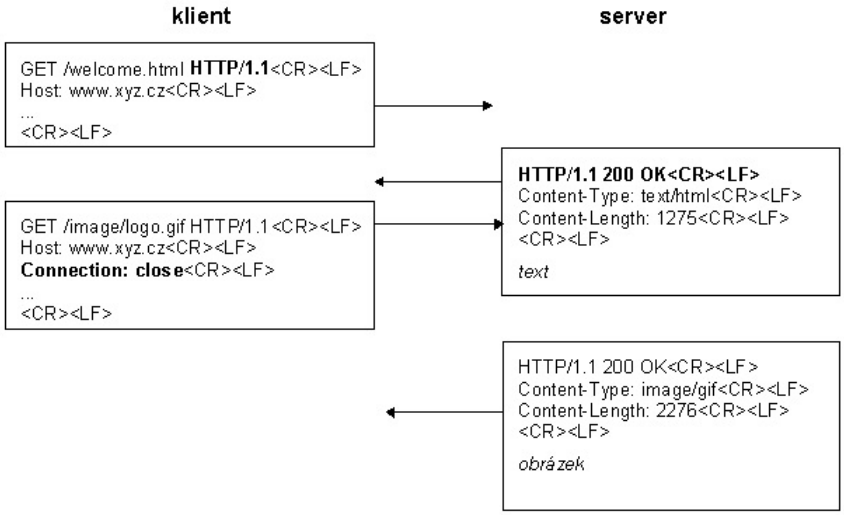
1.4 HTTP 1.1

Tato verze byla vydána v roce 1999 a je popsána v RFC 2086. Struktura zůstává stejná jako v předchozí verzi HTTP 1.0, ale jsou přidány nové hlavičky a některé jsou i povinné. Přibyly i nové metody např. PUT, DELETE, TRACE, CONNECT a OPTIONS.

Tato verze ale v sobě skrývá i další novinky. V minulých verzích se muselo pro každý požadavek/odpověď navazovat nové spojení, což bylo velmi neefektivní. Pokud jsme žádali jen o jednu HTML stránku, vše proběhlo rychle. Ale jakmile nastala situace, že jsme měli stránku, která obsahovala text a spoustu obrázků, musel se zasílat požadavek na text a počkat na odpověď. Po odpovědi se zaslal další požadavek na obrázek a opět se čekalo na odpověď a to trvalo tak dlouho, dokud se nestáhla celá požadovaná stránka. Postupem času začaly být webové stránky mnohem složitější a objemnější. Přenosové cesty byly velmi zatížené.

Ve verzi 1.1 je již možné navázat spojení jen jednou a v tomto spojení zasílat několik dotazů, aniž bychom čekali na vyřízení dotazu předešlého. Toto řešení nese název Pipelining. Pravdou je, že ve verzi 1.0 bylo také možné jisté udržované spojení (hlavička - Connection: Keep - Alive), ale bylo volitelné. Bohužel bylo řešeno nepatrně nešťastně, jelikož muselo být znovu požadováno a potvrzováno při každém požadavku. U verze 1.1 se musí spojení explicitně ukončit. Pokud klient i server podporují HTTP 1.1 trvá spojení tak dlouho, dokud nedojde časový limit, nebo není

spojení ukončeno explicitně - to má na starosti hlavička Connection viz obr. 1.6. [2] [3] [4]



Obr. 1.6: Struktura HTTP 1.1 a udržované spojení

1.4.1 Virtuální servery

Pokud měl dříve webový server nějaké virtuální servery, tak musel mít pro každý virtuální server svou síťovou adresu (HTTP 1.0), ale to znamenalo veliké plýtvání adres.

HTTP 1.1 musí obsahovat hlavičku Host, která musí obsahovat jméno serveru na straně klienta. Přítomnost této hlavičky je výhodná proto, aby server rozeznal, pro který virtuální server je požadavek určen. Což je velmi efektivní, jelikož všechny virtuální servery mají stejnou síťovou adresu.

1.4.2 Přenos dokumentů o neznámé délce

Ve verzi HTTP 1.0 bylo možné rozpoznat konec přenášeného souboru tak, že server ukončil spojení. U verze HTTP 1.1 je nutno v hlavičce Content - Length uvést délku přenášených dat při požadavku i při odpovědi obsahující data. Bohužel se může stát, že narazíme na soubor, kde není dopředu známo, jakou velikost bude mít (př. CGI skript). Taková data lze odesílat po částech, jejichž délka je známa. V tomto případě zašle server hlavičku Transfer-Encoding: chunked a každá velikost dané části je zapsána hexadecimálně, za níž následuje její obsah. Část je vždy ukončena prázdným řádkem. Poté může přijít další část a nebo prázdná s nulovou délkou. Za koncem dokumentu se mohou objevit i nepovinné dodatečné hlavičky.

1.4.3 Výběr typu dokumentu, kódování a jazyka

V této verzi protokolu se objevuje další novinka a to předem definovaný výběr dokumentu podle určitých atributů - jazyka (Accept - Language), typu prezentace (Accept), kódu (Accept - Charset) a způsobu přenosu (Accept - Encoding) - hlavičky.

Prohlížeč může jednotlivým atributům přiřadit váhu a to v rozmezí od 0 po 1.0 a je označena písmenem q. Pokud se tato váha neuvede, tak je automaticky přiřazená hodnota 1.0.

Toto je velmi výhodné pro to, abychom si nastavili například jako prioritní jazyk český, pokud by nebyl k dispozici, tak anglický apod. Tímto způsobem si můžeme navolit, co bychom rádi očekávali od serveru.

Dále také můžeme volitelně nastavit kódování jazyka (UTF-8, CP-1250 apod.) a poté server stránku překóduje podle požadovaného kódování. Dále můžeme požádat server jen o určitý formát stránky, audia, videa apod. Další poměrně užitečnou funkcí je komprimování (kódování) dat. To můžeme ocenit např. u JavaScriptu, CSS či (X)HTML.

1.4.4 Přenos části dokumentu

Někdy se může stát, že když je přenášen soubor, tak selže spojení a poté se musí přenášet znovu. U nevelkých souborů to není až tak veliký problém, ale u souboru větších to není nic příjemného.

O přenos části dokumentu se stará hlavička Accept-Ranges. V této hlavičce se nastaví určitý interval, který označuje první až poslední požadovanou slabiku dokumentu. Pokud se uvede v hlavičce více intervalů, tak přijde dokument po částech ve tvaru zprávy (MIME) a každá část obsahuje hlavičku Content-Range s daným intervalem.[3]

1.4.5 Struktura HTTP

Dotaz na server

Formát:

- <meotda><cesta><verze protokolu>
- <hlavičky>
- <prázdný řádek>
- <tělo požadavku>

Metoda - určuje akci od serveru
Cesta - určuje polohu k prostředku
Verze - určuje verzi protokolu HTTP

Odpověď serveru

Formát:

- <verze><stavový kód><popis>
- <hlavičky>
- <prázdný řádek>
- <tělo odpovědi>

Verze - určuje verzi protokolu HTTP
Stavový kód - je celočíselný návratový kód
Popis - popisuje význam návratového kódu

Hlavička

Poskytuje informace o zprávě a datech v těle zprávy.

Struktura hlavičky

- <název hodnoty>:<hodnota>
- <prázdný řádek>

Jak jsem již uvedla - v HTTP verze 1.0 není žádná hlavička povinná, ale ve verzi HTTP 1.1 jsou již určité hlavičky povinné.

Př:
Host: xx.xx.xx.xx (IP)
Date: Wen, 12 Jun 2013 14:52:32 GMT
Content-Type: image/jpeg
Connection: close [5] [6] [7]

1.4.6 Metody HTTP

Metody jsou implementovány od samého začátku vzniku HTTP. Sice jich ze začátku nebylo tak veliké množství, ale už v první verzi se objevila metoda GET. Metoda se vykoná nad určitým dokumentem. Existují metody bezpečné, jako je např. HEAD, GET apod., jsou určeny pouze pro získání informací a nevykonávají na serveru žádné akce, takže by neměly měnit jeho stav.

Dále existují metody nazývané jako nebezpečné, jako je např. DELETE, PUT apod. a ty mohou změnit stav serveru a někdy také změnit stav nežádoucím účinkem. Mezi další nebezpečné metody se považují TRACE, TRACK a DEBUG, jelikož mohou být zneužity útočníky k získání informací při útocích.

- GET Nejpoužívanější metoda, která slouží pro požadavek na zobrazení hypertextové stránky, RSS apod.
- HEAD Podobná metodě GET, ale zasílá pouze meta data, čili informace o datu, času, velikosti apod.
- POST Tato metoda se používá např. při odesílání formulářů na webovou stránku a odesílá data uživatele na server. Tuto vlastnost má i metoda GET, ale metoda POST se využívá hlavně pro objemnější data, která jsou větší než 512 bajtů. Také se používá v případě, kdy není vhodné zobrazit tyto data jako součást URL.
- PUT Velmi ojediněle používaná metoda odesílající data na server.
- DELETE Tato metoda smaže data ze serveru.
- TRACE Po obdržení požadavku odešle kopii zpět odesílateli.
- OPTIONS Metoda odešle dotaz na server, čímž zjistí, jaké metody server podporuje.
- CONNECT Metoda se používá pro spojení s proxy servery.
  Spojí se s uvedeným objektem přes uvedený port. Používá se při průchodu skrze proxy pro ustanovení kanálu SSL. [8]

1.5 HTTP 1.2

Další verzí protokolu, která se v konečné fázi neuchytila, byla verze HTTP 1.2. Některé z funkcí, které se objevily v této verzi, byly součástí již zmíněného protokolu Gopher, který byl předchůdce WWW. Jedná se hlavně o podporu pro rozhraní textového menu, které je vhodné pro mobilní klienty. Položky v menu jsou řazeny administrátorem serveru.

Mezi další vylepšení oproti předchozí verzi je, že využívá SRV záznamy pro lepší rozložení zatížení spoje a tato funkce se používá jak pro emailové, tak pro webové domény. Také je zde vylepšené ověřování Basic (jednoduchá autentizace při přístupu na webové stránky - zažádání serveru o heslo a uživatelské jméno) a Digest (server posílá před odesláním dat klientovi hash heslo) autentizace přístupu.

Tato nová verze protokolu byla vylepšena o novou sadu hlaviček. Nově přijaté hlavičky se dají přidávat pomocí IETF webové stránky (za poplatek).

Celkově HTTP 1.2 v sobě zahrnuje mnohem přísnější požadavky na spolehlivé plnění svých funkcí oproti HTTP 1.1.

Rozšířením tohoto protokolu se stal protokol PEP (protocol extension protocol). Je to systém pro HTTP klienty, servery a proxy servery a díky němu je zajištěna dynamická rozšířitelnost protokolu HTTP. Do HTTP 1.2 byl plně implementován, ale pravdou je, že je částečně kompatibilní také s verzemi HTTP 1.0 a HTTP 1.1.

PEP je určité rozšíření HTTP, které slouží pro nestandardizovanou formu komunikace v HTTP mezi klientem a serverem. Při dotazu klienta na server s tímto rozšířením přijme server dotaz a odešle URI odkaz s odpovědí na rozšíření (specifikace, odkaz na spustitelný kód, meta popis apod.). Klient přijme odpověď a vykoná akci, která od serveru přišla (spustí kód, implementuje rozšíření apod.). Pokud s tímto rozšířením nebude umět klient pracovat, má možnost rozšíření přeskočit a pokračovat dál a nebo celou komunikaci se serverem ukončí. [9] [10] [11]

1.6 HTTPS

Komunikace přes HTTP je nešifrována, což může mít fatální následky. Pokud navštívíme pouze obyčejnou webovou stránku, která není obsahově nikterak důležitá, měl by být využit klasický HTTP. Pokud přistupujeme na webovou stránku, která patří např. nějaké bankovní instituci, či je to e-shop, kde se dá platit platební kartou a zadáváme citlivé údaje, měl by tento přístup být šifrovaný pomocí SSL/TLS, využívající port 443 na straně serveru.

Klient i server si před samotnou komunikací vymění privátní a veřejný klíč. Při zahájení komunikace si mezi sebou vymění veřejný klíč, který by si měly ověřit např. digitálním certifikátem. Pokud by neproběhlo toto ověření, je klient i server ohrožen útokem třetí strany (Man in the middle).

Pokud vstupujeme poprvné na zabezpečenou webovou stránku, prohlížeč nás upozorní a poskytne nám informace o digitálním certifikátu. Taktéž je uvedeno v adresním řádku, že vstupujeme na webovou stránku přes HTTPS.

Komunikace přes HTTPS může být lehce nekomfortní a to z důvodu mírného zpomalení oproti HTTP, ale myslím si, že bezpečnost je vždy na prvním místě, proto bych zpomalení nepovažovala jako nevýhodu.

Rozsah bezpečnosti záleží na daném certifikátu a uživateli, serveru, algoritmu šifrování apod.

Kvůli virtuálním serverům se muselo do RFC-3546 definovat rozšíření s názvem Server Name Indication (SNI), které mají v sobě implementovány prohlížeče. Díky této implementaci je možné vytvořit na jedné IP adrese více virtuálních HTTPS s různými doménovými jmény. [12]

1.7 SPDY

Postupem času se staly webové stránky náročnějšími a obsahově mohutnějšími a přenos je náročnější a hlavně zdlouhavější. Říká se, že dnes žijeme v době, kdy je vše

uspěchané a rychlé. Proto je prioritní, aby byl i přenos webových stránek co nejrychlejší. Vývojáři v Googlu přišli s novinkou zvanou SPDY (SPeeDY).

Jedná se o protokol, který je úzce spjat se známým prohlížečem Chrome a ten je taktéž majetkem Googlu. Do prohlížeče je tento protokol implementován a dá se říci, že je v něm i testován. Podle předpokladů by mohl zrychlit celý přenos až o 60-78 procent.

Jak jsem se již zmínila, HTTP ve vyšších verzích využívá pipelining. Můžeme serveru poslat při jednom spojení více dotazů a on je postupně bude zpracovávat. Pipelining byl opravdu krok dopředu, ale stále má své nedokonalosti. Tou hlavní nedokonalostí je, že využívá principu FIFO, což znamená, že server bude postupně zpracovávat dotazy v takovém pořadí, v jakém k němu přišly, což může být někdy nevýhodné a to například v takovém případě, pokud přijde první dotaz poměrně náročný a obsáhlý, tak ostatní, byť i jednodušší dotazy, musí počkat ve frontě, než server první dotaz zpracuje a až poté se dostane k dotazům dalším.

1.7.1 Více požadavků s prioritou

Výše zmíněný problém řeší protokol SPDY, který dokáže zpracovávat dotazy podle určené priority, což je velmi výhodné. Funguje tedy na principu pipeliningu, ale bez fronty FIFO. Může se stát, že bychom měli pomalé připojení a potřebovali bychom načíst náročnou webovou stránku se spoustou velkých souborů. V tom případě může klient poslat na server více požadavků naráz a těm určí prioritu (0-7), aby se nejprve načetlo HTML a až poté kaskádové styly, javascript apod.

1.7.2 TCP spojení, nové rámce, komprese

Další novinkou je způsob, kdy SPDY přidává relaci nad vrstvou SSL, která umožňuje po jednom TCP spojení přenášet libovolný počet dat, streamů. Každý stream má svoje jedinečné ID a přenášená data toto ID znají. Takže obě komunikující strany dokáží poznat, jaká data patří k jakému streamu.

V SPDY se objevují dva nové rámce s názvem Contol a Data, které slouží ke snažší identifikaci. V HTTP bylo možné použít kompresi na celý obsah. V protokolu SPDY se používá automatická komprese nejen na samotný obsah, ale i na hlavičky. Přibyly i nepovinné funkcionality jako je Server push a Server hint. [13] [14] [15]

1.7.3 Server push, Server hint

Server push: Novinkou oproti HTTP je, že server může sám od sebe začít odesílat data klientovi, která si od něj prozatím nevyžádal. Tuto informaci oznámí server

klientovi v hlavičce. Pokud klient navštíví stránku, kterou nikdy předtím nenavštívil, bude načítání rychlejší.

Server hint: Pokud má klient pomalé připojení, může být výhodou to, že server má možnost informovat klienta o datech, která jsou pro něj potřebná, aniž by mu začínal data odesílat. Proto může klient rychleji zareagovat na to, jaká data v danou chvíli potřebuje a která jsou pro něj zbytečná. [14]

1.7.4 Budoucnost

Podle vyjádření Googlu se nejedná o jakousi náhražku protokolu HTTP, ale je vyvinut proto, aby HTTP doplnil a vylepšil ve slabších místech, jako je přenos dat.

Tento protokol je implementován i do známých webových prohlížečů, jako je např. Mozilla Firefox, Opera, IE 11 a již zmíněný Google Chrome. SPDY funguje už i na Facebooku, Twitteru v Googlu i ve Wordpressu. Existují i webové stránky, kde si můžeme otestovat podporu SPDY v našem prohlížeči a také ověření podpory webových stránek. Vzhledem k tomu, že je to protokol „nové generace“, tak je v návrhu pro připravovaný standard HTTP 2.0, jehož dokončení by mělo být koncem roku 2014. [14] [16]

1.8 HTTP 2.0

Kdyby někdo před pár lety ukázal vývojářům webové stránky, které běžně navštěvujeme dnes, asi by se podivili. Nikdo netušil, že vývoj půjde takovou rychlostí. Protokol HTTP 1.1 nám již slouží pěknou řádku let a můžeme s upřímností prohlásit, že princip a struktura protokolu zůstala od prvního vypuštění až do dnes skoro stejná. Protokol pracuje velmi slušně, ale bohužel už dnešní době nepostačuje. Je načase, aby přišel nový, kvalitní, rychlý a bezpečný protokol, který bude uživatelům šetřit drahocenný čas, nebude tolik zatěžovat síť a prodlouží (i když nepatrně) baterii u přenosných zařízení.

Již jsem v předchozí kapitole naznačila, že se pomalu, ale jistě dostává na povrch protokol s názvem SPDY. Přesněji řečeno protokol, který bude implementován do protokolu HTTP (jádro zůstane zachovalé) a vznikne nový binární protokol s názvem HTTP 2.0.

Protokol SPDY je již v testování nějaký čas, ale běžní uživatelé, myslím si, to ani nezaznamenali. Pomalu, ale jistě organizace International Engineering Task Force dokončuje specifikaci nového protokolu HTTP 2.0 a bude připravený k implementaci.

Dalo by se tedy říci, že za vznik nového protokolu se do značné míry zasloužila společnost Google.

1.8.1 Šifrování

Starší verze protokolů nejsou defaultně šifrovány. Pokud je potřeba přenos šifrovat, musí se použít nástavba SSL, nebo TLS. Protokol HTTP 2.0 ale čeká změna. Šifrování by mělo být defautní a každý přenos mezi klientem a serverem by měl být šifrovaný. Existují tři návrhy, jak protokol šifrovat v základu.

První návrh šifrování

Oportunistické šifrování - pokud bude možné šifrovat, tak se šifrování automaticky zapne, pokud ne, přepne se a komunikace bude nešifrovaná. Toto všechno bude probíhat bez ověření identity serveru.

Druhý návrh šifrování

Druhý návrh bude naprosto stejný jako přechozí, pouze bude s ověřením identity serveru.

Třetí návrh šifrování

V posledním návrhu se jedná o povinné schéma HTTPS pro tento nový protokol. Pro obyčejné HTTP by se používala starší verze HTTP 1.1. [16] [17]

1.9 Identifikátor zdroje

1.9.1 URI

URI (definováno v RFC 3986) neboli Uniform Resource Identifier je jednotný zdrojový identifikátor (slouží k přesné identifikaci zdrojů dat). Tento identifikátor může zdroj popsat tak, že buď určí jeho identitu, ale neuvede místo nalezení a nebo naopak - nepopíše identitu, ale popíše, jak je možno nalézt zdroj. Také umí tyto dvě kombinace dohromady.

Struktura URI
schéma:hierarchická část?dotaz#fragment

- **Schéma** - určuje, o jaký druh URI se jedná a může obsahovat pouze písmena (velká i malá), číslice a znaky plus, minus a tečku
- **Hierarchická část** - obsahuje identifikátor zdroje podle hierarchické struktury
- **Dotaz** - slouží k bližšímu určení zdroje a popisuje nehierarchickou část identifikátoru

- **Fragment** - popisuje nepřímo doplňkový zdroj (popisuje konkrétní část např. kapitolu knihy apod.) [18]

1.9.2 URL

URL (definováno v RFC 1738) neboli Uniform Resource Locator je jednotný zdrojový lokátor sloužící k přesné specifikaci umístění zdrojů dat. Strukturou URL je doménová adresa serveru, umístění zdroje na serveru a také protokol. Takže popisuje způsob, jakým se dá ke zdroji dostat.

Struktura URL
protokol://server.doména druhého řádu.generická doména:port/umístění v rámci serveru?formulářová data (parametry)#kotva

Některá pole jsou nepovinná, protože buď nemají určitý význam a nebo se předpokládá, že bude hodnota již předdefinována. Kotva slouží jako odkaz na zdroj a také může sloužit jako odkaz na konkrétní informace ze zdroje. [19]

1.9.3 URN

URN (definováno v RFC 2141) neboli Uniform Resource Name je jednotný jmenný identifikátor, který neřeší dostupnost zdroje. URN se často přirovnává k ISBN, což je jednoznačný identifikátor knihy.

Struktura URN
Uvedené fráze v uvozovkách se považují za povinné!
<URN> ::= „urn:“ <NID> „:“ <NSS>

- **NID** - jmenný prostor identifikátoru
- **NSS** - specifický řetězec jmenného prostoru
- **urn:** - sekvence je citlivá na velikosti písmen [20]

1.10 REST

V roce 2000 v disertační práci Architectural Styles and the Design of Network-based Software Architectures její autor Roy Fielding popisuje a navrhuje architekturu REST. Roy Fielding je jedním se spoluzakladatelů HTTP, takže je více než patrné, že REST a HTTP k sobě neodmyslytelně patří.

REST (Representational State Transfer) je architektura rozhraní, která je navržená pro distribuované systémy, jako je např. World Wide Web, byla souběžně

navrhnuta s protokolem HTTP 1.1 a je orientována na zdroje - je použitelná pro snadný a také jednotný přístup ke zdrojům (resources). Zdroji se rozumí jakákoliv data (informace), která mohou mít různou reprezentaci (XML, PDF, JSON, SVG, HTML). Všechny tyto zdroje v síti mají svůj vlastní identifikátor URI a REST definuje čtyři metody nazývané CRUD, jak k těmto zdrojům přistupovat. Mezi tyto metody patří Create (vytvoření dat), Retrieve (získání dat), Update (změna dat) a Delete (smazání dat) a jsou implementovány pomocí metod protokolu HTTP.

1.10.1 Retrieve - GET

Získání zdroje opatřuje metoda GET. S touto metodou se každý denně setkává. Pomocí této metody v HTTP vyšleme požadavek od klienta na server a server odešle data konkrétního zdroje.

1.10.2 Create - POST

Tato metoda je velmi známá např. z tvorby webových formulářů pomocí HTML. Pokud se použije metoda POST, tak v okamžiku volání metody není známý žádný identifikátor, jelikož ještě není vytvořen. K vytvoření dat je nutné zavolat pomocí HTTP - POST zdroj s URI. Ve statusu se předá text pro novou zprávu a volání se následně autorizuje (jméno a heslo autora). Po takovém odeslání vrátí server klientovi kód (v HTTP je to kód 201), který obsahuje URI nově vytvořeného zdroje. Pokud by došlo k chybě, server vrátí chybový kód.

1.10.3 Delete

Tato metoda je velmi podobná již zmíněné metodě GET. Pokud klient zavolá URI pomocí metody DELETE, následně budou data na serveru smazána. Může se stát, že metodu DELETE nebude HTTP podporovat a bude omezen pouze na metodu POST a GET - v tomto případě se použije metoda POST s dodaným parametrem, který informuje, že bude ve skutečnosti použita metoda DELETE.

1.10.4 Update

Metoda je podobná metodě POST s tím rozdílem, že v této metodě je již známé URI a může být zadáno a v těle předáme serveru novou hodnotu. Opět tato metoda nemusí být podporována, proto se mohou používat alternativní parametry.

1.10.5 Základní principy

Komunikace probíhá mezi klientem a serverem a komunikační rozhraní musí být jednotné (Uniform Interface). Fielding požaduje, aby se mezi klienta a server daly vkládat další, na sobě nezávislé a neviditelné vrstvy, jako je například vrstva starající se o průběžné ukládání dat do mezipaměti (cache). Vrstvy řídící příchozí požadavky a podle dané priority je předává dále, přičemž vyrovnává zátěž daného serveru (load balancing) a vrstvy, které se starají o zabezpečení webových služeb.

1.10.6 Cache

Jak jsem již uvedla, cache je vyrovnávací paměť (musí být podporována aplikační vrstvou), do které se průběžně ukládají data. Toto je velmi výhodné, jelikož server nemusí data pokaždé zpracovávat (rychlejší odezva serveru), což může být výhodné i při vyskytnutí se poruchy - data mohou být z vyrovnávací paměti odeslána klientovi, pokud si o ně zažádá. Další výhodou je, že server může obsluhovat více klientů posílající serveru stejné požadavky. Někdy se ale může stát, že data budou ve vyrovnávací paměti nekompletní a tak mohou vznikat chyby v aplikaci.

1.10.7 Bezstavovost

Důležité je, aby každý požadavek, který je zasílán na server a ze serveru, obsahoval všechny informace, které slouží k jeho vykonání - tím pádem servery nemusí uchovávat žádný stav.

1.10.8 HATEOAS

Celým názvem Hypermedia as the Engine of Application State. REST potřebuje ke své práci s aplikací pouze vstupní URI. Další informace si dokáže dynamicky najít pomocí odkazů, které dostane klient od serveru.

1.10.9 Code-On-Demand

Jedná se o mechanismus (kód na vyžádání), kdy si klient může od serveru vyžádat kód dalších funkcí (různé skripty či applety), ale nemusí. Server je automaticky nepošle, pokud je klient nebude požadovat. Tento mechanismus se používá např. u JavaScriptu.

1.10.10 Hlavní cíle REST

- použití různorodých komponent - serverů a klientů

- všeobecnost rozhraní
- podpora s komponentami za účelem zvýšení bezpečnosti, zapouzdření a snížení latence [21] [22]

1.11 Proxy, brána a tunel

1.11.1 Proxy

Proxy server je mezistanice, která přebírá požadavky od klientů a následně je posílá na cílový server. Cílový server vyřídí požadavky, pošle je na proxy server a ten je pošle klientovi. Takže se dá říci, že pro klienta je proxy jako server a pro server je proxy jako klient – podobně jako router, ale s tím rozdílem, že router nevyřizuje požadavky, pouze je přeposílá. Proxy může vyřizovat jak HTTP požadavky, tak ale i FTP, POP3, SMTP atd. Proxy server má také cache paměť, která slouží k tomu, aby uložil požadavek, který již byl jednou vyřizován. Pokud by přišel opět stejný, nemusí se znovu vyřizovat a rovnou je odeslán.

Proxy také umí filtrovat určité protokoly – požadavky vyřizované na povolených portech a některé umí i filtrovat konkrétní protokoly (podle IP, DNS jména), nebo obsah odpovědi. Funguje tedy jako firewall. Je také vhodný pro ochranu soukromí, jelikož veškeré informace o hlavním klientovi jsou pro cílový server utajeny. Některé se mohou použít pro ochranu přístupu k privátní síti, autentizaci, dešifrování, kompresi dat, zvýšení výkonu TCP apod.

1.11.2 Brána

I když má proxy server s gateway mnoho společného, tak gateway má na starosti konverzi mezi dvěma aplikačními protokoly. Může, ale i nemusí pracovat s různými tansportními protokoly. Často se využívá gateway mezi protokoly HTTP a FTP, GSM a Internetem, ale také se používá např. mezi SMTP a X.400, Internetem a X.25, mail a FTP, mail a HTTP. Default gateway je směrovač, který zajišťuje připojení do vnější sítě.

1.11.3 Tunel

Tunelování se využívá ve VPN, což je virtuální síť. Je to metoda, která přenáší data po speciálně vytvořeném tunelu po síti – mezi dvěmi stanicemi. Nejběžnější tunelové spojení mezi cílovým a zdrojovým směrovačem je GRE General Routing Encapsulation protocol (všeobecný zapouzdřovací protokol). [23] [24]

2 PŘÍPRAVA PRO MĚŘENÍ

2.1 Ubuntu

Ubuntu, neboli „lidskost“ jak zní překlad ze zulštiny, je distribuce operačního systému Linux založená na Debian GNU/Linux a je volně šířitelná tj. může být zdarma použita na počítačových stanicích, noteboocích či serverech neomezeného počtu a to jak pro osobní účely, tak pro veřejné účely. Totéž platí pro software, který je součástí Ubuntu. Je zde minimální riziko zavirování nebo jiného narušení systému. Velkou výhodou je, že každý půlrok vyjde nové vydání a jsou zdarma i jakékoliv pravidelné aktualizace. Pro základní používání je k dispozici předinstalovaný balík základních programů důležitých pro práci (textový editor, internetový prohlížeč apod.). Pro stažení dalšího potřebného softwaru slouží Centrum softwaru. Dále jsou k dispozici čtyři pracovní plochy, které usnadňují práci. Ubuntu je jak ve 32 bit, tak ve 64 bit verzi a ve verzi pro server. Také existuje jako Live verze (na flash paměť či CD) pro vyzkoušení bez instalace na pevný disk. Systém je lokalizován do 25 světových jazyků, včetně jazyka českého a obsahuje grafické rozhraní.

Za distribuci operačního systému jsem zvolila Ubuntu v české lokalozaci ve verzi 32 bit Ubuntu Desktop 12.04.4. Tuto verzi jsem si vybrala hlavně z důvodu podpory Apache 2.2.22 a následné podpory protokolu SPDY pro Apache. Dalším důvodem pro výběr této distribuce byl fakt, že jsem nikdy předtím s operačním systémem Linux nepracovala a tato distribuce mi byla pocitově nejpřívětivější. Poměrně lehce jsem se naučila ovládat terminál se základními příkazy a dokázala jsem se zorientovat v grafickém prostředí.

Instalace proběhla klasicky a bez jakéhokoliv problému. Po instalaci bylo zapotřebí nainstalovat webový server Apache, který je potřeba pro otestování protokolů a vložení webových stránek sloužící jako předmět pro toto otestování. [25]

2.1.1 Práva v Ubuntu

Každý soubor či adresář je chráněný právy pro běžné uživatele. Správce, neboli root, těmito právy omezen není. Tím je systém chráněn před přepsáním konfiguračních souborů - podstatnou změnou systému. Každý uživatel, který vytvoří nějaký soubor, tak se stává i vlastníkem. Vlastník si může nastavit práva pro uživatele/skupinu, jak uzná za vhodné. Práva se dají nastavovat jak v grafickém režimu, tak také v terminálu, kde to většinou bývá jednodušší.

Práva k souborům zjistíme příkazem *ls –l* (*ls –la* pro skryté soubory). Výpis může vypadat následovně viz obr. 2.1

Výpis je ve formátu:

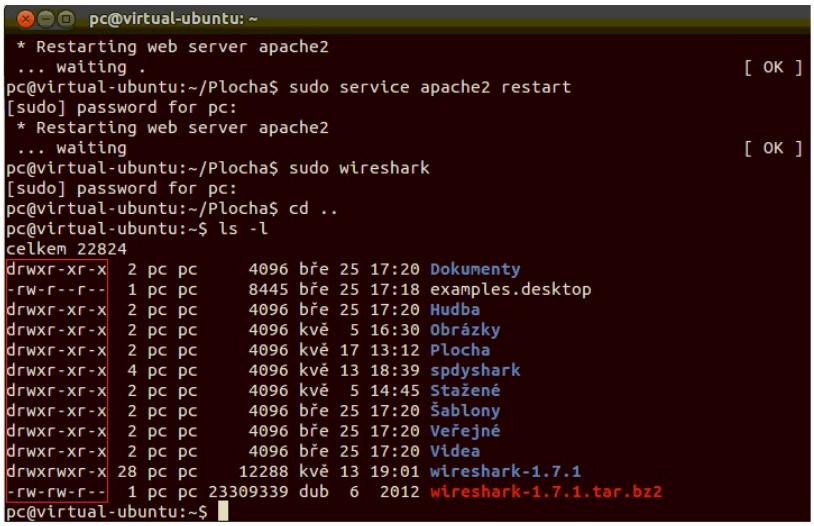
TYP – PRÁVA VLASTNÍKA – PRÁVA SKUPINY – PRÁVA OSTANÍ

Typ (jednoznakový)

- - (soubor)
- c (zařízení)
- b (disky)
- d (adresář)
- l (link)

Další tři sloupečky slouží pro nastavení práv pro uživatele, skupiny nebo ostatní.

- r – Read (čtení souboru, výpis obsahu adresáře)
- w – Write (zápis do souboru, mazání a vytváření souborů a adresářů)
- x – eXecute (spuštění souboru, procházení adresářů)



Obr. 2.1: Výpis práv v Ubuntu

Př.

-rwxr-xrw-

- - jedná se o soubor
- rwx – vlastník může číst, zapisovat a spouštět
- r-x – skupina může číst a spouštět
- r-w – ostatní mohou číst a zapisovat

Změnu práv můžeme nastavit dvěma způsoby. První možností je nastavení pomocí příkazu **chmod, znaku a jména souboru/adresáře.**

- u - user (vlastník)
- g - group (skupina)
- o - others (ostatní)
- a - all (všichni)

Pro přidání práv použijeme +, pro odebrání – a pro kompletní změnu =.
Další možností je **číselné nastavení**.

Právo hodnota

- r (4)
- w (2)
- x (1)

Práva jsou nastavena součtem těchto hodnot.

Př. 4 5 7

r- r-x rwx (4, 4+1, 4+2+1)

Zápis do terminálu

Sudo chmod 457 jméno souboru [26]

2.2 Apache

Webový server neboli webserver může být stanice nebo software, který vyřizuje HTTP požadavky od klientů, kterými bývají nejčastěji webové prohlížeče. Server požadavek přijme a vrací klientovi odpověď – HTML stránku, obrázek apod. Vznik prvního webového serveru se datuje k době 1990 a nesl název CERN httpd a za vývojem stojí Tim Berners-Lee, Ari Luotonen a Henrik Frystyk Nielsen a byl vyvinut na systému NeXTSTEP.

Obsah požadavku může být buďto statický, nebo dynamický. Požadavek statické webové stránky je beze změny zpracován a odeslán. Pokud je požadavek dynamický (PHP apod.), musí být požadavek „zformátován“ a poté je odeslán zpět klientovi. Odpověď je vrácena ve formě HTTP hlavičky a obsahu.

Jak jsem již zmínila, tak v hlavičce HTTP protokolu je udáván stavový kód, který popisuje, jestli byl požadavek zpracován úspěšně, nebo chybně (viz stavové kódy). Webový server má velké množství možností nastavení, které jsou uvedeny níže. Webový server může být také napadem nějakým útokem (DDoS), nebo virem. Server je pak přetížen – má pomalou odezvu, vyskytují se chyby 5xx.

Nejpoužívanějším webovým serverem je Apache. Apache je distribuován jak pro operační systém Linux, tak pro Windows, Mac OS, BSD apod. Jeho výhodou je, že má otevřený kód.

Vývoj začal v roce 1993 v National Center for Supercomputing Aplications (NCSA) na Illinoiské univerzitě programovaný Robem MCCoolem. Původně byl vyvíjen pod názvem NCSA HTTPd. Později si NCSA HTTPd pod svá křídla vzali Brian Behlendorf a Cliff Skolnick. Poté byl celý kód přepsán a upraven a vzápětí vznikla Apache Group a název Apache2 nahradil NCSA HTTPd.

Apache podporuje i Virtual Hosts, což je provozování více serverů na jednom zařízení, které jsou rozlišeny jinými doménovými jmény. [27]

2.2.1 Instalace a konfigurace Apache

Instalace Apache na operační systém Ubuntu je velmi snadná a to zadáním příkazu *sudo apt-get install apache2* do terminálu. Při zadání příkazu se objeví výzva pro zadání rootovského hesla. Poté se již automaticky Apache nainstaluje.

Konfigurace Apache je bohatá na různá nastavení, přídavné moduly apod. Je výhodné instalovat rovnou z distribuce přes terminál, jelikož různé distribuce mají různý přístup ke konfiguračním souborům.

V Ubuntu je hlavní kořenovou složkou pro různá nastavení Apache2 nacházející se v */etc/apache2*. Nejdůležitějším souborem je *apache2.conf*. Pokud je potřeba doplnit nastavení, použije se soubor *httpd.conf*.

Soubor *ports.conf* slouží pro to, pokud by bylo potřeba, aby Apache naslouchal na jiném portu, než je defaultní port 80 a slouží také ke konfiguraci IP adres. Adresář *conf.d* slouží pro dodatečné konfigurační soubory.

2.2.2 Mods-available, mods-enabled

Slouží jako „šuplíky“ pro dodatečné zásuvné (rozšiřující) moduly. V základní kofiguraci jsou již nejběžnější a nejpoužívanější moduly nainstalované. Je ale možné tyto moduly rozšířit (např. PHP, SSL, SPDY apod.). Jen je potřeba moduly zavést a nastavit. Ke každému modulu je konfigurační soubor, který je umístěn v adresáři *mods-available* s příponou *.conf*. Soubory obsahující příponu *.load* slouží pro nasazení modulů. Všechny aktivní moduly jsou v adresáři *mods-enabled*. Ty, co jsou v adresáři *mods-available* Apache ignoruje. Takže z *mods-available* se „natahují“ moduly do *mods-enabled*. Pro aktivování a deaktivování modulů slouží příkazy *–a2enmod* a *–a2dismod*.

2.2.3 Sites-available a sites-enabled

Tyto dva adresáře slouží pro práci s virtuálními webovými hosty, jejichž použití je velmi běžné. Opět Apache ignoruje to, co je umístěné v adresáři *sites-available* a to, co je v adresáři *sites-enabled* je pro něj klíčové. Opět jako u modulů – existují dva příkazy pro aktivaci a deaktivaci virtuálních hostů a k tomu slouží příkazy *-a2ensite* a *-a2dissite*. Jako hlavní adresář, který je určen pro nahrání webových stránek je adresář nacházející se ve *var/WWW/*. Hlavním souborem je *index.html*. Index lze spustit zadáním „localhost“ do prohlížeče, anebo zadáním 127.0.0.0.

2.2.4 Nastavení portů

Při změnách v konfiguraci je důležité, aby měly složky, či soubory nastavená práva pro přepisování. Toho docílíme např. tím, že každé složce/souboru, který potřebujeme měnit, nastavíme práva tím, že zadáme do terminálu příkaz např. *sudo chmod 777 httpd.conf*. Poté již nebude žádný problém s ukládáním a přepisováním či přesouváním. Při každé změně konfiguračních souborů je nutné Apache restartovat příkazem *sudo\etc\init.d\apache2 restart*.

V souboru *ports.conf* jsem nastavila poslouchání na klasickém *portu 80* a pro šifrovaný přenos klasický a běžně používaný *port 443*. V soboru *httpd.conf* jsem využívala nastavení přídavných modulů, které byly potřeba pro další testování protokolů, o kterých se podrobněji zmíním později.

V adresáři *site-enabled* v souboru *000-default* jsem zkontrolovala nastavení *portu 80* a *ServerName*, čili jméno používaného serveru.

Totéž v souboru *default-ssl*, ale pro *port 443*. Tento soubor slouží pro nastavení šifrované komunikace. V tomto souboru je také možné nastavit, aby ssl bylo aktivní, nebo neaktivní - *SSLEngine on – SSLEngineoff* a jsou zde také informace o certifikátu, které odkazují na soubory:

\etc\apache2\ssl\apache.ctr
\etc\apache2\apache.key

Pro testování bylo potřeba aktivovat ssl modul příkazem:
sudo a2enmod ssl
sudo service apache2 restart

2.2.5 Složka pro certifikáty

Příkazem *sudo mkdir\etc\apache2\ssl* vytvoříme složku.

poté je třeba zadat

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout etcapache2sslapache.key -out etcapache2sslapache.crt
(certifikát s platností na 365 dní)

Dále se objeví výzva k vyplnění údajů o serveru, pro který je certifikát určený:

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank For some fields there will be a default value, If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:NYC
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Awesome Inc
Organizational Unit Name (eg, section) []:Dept of Merriment
Common Name (e.g. server FQDN or YOUR name) []:example.com
Email Address []:webmaster@awesomeinc.com

Dále otevřeme soubor *default-ssl* např. v nano. Lze ale i otevřít klasicky, je ale potřeba nastavit práva pro přepsání.

nano\etc\apache2\sites-available\default-ssl

Zde je potřeba nastavit *ServerName andrysik.eu:443* a také je potřeba zkontrolovat cestu k souborům a zapnout *SSLEngine*.

SSLEngine on
SSLCertificateFile \etc\apache2\ssl\apache.crt
SSLCertificateKeyFile \etc\apache2\ssl\apache.key

Poté stačí už jen uložit, zapnout virtuálního hosta a restartovat Apache.

sudo a2ensite default-ssl
sudo service apache2 restart [28]

2.2.6 Modul SPDY

Tímto jsem nastavila podporu ssl. Dále je potřeba doinstalovat modul pro protokol SPDY.

Modul pro SPDY slouží k podpoře protokolu SPDY v serveru. Po nastavení aktivního SPDY využívá server pro komunikaci právě tento protokol. Nejjednodušší instalací je stažení a instalace ze software centra. Další možností je instalace pomocí terminálu a to zadáním příkazů:

Sudo dpkg -i mod-spdy-.deb*

a pro instalaci

sudo apt-get -f instar

Poté už jen stačí restartovat Apache.

Pro konfiguraci tohoto modulu slouží soubor s názvem *spdy.conf* uložený v *\etc\apache2\mods-available*

Modul se dá zapnout/vypnout na řádku

SpdyEnabled on/off

Dále se v tomto souboru dají nastavit toky a žádosti.

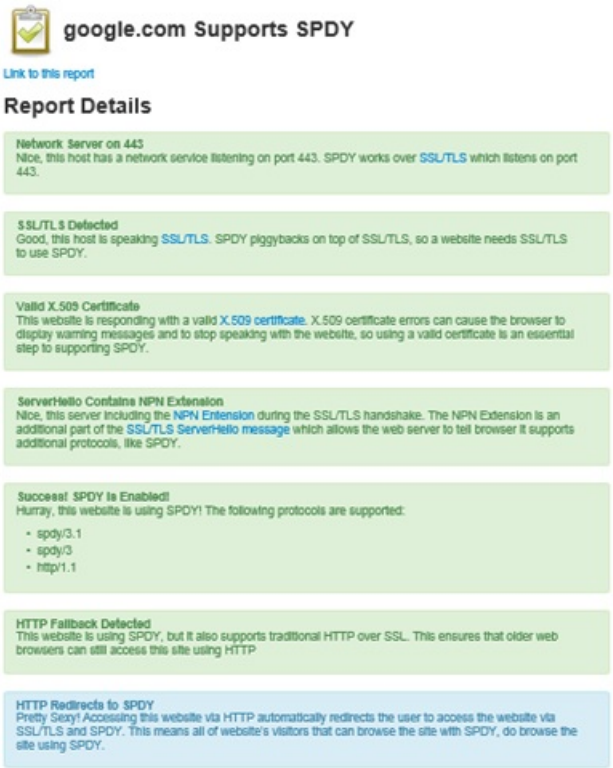
SpdyMaxThreadsPerProcess n

SpdyMaxStreamsPerConnection n

Takto by měl být Apache server připraven pro komunikaci s podporou SPDY.

Jako výchozí prohlížeč jsem používala Google Chrome, který má v sobě funkci, která podporuje SPDY.

Pro zjištění, zda je vše správně nastavené jsem použila internetovou stránku *http://spdycheck.org/*. Po zadání doménové adresy serveru se otestuje podpora SPDY.



Obr. 2.2: Podpora SPDY

Př. zadání *google.com*, který SPDY podporuje, vrátí server výsledek viz obr. 2.2

Jsou zde vypsány všechny užitečné informace o zapnutém SSL, o certifikátu, podpoře SPDY apod. Takto je možné otestovat i funkční SPDY na serveru, který funguje na Apache.

Dále je možný test funkčnosti na adrese, která se zadá do vyhledávače Google Chrome.

chrome://net-internals/#spdy

Pokud by došlo k chybě, je dost možné, že není SPDY v Chrome zapnutý. Toho se dá docílit, když přes terminál pustíme Chrome s parametrem [29] [30]

- -use-spdy=npn

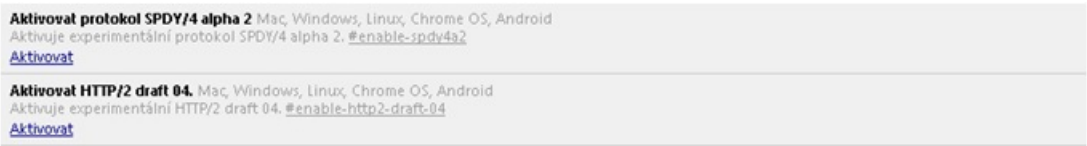
2.3 Google Chrome

Prohlížeč Google Chrome má jednu velmi užitečnou funkci. Obyčejný uživatel pravděpodobně ani netuší, co se v tomto prohlížeči skrývá. Funkce nazývaná Flags, neboli „Vlaječky“ umožňuje spoustu rozšířitelných možností. Jsou to jisté experimenty, ale vše, co se zde uživatel nastaví, je na jeho vlastní riziko. Nastavením se dá ovlivnit chování a výkon prohlížeče. Google na oficiální stránce uvedl:

„*Pozor, tyto experimenty mohou skončit všelijak. UPOZORNĚNÍ Tyto experimentální funkce se mohou kdykoli změnit, zhroutit nebo zmizet. Nemůžeme vůbec zaručit, co se po zapnutí těchto experimentů stane. Prohlížeč může smazat veškeré vaše údaje nebo neočekávanými způsoby narušit vaše zabezpečení či soukromí. Experimenty, které zapnete, budou k dispozici všem uživatelům prohlížeče. Buďte prosím obezřetní.*“ [31]

Pro nastavení se do políčka pro vyhledávání napíše *chrome://flags/*

Jsou zde různé možnosti pro vykreslování, akceleraci, ladění, apod. Pro mě byla důležitá dvě nastavení viz obr. 2.3



Obr. 2.3: Aktivace SPDY

Tato možnost slouží pro zkoušku SPDY. Poté všechny požadavky jdou přes proxy od Google, se kterou probíhá komunikace přes SPDY.

Výbornou pomůckou je také plugin do prohlížeče Google Chrome s názvem SPDY indicator, který je ke stažení na internetovém obchodu Google. Tento plugin indikuje protokol SPDY na Internetu. Pokud je zrovna použita komunikace přes SPDY, objeví se indikátor viz obr. 2.4 [32]

Obr. 2.4: Plugin SPDY

2.4 Vytvoření stránky pro testování

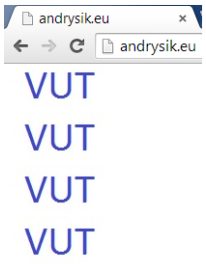
Abych mohla otestovat rychlost a rozdíly v přenosu mezi různými verzemi HTTP a protokolem SPDY, tak jsem do adresáře *var/www* určeného pro obsah webových stránek nahrála jednoduchou webovou stránku, která obsahuje více menších objektů (obrázků) ve formátu JPEG. Počet objektů se rovná 26 a jsou totožné, jen má každý objekt jiný název, aby server i klient bral každý objekt jako samostatný prvek a aby na každý objekt byl požadavek zvlášť viz obr. 2.5.

Stránka je vytvořena jednoduchým značkovacím jazykem HTML a nejsou použity kaskádové styly (CSS) a ani PHP či jiné skripty. Takže se jedná o statickou webovou stránku. Ve zdrojovém kódu jsem použila tag *<img src="vut.jpg"alt="vut">* (cesta k obrázku a popis, který by byl vidět, pokud by bylo vypnuto načítání obrázků) a mezeru mezi obrázky *
*. Vše ohraničené párovým tagem *<html> </html>* (začátek a konec html dokumentu).

Více objektů jsem zvolila hlavně z důvodu toho, abych mohla otestovat rozdíl mezi HTTP 1.0 a HTTP 1.1. Jak jsem již výše uvedla, hlavním rozdílem mezi HTTP 1.0 a HTTP 1.1 je ten, že HTTP 1.0 nepodporuje udržované spojení. Proto pro každý obrázek je potřeba navázat nové spojení a to je náročnější na čas a na přenosovou kapacitu sítě. Pokud je tedy na stránce 26 objektů, tak je potřeba navázat 27 spojení (26 spojení pro objekty a 1 spojení pro stránku). U HTTP 1.1, HTTPS a i u protokolu SPDY je již navázáno trvalé spojení se serverem, čímž se šetří čas a přenosová kapacita sítě. Spojení může ukončit jak klient, tak server a to hlavičkou *Connection: close* zařazenou do žádosti/odpovědi.

Webovou stránku jsem dala na server *andrysik.eu*, který jsem měla k dispozici pro tento projekt a využívá pro HTTP klasický port 80 (*http://andrysik.eu*) a pro HTTPS (*https://andrysik.eu*) klasický port 443. Vše jsem testovala „zvenčí“ a ne

z localhostu, aby měření bylo reálnější, jelikož by se mohla projevit určitá latence připojení.



Obr. 2.5: Jednoduchá webová stránka

2.5 Wireshark

Velmi užitečný nástroj, který slouží pro analýzu celé síťové komunikace. Je následovníkem programu Ethereal, jenž byl vyvinut v roce 1977. Zachytává pakety ze sítě, dekóduje a zobrazuje je. Velkou výhodou je, že je to open-source, takže je poskytován zdarma. Umí také číst data ze souboru, pracuje jak v promiskuitním (program zachytí kompletní komunikaci, která se objeví na sdíleném médiu) i nepromiskuitním módu, obsahuje spoustu filtrů a spoustu dalších funkcí. Existuje jak pro Linux, Windows, tak pro další operační systémy.

Pro zachytávání si vybereme zdroj, na kterém chceme zachytávat a také to, jestli chceme zachytávat s promiskuitním, či nepromiskuitním režimu. Vše je velmi pěkně graficky upraveno pro lepší přehlednost a pakety jsou očíslovány. Nalezneme zde všechny důležité informace o zdrojové a cílové IP adrese, protokolu, navázání a ukončení spojení, čas a spoustu dalšího. Můžeme také odposlouchat spoustu informací při nešifrované komunikaci. Všechny výsledky se dají uložit do souboru pro pozdější analýzu. Z výsledných hodnot lze utvořit i grafy.

V mém projektu jsem zachytávala na určeném médiu a v nepomiskuitním módu. Zachytávala jsem komunikaci jak pro HTTP 1.0, HTTP 1.1, HTTPS, tak pro protokol SPDY. U obou se zachytávané výsledky lišily, což je logické. Hlavním rozdílem bylo, že se při spojení s HTTP 1.0 znovu navazovalo spojení a u vyšších verzí protokolu je již spojení stálé, takže vše proběhlo podle teoretických předpokladů. Při HTTP jsem si zvolila jako filtr HTTP (TCP), abych viděla jen tyto pakety a při šifrovaném spojení jsem si odfiltrovala TCP (TLS). Z těchto výsledků jsem utvořila názorné grafy viz dále.

Jednotlivé pakety jsou ve Wiresharku opravdu podrobně rozebrané. V první řadě jsem se zaměřila na hlavičky protokolů HTTP, které jsou v části Hypertext Transfer

Protokol. Zde jsou všechny informace o přenášené stránce, hostovi, klientovi, kódování, spojení apod. U protokolu TCP jsem si všímala hlavně navázání a ukončení spojení, jenž jsou označovány jako SYN, ACK, FIN. Více u šifrované komunikace vidět není. [33]

2.5.1 Navázání spojení - trojcestný handshaking

Klient odesílá na server data – SYN a server odešle odpověď SYN a ACK klientovi a klient opět odešle odpověď serveru ACK. Pokud vše proběhne v pořádku, je navázáno spojení.

2.5.2 Ukončení spojení

Je na stejném principu jako navázání spojení – klient odešle data s FIN, server odešle odpověď ACK klientovi, poté odešle FIN a klient odpoví ACK. Pokud vše proběhlo v pořádku, spojení je ukončeno. Dále je možné také sledovat, na jakých portech probíhá komunikace. V mém případě HTTP na klasickém portu 80 a HTTPS na portu 443. [34]

3 MĚŘENÍ

3.1 Testy rychlostí

Tímto nastavením jsem si „připravila půdu“ pro hlavní cíl této bakalářské práce a tím je otestování rychlosti a chování protokolů HTTP 1.0, HTTP 1.1, HTTPS a SPDY.

Pro otestování protokolu HTTP 1.0 bylo potřeba nastavit Apache tak, aby využíval tento protokol. Nastavení je velmi jednoduché a provede se v souboru *httpd.conf*. Jak jsem již výše uvedla, soubor slouží pro dodatečnou konfiguraci serveru.

Do tohoto souboru se uvedou následující tři řádky:

SetEnv downgrade-1.0, SetEnv force-response-1.0, SetEnv nokeepalive

SetEnv: Nastaví argument uvedený za tímto příkazem.
downgrade-1.0: Vynutí žádosti, aby byly považovány za žádost o HTTP 1.0.
force-response-1.0: Opět vynucení žádosti o HTTP 1.0 - pro klienty HTTP 1.0, kteří mají problém s odpovědí HTTP 1.1.
nokeepalive: Pokud by bylo nastavené trvalé spojení při požadavcích, tak jej tento příkaz zakáže. [35]

Dále jsem zakázala SSL v souboru *default-ssl* tak, že jsem hodnotu *SSLEngine* nastavila na *off*.

Modul SPDY jsem také zakázala nastavením hodnoty *SpdyEnabled* na *off* a to v souboru *spdy.conf*, který se nachází v adresáři *Mods-available*.

Po všech změnách jsem restartovala Apache, aby se změny aktivovaly.
sudo service apache2 restart

Po restartování jsem využila dvou nástrojů pro otestování rychlosti a to webový nástroj *http://tools.pingdom.com/fpt/*, který testuje rychlost přes server v New Yorku a druhým nástrojem byl plugin do internetového prohlížeče Firefox s názvem *Firebug*, který testuje načítání stránky přímo. Ve všech měřeních probíhalo 27 žádostí a byla použita stejná testovaná webová stránka. Měnil se pouze nastavený protokol na serveru.

3.1.1 Test HTTP 1.0

V tomto testování byl celkový čas načítání 1,73 s viz obr. 3.1. S tímto časem se tento protokol zařadil na poslední místo, jelikož je to nejdelší naměřený čas.

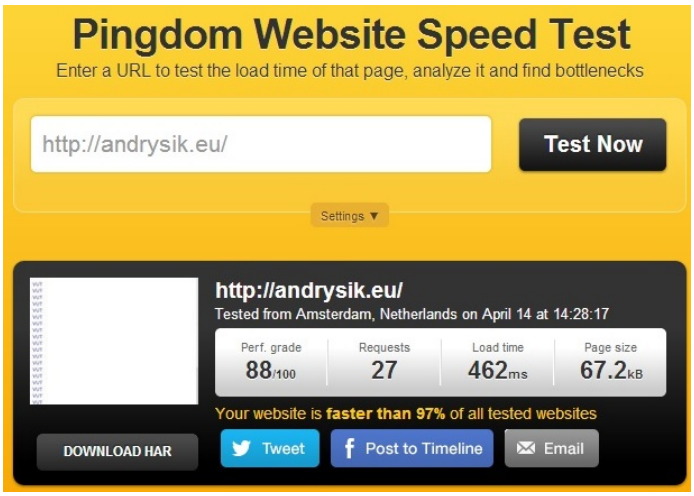
Obr. 3.1: Web test HTTP 1.0

Pomocí pluginu v prohlížeči byla již celková rychlost vyšší, jelikož se testuje načítání stránky přímo. V tomto případě byl celkový čas roven 19 ms viz obr. 3.2. Opět jde o jeden z nejvyšších časů kvůli stálému navazování spojení.

Obr. 3.2: Firebug test HTTP 1.0

3.1.2 Test HTTP 1.1

Dalším testovaným protokolem byl HTTP 1.1, který se přes webové testování dostal na čas 462 ms viz obr. 3.3. Je tedy podstatně rychlejší, než protokol HTTP 1.0.



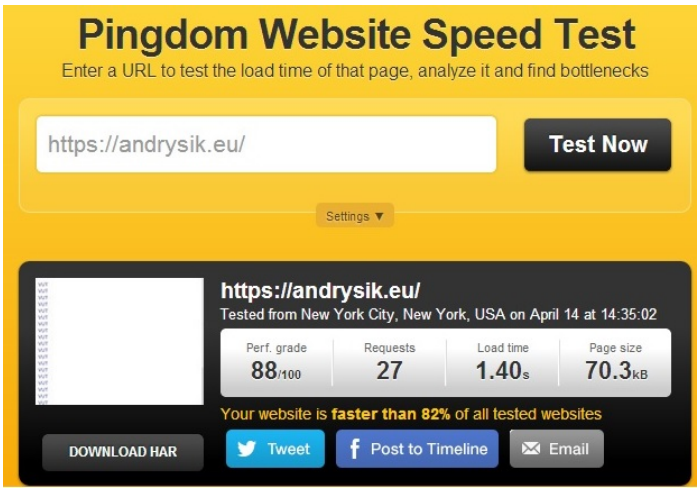
Obr. 3.3: Web test HTTP 1.1

Při druhém testu v pluginu byl také rychlejší, než protokol HTTP 1.0 a čas načítání byl 6 ms viz obr. 3.4. Zde již dochází k trvalému spojení.

Obr. 3.4: Firebug test HTTP 1.1

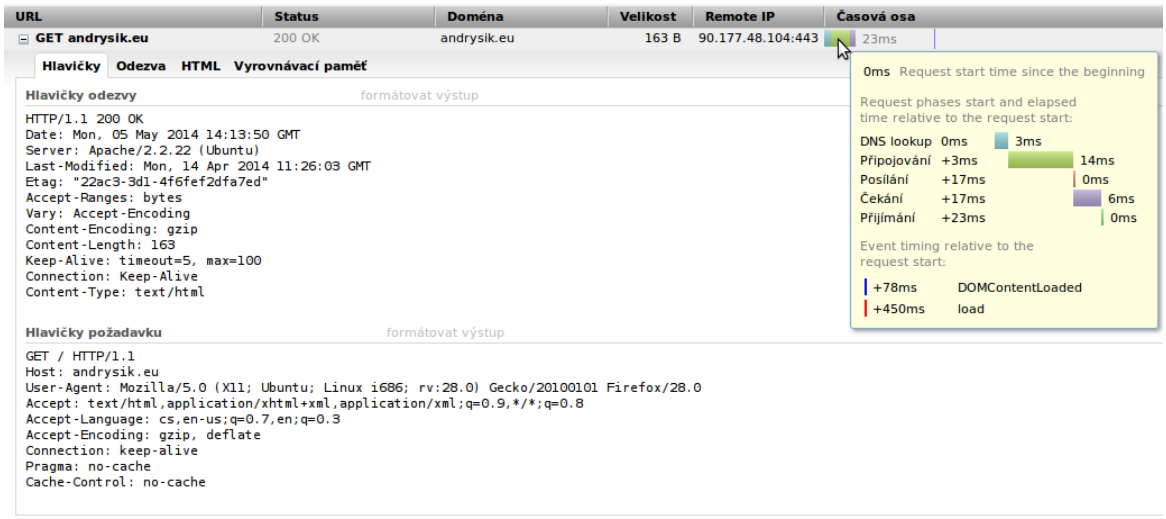
3.1.3 Test HTTPS

Čas 1,40 s byl naměřenem při testování protokolu HTTPS viz obr. 3.5. Je vyšší oproti HTTP 1.1 a je to způsobeno tím, že se jedná o protkol využívající šifrování a je nutné, aby si servery mezi sebou vyměnily klíče.



Obr. 3.5: Web test HTTPS

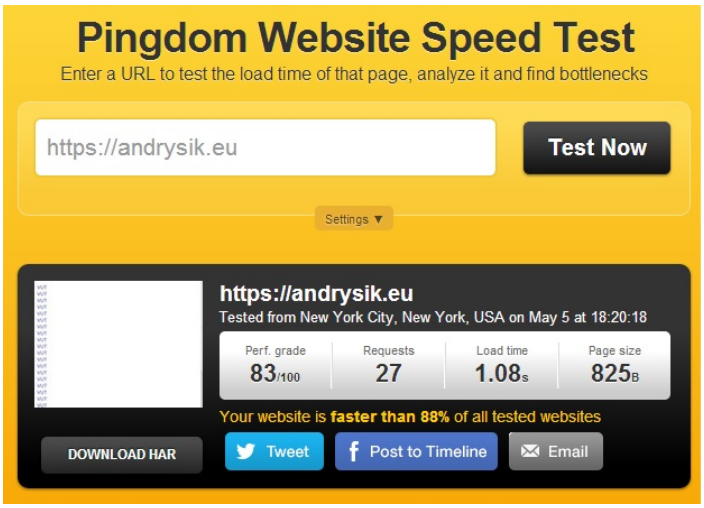
Pomocí pluginu bylo načítání rychlejší než u webového testování. Čas načítání byl 23 ms viz obr. 3.6, což byl nejvyšší naměřený čas v pluginu. Za delší načítání opět může šiforvání.



Obr. 3.6: Firebug test HTTPS

3.1.4 Test SPDY

Poměrně překvapivý výsledek byl při webovém i pluginovém testování. Při webovém testování protokol dosáhl rychlosti 1,08 s viz obr. 3.7 a tento čas je nižší než u HTTPS, i když SPDY taktéž využívá šifrování a také byl nižší, než u HTTP 1.0.



Obr. 3.7: Web test SPDY

Pluginové testování ukázalo čas 18 ms viz obr. 3.8, který byl taktéž nižší než HTTPS a HTTP 1.0.

Obr. 3.8: Firebug test SPDY

3.2 Grafy zatížení linky

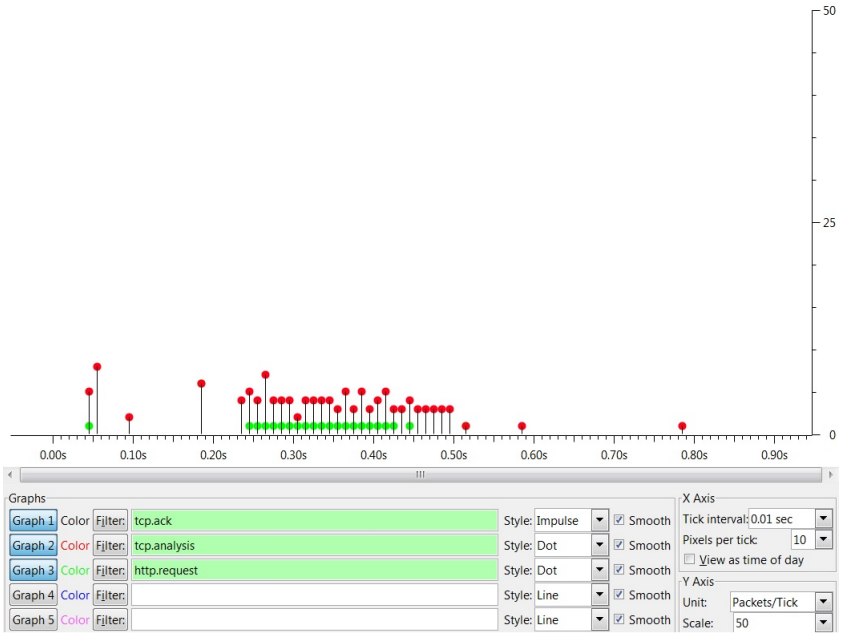
3.2.1 HTTP 1.0

Graf zatížení linky protokolem HTTP 1.0 viz obr. 3.9 názorně ukazuje, že není vyřizováno více požadavků v jednom spojení, jako je tomu u HTTP 1.1. Takže je zde více navazování a ukončování spojení. Přenos protokolem HTTP 1.0 má za následek větší zatížení linky a to souvisí jak s dobou trvání přenosu (kvůli navazování spojení), tak s velikostí paketů, která je větší.

Obr. 3.9: Graf zatížení linky HTTP 1.0

3.2.2 HTTP 1.1

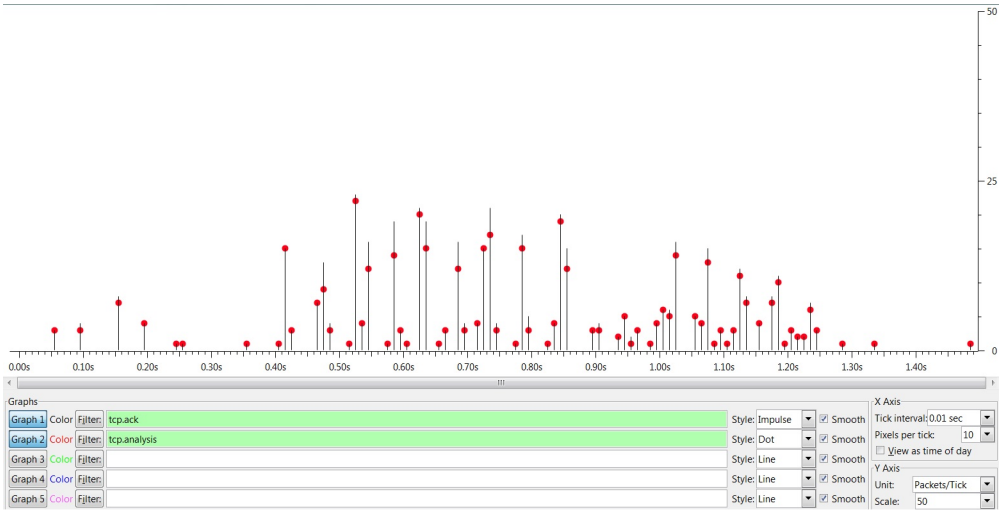
Graf zatížení linky protokolem HTTP 1.1 viz obr. 3.10 ukazuje, že je v jednom spojení vyřizováno více požadavků a to má vliv na celkové zatížení linky a velikost paketů. Oproti HTTP 1.0 klesla velikost paketů, doba vyřízení všech požadavků a také klesl počet všech požadavků.



Obr. 3.10: Graf zatížení linky HTTP 1.1

3.2.3 HTTPS

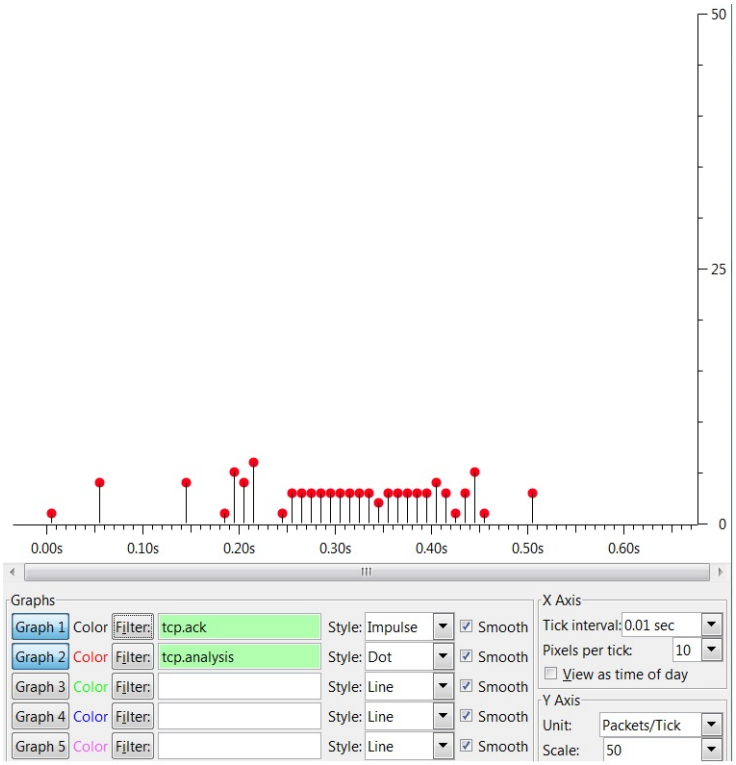
Graf zatížení linky protokolem HTTPS viz obr. 3.11 ukazuje šifrovaný přenos. Zatížení linky je vyšší, ale je to opět z důvodu, jak jsem již zmínila, vyměňování klíčů a šifrování.



Obr. 3.11: Graf zatížení linky HTTPS

3.2.4 SPDY

Graf zatížení linky protokolem SPDY viz obr. 3.12 ukazuje, že oproti přenosu HTTPS je zatížení linky mnohem menší, je zde méně žádostí, menší pakety a celková doba přenosu se taktéž zkrátila.



Obr. 3.12: Graf zatížení linky SPDY

4 ZAJÍMAVOSTI

4.1 PageSpeed modul

K optimalizaci webových stránek patří také page speed modul. Pokud se bavíme o celkovém zrychlování načítání webových stránek, měli bychom využít také page speed. Google vypustil do světa mod_pagespeed pro linux a jedná se o modul pro celkovou optimalizaci stránek a to má následně vliv na komunikaci mezi serverem a prohlížečem.

Pokud jsou webové stránky obsahově větší, takže mají rozsáhlý kód, tím více času je potřeba pro načtení. V dnešní době se již nehraje až tak na to, aby byly stránky jednoduché a měly co nejmenší velikost obrázků a reklam. Je to spíše naopak. Většinou se s problémem pomalého načítání setkáme u mobilních zařízení, jako jsou tablety, smartphony apod. Připojení je pomalé, hlavně pokud je to připojení od operátora. O to více, pokud je aktivní FUP. V tomto případě jsme rádi za každou setinu sekundy, kdy se načítání zrychlí.

Proto byl vynalezen mod_pagespeed, který udělá celkovou optimalizaci za nás. Obsahuje určité množství (přes 40) filtrů, které se o tuto optimalizaci starají – odstraňují nadbytečné mezery z kódu, odřádkování. Dokáží i vložit CSS či JS do HTML stránky, starají se o cache paměť apod. [36]

4.2 SPDY a Android

Vzhledem k tomu, že prohlížeč Google Chrome od společnosti Google je vyvíjen i pro operační systém Android, který je ve většině přenosných zařízení (tablety, smartphony), tak je možné i zde využít protokolu SPDY. Podporu lze nastavit stejně, jako pro PC verzi - pomocí *chrome://flags*.

5 ZÁVĚR

V první části bakalářské práce jsem se věnovala teoretickému popisu problematiky protokolu HTTP, přičemž tyto nabyté teoretické poznatky jsem zužitkovala v praktické části.

Zaměřila jsem se na praktické porovnání HTTP 1.0, HTTP 1.1, HTTPS a SPDY. Mým cílem bylo navrhnout experiment, který bude demonstrovat rozdíly v chování a vlastnostech těchto protokolů (rozdíly v rychlosti, bezpečnosti).

Vytvořenou webovou stránku jsem následně testovala z „venku“, jelikož si myslím, že nejlepší optimalizace webových stránek je taková, když je test prováděn ze strany uživatele a ne ze strany správce na localhostu, kde se neprojeví vnější vlivy. Poté můžeme lépe zhodnotit stav webového serveru a webových stránek a můžeme případně vše dooptimalizovat.

Při testování jsem nevyužívala cache paměť, jelikož by výsledky nebyly relevantní.

Experiment se mi podařilo zrealizovat bez větších potíží a výsledky všech měření odpovídají teoretickým předpokladům. Pokud se zaměřím na měření rychlosti načítání webové stránky (pomocí internetové stránky a pluginu), je podle výsledků podtržen teoretický předpoklad.

Nejdéle se načítala stránka běžící na protokolu HTTP 1.0 kvůli neudržovanému spojení. Déle také trvalo načítání v šifrovaném HTTPS z důvodu výměny veřejných a privátních klíčů, které si vyměňují klient a server. V zájmu bezpečnosti se dá větší časová prodleva pochopit. Ještě časově lépe vyšel nový šifrovaný protokol SPDY, který byl skoro o cca 20% rychlejší, než HTTPS. Toto je poměrně slušná časová úspora. Nejrychlejší byl přenos protkolem HTTP 1.1, ale protokol je nešifrovaný, proto bych dala přednost protkolu SPDY.

Pokud se zaměřím na grafy, které byly vytvořeny ze zachycených dat v programu Wireshark, tak jsou zde také vidět podstatné rozdíly v přenosech při použití různých verzí protokolu.

V grafu ptokolu HTTP 1.0 je názorně vidět, že oproti HTTP 1.1 zde probíhá více požadavků s větší velikostí paketů a delší dobou přenosu. Z toho důvodu je linka protokolem HTTP 1.0 zatěžována podstatně více, než s protokolem HTTP 1.1, kde je požadavků méně, velikost paketů se změnšila a i celkový čas přenosu klesl, takže je zatížení linky podstatně menší. U protokolu HTTPS je komunikace kvůli šifrování paketově a časově náročnější, ale opět protokol SPDY se projevil jako rychlý a nezatěžuje linku tolik, jako HTTPS. Je to hlavně kvůli multiplexovanému spojení - může při jednom TCP spojení přenést více dotazů zároveň a využívá komprimaci hlaviček. Wireshark neprozradil u HTTPS a SPDY kvůli šifrování žádná data, pouze u HTTP by bylo možné z paketů dostat přenášené informace.

Jak jsem se již zmínila, tak pro celkovou optimalizaci a úpravu webových stránek se může také využít modul Pagespeed, který může také přispět k rychlejšímu přenosu.

Práce pro mě byla velmi přínosná, jelikož jsem nahlédla do historie, postupného vývoje a celkové problematiky protokolů pro přenos hypertextových dokumentů. Lze s nadsázkou říci, že bez vzniku protokolu HTTP by dnešní požitek z Internetu nebyl takový, jaký jej známe.

Při testování mě hodnoty protokolu SPDY překvapily. Jedná se o poměrně bezpečný, rychlý a spolehlivý protokol. Při návštěvě dnešního Internetu se počítá každá ušetřená milisekunda a hlavně v přenosných zařízeních, kterými je zaplněn trh s elektronikou. SPDY může mít i dobrý vliv (i když ne úplně zásadní) na výdrž baterie.

Myslím si, že protokol SPDY bude revoluční a doufám, že se brzy rozšíří a bude stejně úspěšný, jako protokol HTTP 1.1.

LITERATURA

[1] SATRAPA, Pavel. *Gopher server*. Technická univerzita v Liberci: Ústav Nových technologií a aplikované informatiky [online]. [cit. 2013-11-26]. Dostupné z WWW: <http://www.nti.tul.cz/~satrapa/docs/iserver/gopher.html>.

[2] KOSEK, Jiří. *Základy protokolu HTTP*. Kosek [online]. © 1999 [cit. 2013-11-26]. Dostupné z WWW: <http://www.kosek.cz/clanky/iweb/05.html>.

[3] LAMPA, Petr. *Protokol HTTP 1.1*. Fakulta informačních technologií [online]. 26.9.2002 [cit. 2013-11-26]. Dostupné z WWW: <http://www.fit.vutbr.cz/~lampa/WWW/http11.html.cs>.

[4] KUČERA, František. *Protokol HTTP*. Zdrojak [online]. 2011 [cit. 2013-12-28]. Dostupné z WWW: <http://www.zdrojak.cz/clanky/protokol-http/>.

[5] PUŽMANOVÁ, Rita. *TCP/IP v kostce*. 2. upr. a rozš. vyd. České Budějovice: Kopp, 2009, 619 s. ISBN 978-80-7232-388-3.

[6] KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5., aktualiz. vyd. Brno: Computer Press, 2008, 488 s. ISBN 978-80-251-2236-5.

[7] NOVOTNÝ, Petr. *HTTP protokol*. [online]. [cit. 2013-12-28]. Dostupné z WWW: <http://www.kiv.zcu.cz/~ledvina/vyuka/PSI/Presentace/HTTP-novotny-prezentace.pdf>.

[8] *Hypertext Transfer Protocol - HTTP/1.1*. Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://tools.ietf.org/search/rfc2616>.

[9] SYNODINOS, Dio. *HTTP 1.2 Released with Improved Support for Hierarchies and Text-Menu Interfaces*. Infoq [online]. 2011 [cit. 2013-12-28]. Dostupné z WWW: <http://www.infoq.com/news/2011/04/http-1.2-released>.

[10] *HTTP Authentication: Basic and Digest Access Authentication*. Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://tools.ietf.org/html/rfc2617>.

[11] *PEP*. W3 [online]. 1996 [cit. 2013-12-28]. Dostupné z WWW: <http://www.w3.org/TR/WD-http-pep-951122>.

[12] *HTTP Over TLS*. Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://tools.ietf.org/html/rfc2818>.

[13] VÁCLAVÍK, Lukáš. *Google o čtvrtinu zrychlí surfování. Jeho protokol SPDY bude součástí HTTP/2.* Cnews [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://www.cnews.cz/google-o-ctvrtinu-zrychli-surfovani-jeho-protokol-spdy-bude-soucasti-http2>.

[14] *SPDY Protocol.* Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00>.

[15] MALÝ, Martin. *Bude web rychlejší s protokolem SPDY?.* Zdrojak [online]. 2011 [cit. 2013-12-28]. Dostupné z WWW: <http://www.zdrojak.cz/clanky/bude-web-rychlejsi-s-protokolem-spdy/>.

[16] KŘÍŽ, Lukáš. *IETF dokončuje specifikaci HTTP 2.0.* Businessit [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://www.businessit.cz/cz/ietf-dokoncuje-specifikaci-http-2-0.php>.

[17] JELÍNEK, Lukáš. *HTTP 2.0 může mít šifrování jako výchozí.* Linuxexpres [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://www.linuxexpres.cz/novinky/http-2-0-muze-mit-sifrovani-jako-vychozi>.

[18] *Uniform Resource Identifier (URI): Generic Syntax.* Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://tools.ietf.org/html/rfc3986>.

[19] *Uniform Resource Locators (URL).* Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://tools.ietf.org/html/rfc1738>.

[20] *URN Syntax.* Ietf [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://www.ietf.org/rfc/rfc2141.txt>.

[21] *REST - moderní alternativa programování webových aplikací.* Softec [online]. 2013 [cit. 2013-12-28]. Dostupné z WWW: <http://www.softec.cz/aktuality/rest-moderni-alternativa-programovani-webovych-aplikaci.html>.

[22] PAŽOUREK, Tomáš. *Webové služby v architektuře REST na platformě .NET.* Brno, 2012,[online]. [cit. 2013-12-28]. Dostupné z WWW: <http://is.muni.cz/th/359464/fi_b/thesis.pdf>. Bakalářská práce. Masarykova univerzita.

[23] THON, Miloslav. *Použití programu WinProxy: pro připojení domácí sítě k internetu.* Český Krumlov, 2005,[online]. [cit. 2014-05-27]. Dostupné z WWW: <http://www.milost.wz.cz/download/winproxy.pdf>.

[24] Kára, Michal. *Tuneluji, tuneluješ, tunelujeme: přesměrování portů*. Root.cz [online]. 2003 [cit. 2014-05-27]. Dostupné z WWW: <http://www.root.cz/clanky/tuneluji-tunelujes-tunelujeme-presmerovani-portu/#ic=serial-box&icc=text-title>.

[25] *Co je UBUNTU*. Ubuntu.cz [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://www.ubuntu.cz/cojeubuntu>.

[26] *Chmod*. Wiki.ubuntu.cz [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://wiki.ubuntu.cz/chmod>.

[27] Cailliau, Robert. *Tim Berners-Lee, Robert Cailliau, and the World Wide Web*. Livinginternet.com [online]. 1995 [cit. 2014-05-27]. Dostupné z WWW: <http://www.livinginternet.com/w/wi_lee.htm#dev>.

[28] Dočekal, Michal. *Správa linuxového serveru: Úvod do konfigurace Apache*. Linuxexpres.cz [online]. 2011 [cit. 2014-05-27]. Dostupné z WWW: <http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-uvod-do-konfigurace-apache>.

[29] Štrauch, Adam. *Zrychlete komunikaci s Apache pomocí Google SPDY*. Root.cz [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://www.root.cz/clanky/zrychlete-komunikaci-s-apache-pomoci-google-spdy/>.

[30] *Mod-spdy: Apache SPDY module*. Code.google.com [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <https://code.google.com/p/mod-spdy/wiki/GettingStarted>.

[31] *Chcete rychlejší, chytřejší a lepší Google Chrome? - 2. díl*. Pcworld.cz [online]. 2013 [cit. 2014-05-27]. Dostupné z WWW: <http://pcworld.cz/software/chcete-rychlejsi-chytrejsi-a-lepsi-google-chrome-2-dil-45618>.

[32] Čížek, Jakub. *Mocné vlaječky prohlížeče Chrome*. Zive.cz [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://www.zive.cz/clanky/mocne-vlajecky-prohlizece-chrome/sc-3-a-166306/default.aspx>.

[33] Vavřina, Josef. *Wireshark - Kontrolujeme provoz na síti*. Magazin.stahuj.centrum.cz [online]. 2009 [cit. 2014-05-27]. Dostupné z WWW: <http://magazin.stahuj.centrum.cz/wireshark-kontrolujeme-provoz-na-siti/>.

[34] Bouška, Petr. *TCP/IP - navázání a ukončení spojení*. Samuraj-cz.com [online]. 2007 [cit. 2014-05-27]. Dostupné z WWW: <http://www.samuraj-cz.com/clanek/tcpip-navazani-a-ukonceni-spojeni/>.

[35] *Environment Variables in Apache*. Httpd.apache.org [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://httpd.apache.org/docs/2.2/env.html>.

[36] Štrauch, Adam. *Google mod_pagespeed: rychlejší komunikace s webovým serverem*. Root.cz [online]. 2012 [cit. 2014-05-27]. Dostupné z WWW: <http://www.root.cz/clanky/google-mod-pagespeed-rychlejsi-komunikace-s-webovym-serverem/>.