České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Richard Wimberský**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Protokol komentování webových článků - klient**

Pokyny pro vypracování:

Mnoho redakčních serverů publikujících články na webu umožňuje také tyto články komentovat. Webový protokol http ale není příliš vhodný pro komunikaci v reálném čase. Seznamte se problematikou komentování webových článků. Porovnejte existující řešení komentování a na základě jejich zhodnocení navrhněte nový protokol komentování. Protokol slouží pro definování komunikace mezi serverem a jeho klienty. Pro moderování diskuze bude protokol nabízet možnost autentifikace a autorizace uživatele. Spolu s protokolem navrhněte klientský program. Navržený program implementujte a otestujte jeho spolupráci se serverem. Výsledky vyhodnoťte.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Božena Mannová, Ph.D.

Platnost zadání: do konce zimního semestru 2013/2014

L.S.

doc. Ing. Miroslav Šnorek, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 18. 2. 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

bakalářská práce

Protokol komentování webových článků – klient

Richard Wimberský

vedoucí práce: Ing. Božena Mannová, Ph.D.

studijní program: Softwarové technologie a management, bakalářský
obor: Softwarové inženýrství

Praha, 2015

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 22. 5. 2015

_____

Poděkování

Rád bych poděkoval svému bratrovi Tomášovi Wimberskému, který se mnou studoval, motivoval mě, absolvoval všechny předměty a nakonec i spolpracoval na vypracování bakalářské práce praktikou Párového programování, která je jedním z hlavních pilířů Extrémního programování. V neposlední řadě bych chtěl poděkovat naší vedoucí práce paní Ing. Boženě Mannové, Ph.D., která nás vedla při vypracování této práce.

Abstrakt

Hlavním cílem této bakalářské práce je navrhnout vhodný protokol pro diskutování na internetu. Protokol vzniká jako zcela nový a samostatný projekt, určený k dalšímu rozvíjení. Důraz je kladen především na distribuovanost a decentralizaci přípojných bodů a úložišť diskusních příspěvků.

Součástí této práce je návrh a implementace prototypu klienta, který používá navržený komunikační protokol. Implementace je provedena v programovacím jazyku Python.

Abstract

The main aim of this bachelor thesis is design of suitable protocol to discuss on the internet. Protocol arises as a completely new and independent project and is intended to be further extended. Emphasis is placed especially on distribution and decentralization of connecting points and storages of discussion contributions.

Design and implementation of a client prototype using this communication protocol is included in this work. Implementation is done in the Python programming language.

Obsah

1. Úvod

1.1. Představení způsobu vypracování práce

Tato bakalářská práce koreluje s bakalářskou prací Tomáše Wimberského s názvem Protokol komentování webových článků – server. Samotný protokol musí být stejný jak pro klienta, tak i pro server, aby spolu mohli správně komunikovat. Protokol tedy musíme definovat oba stejně. Protokol jsme navrhli metodou Párového programování. Z tohoto důvodu je nutné, aby byly některé části obou bakalářských prací společné. Společně jsme pracovali na Úvodu, Definici pojmů, máme společnou Vizi celého projektu a společně jsme provedli rešerši existujících řešení. Částečně stejnou na nutné úrovni máme Analýzu požadavků. Požadavky, týkající se samotného klienta jsem analyzoval sám. Návrh řešení pak bylo potřeba vytvořit společně a k tomu jsme společně vytvořili použité obrázky. Samotnou implementaci prototypu klienta jsem pak vypracoval sám. Testování klienta se serverem jsme museli provést společně, samotného klienta jsem již mohl otestovat sám jednotkovými testy.

1.1.1. Kaskádové, Iterační a agilní programování

V jednotlivých kurzech studia jsme byli seznámeni s mnoha postupy vývoje softwaru. Jedním z nejstarších způsobů programování, je Kaskádové programování. Je tvořeno třemi částmi, které se vzájemně neprolínají a navazují na sebe. První část je návrh, druhou částí je samotná implementace, tedy naprogramování a třetí částí je nasazení či vydání softwaru. Tento způsob je vhodný pro školní účely, pro začínající programátory, kteří se pod vedením školitele mají naučit syntaxi daného programovacího jazyka a mají samostatně vytvořit implementaci nějakého jednoduššího návrhu. Je pak snadné ohodnotit každého jednotlivého studenta, jak přesně implementoval návrh zadaný vyučujícím. Zároveň studentovi nedává možnost spekulovat, či konzultovat smysl, či využitelnost nebo možnosti integrování daného návrhu a v důsledku toho návrh pozměnit.

V průběhu času se ukázalo, že kaskádový způsob vývoje softwaru není příliš dynamický a efektivní, velmi pozdě se odhalují chyby nebo nepřesnosti v návrhu a následné hledání a odstraňování chyb v implementaci je časově velmi náročné a i velmi pracné. V důsledku toho bylo vyvinuto a popsáno vícero postupů Iteračního programování, které umožňují agilní vývoj softwaru. Velmi aproximovaně

o těchto způsobech můžeme říci, že původní tři části: návrh, implementaci a nasazení se rozdělí do více podčástí, které spolu vzájemně interagují, a navíc přibyde testování. Metodika agilního softwaru, která nám s kolegou Tomášem Wimberským přišla nejzajímavější, je Extrémní programování a některé jeho praktiky jsme se rozhodli použít při vypracování svých bakalářských prací.

1.1.2. Párové programování

Extrémní programování je metodika agilního vývoje softwaru, jehož zrod se datuje k počátku devadesátých let dvacátého století a za jeho hlavní autory je všeobecně považována trojice Kent Beck, Ward Cunningham a Ron Jeffries. Extrémní programování zavádí 12 praktik a na zavedení a definování těchto praktik se podílelo mnoho odborníků. Proto těžko přisuzovat autorství jednomu člověku nebo nějaké menší skupině autorů. Extrémní programování je v současné době nejobsáhlejší a nejpřísnější definice postupu vývoje software.

Jednou z praktik Extrémního programování je Párové programování. Tato praktika spočívá v tom, že u jednoho počítače sedí dva vývojáři a ve stejnou dobu na stejném místě pracují na jednou na stejné věci. Píše pochopitelně pouze jeden. Aby oba byli spravedlivě a rovnoměrně vytíženi, v psaní se střídají. Ten, který zrovna nepíše, konzultuje a případně upravuje. Tato praktika velmi snižuje pravděpodobnost chyby z nepozornosti v implementaci a napomáhá lepšímu pochopení návrhu, který má být aktuálně implementován. U větších než dvoučlenných týmů se mění i sestavení párů a je vhodné, aby ve výsledném kódu nebylo poznat, kdo psal jakou část. Další z praktik Extrémního programování je častá Refaktorizace a je tedy obvyklé, že v každé verzi implementoval danou část kódu jiný programátor.

Z pochopitelných důvodů je zřejmé, že není příliš reálné vyzkoušet si párové programování v době výuky, kdy se studenti učí jednak algoritmizovat a jednak programovat v daném programovacím jazyce a je potřeba ohodnotit, jak se tyto disciplíny naučil každý jeden student.

Párové programování je rozumné až pro programátora, který má více pokročilé schopnosti v algoritmizaci a mírně pokročilé znalosti v syntaxi používaného programovacího jazyka. Tyto schopnosti a znalosti už v době psaní závěrečné bakalářské práce studenti mají a proto jsme se s kolegou Tomášem Wimberským rozhodli vyzkoušet si praktiku Párového programování pod vedením paní Ing. Boženy Mannové, Ph.D.

Bližší popis Extrémního programování je zpracovaný v příloze 10.1.

1.1.3. Výběr tématu

Abychom si mohli vyzkoušet pracovat společně a zároveň dodrželi konvenci, kdy některá část musí být samostatná, bylo potřeba vymyslet vhodné téma, které by tyto možnosti přirozeně umožnilo. Téma jsme si vymysleli sami. Navržení komunikačního protokolu, definování klienta a definování serveru tyto požadavky splňuje. Aby klient mohl komunikovat se serverem, musí komunikovat stejným protokolem. Definování tohoto protokolu musíme jednoznačně provést oba společně. Prototyp serveru i prototyp klienta jsme pak vytvořili každý zvlášť.

1.2. Cíle práce

Cílem této závěrečné bakalářské práce a práce Tomáše Wimberského s názvem Protokol komentování webových článků – server je definovat komunikační protokol mezi klientem a serverem. Definovat takový protokol, který bude zprostředkovávat distribuovaný systém, který nebude závislý na žádném poskytovateli. Takový systém, ve kterém budou existovat diskuse, dokud budou existovat komentující účastníci a žádná diskuse nebude moci být ukončena nebo smazána rozhodnutím jedné entity.

Individuálním cílem je definovat vlastnosti klienta a implementovat prototyp klienta.

Dalším společným cílem je pak vyzkoušet si v praxi některé z praktik Extrémního programování, zejména pak Párové programování. Zhodnotit způsob práce při párovém programování a sdělit své subjektivní pocity výhod a nevýhod Párového programování.

2. Vymezení pojmu

2.1. Diskuse

Jako diskusi míníme offline internetovou komunikaci, rozhovor několika osob nad určitým tématem, jehož cílem je shromáždit postřehy a argumenty k danému tématu, například webovému článku.

Offline komunikace nevyžaduje, aby uživatel bezprostředně reagoval na zprávu, která mu přijde, ale může si jí přečíst a zareagovat, až když se s počítačem připojí k internetu.

2.2. Protokol

Přesně určená konvence, která specifikuje komunikaci a přenos dat mezi dvěma koncovými body.

2.3. Uzel

Uzel je libovolný koncový účastník komunikace. V našem případě je to buď klient, nebo server.

2.4. Server

Koncový bod komunikace, který je přístupný pro jiné koncové body komunikace. A to jak klienty, tak i servery. Server poskytuje klientům a jiným serverům jednak seznam serverů a jednak diskusní příspěvky.

2.5. Superuzel

Superuzel je uzel, který je serverem.

2.6. Klient

Koncový bod komunikace, pomocí kterého si diskutující stahuje příspěvky v diskusích a pomocí kterého posílá své příspěvky.

Z pohledu konceptu bychom ještě mohli definovat pojmy Peer, Seeder a Leecher. Tyto pojmy se běžně používají v P2P komunikaci. Peer je libovolný účastník komunikace, Leecher je příjemce dat, který data neodesílá a Seeder je posílač dat, který má všechna data z nějakého logického celku.

V našem případě postačí definovat závislosti pomocí pojmů klient a server. Protože se diskuse budou velmi často měnit, budou v nich přibývat komentáře a celý systém bude distribuovaný, mohl by server jen hádat, jestli je pro danou diskusi Seederem nebo jen Peerem. Klient je z tohoto pohledu považován pouze za Leechera.

3. Motivace a vize

3.1. Motivace

Diskutování na internetu je problém, který dosud nebyl uspokojivě vyřešen tak, aby respektoval svobodu slova a původní myšlenku internetu. Diskutování nesmí mít žádnou centrální složku, musí fungovat, i když jsou některé její části mimo provoz, nesmí nikomu bránit v přidávání nových příspěvků a zakládání dalších diskusí. Každá diskuse musí být přístupná a snadno vyhledatelná. Diskutování musí být zdarma, provozované nejlépe na počítačích samotných diskutujících a ne ve vlastnictví korporací, které by tak měly možnost diskusi ukončit nebo změnit. Mou motivací je tento nedostatek napravit.

3.2. Vize

Vytvořit protokol pro komentování webových článků takový, který bude svobodný a umožní existenci diskuse minimálně do té doby, dokud budou mít komentující o diskusi zájem. Protokol, který bude umožňovat, aby celá diskuse nepatřila jedné osobě a nemohla o osudu diskuse rozhodovat jen jedna osoba. Vytvořit takový protokol, který bude zaručovat decentralizovaný systém komentování, aby nebylo možné vypnutím jednoho serveru zrušit celou diskusi. Vytvořit takový protokol, který bude jednoduše rozšiřitelný o nové smysluplné funkcionality.

Vytvořit takový systém, který bude komentujícím umožňovat komentovat a přispívat do diskusí s rozmyslem bez ohledu na připojení k internetu v době, kdy právě píše svůj komentář. Uživatel bude moci napsat komentář, i když není zrovna připojen

k internetu, například z důvodu horšího pokrytí na venkově a až jeho zařízení, na kterém příspěvek napsal, bude připojeno k internetu, dojde k synchronizaci. Uživatel nepřijde o svůj text, kvůli nestabilnímu připojení k internetu.

4. Existující řešení

4.1. E-mailová konference

Jedním z běžně provozovaných způsobů diskutování na internetu je diskuse prostřednictvím elektronické pošty. Nový účastník se přihlásí do diskusní skupiny a nové příspěvky odesílá na určitou adresu, ze které jsou poté poštovním serverem rozeslány všem ostatním přihlášeným účastníkům.

Hlavní výhodou je jednoduchost, neboť pro komunikaci lze použít libovolného poštovního klienta, který bývá dostupný na nejrůznějších zařízeních, bez jakýchkoli dodatečných úprav. Problematické je ale přihlašování do skupiny příjemců nebo zakládání nových diskusí, které není součástí běžné elektronické pošty a musí ho řešit každý server jinou cestou. Další problém se objevuje v rozsáhlejších diskusích. Aby byl zachován kontext diskuse, posílají se ve zprávě i předchozí příspěvky, které později zabírají neúměrně mnoho místa. Při vypuštění některých příspěvků by pak nově příchozí diskutující ztráceli možnost číst a reagovat na starší příspěvky. Zprávy navíc nejsou nijak číslované, a pokud dojde k nedoručení nebo výpadku zprávy některým účastníkům, je to téměř neodhalitelné.

E-mail používá k rozesílání zpráv protokol SMTP, k příjmu zpráv starší POP3 nebo novější IMAP. Ačkoli je e-mail jedním z nejrozšířenějších způsobů komunikace a diskutování na internetu, pro globální diskutování se vůbec nehodí.

4.2. Webové diskusní systémy

Vhodnějším kandidátem by mohla být internetová služba WWW. WWW používá protokol HTTP k přenosu dokumentů, obvykle ve formátu HTML nebo XHTML. Tyto dokumenty nemusejí být statické soubory, ale mohou být generované na straně serveru. Součástí formátu HTML jsou i formulářové prvky, s jejichž pomocí se dají serveru prohlížečem zaslat data, tedy například i nějaká zpráva. Formuláře jsou obecné a parametry zprávy jsou čistě v režii webového serveru. To může být výhodné

například v rozšiřitelnosti diskuzního systému - na webu najdeme nejrozmanitější systémy jako návštěvní knihy, diskusní fóra nebo otázky a odpovědi, ale také nevýhodné například tím, že se nedá spolehnout, že nějaký parametr bude diskuzní systém obsahovat. HTTP servery navíc po přenesení dat ukončují spojení. Je tedy nutné neustále obnovovat celou webovou stránku, což vede ke zbytečné zátěži sítě. Snížení zátěže je možné provést technologiemi AJAX[1] či WebSocket[2].

Dalším problémem je, že webový prohlížeč se připojuje pouze k jednomu serveru, je tedy potřeba spouštět nový proces prohlížeče pro každou stránku s diskusí zvlášť, což s přibývajícím množstvím diskusí přestává být pohodlné a uživatelé se ke starším diskusím už nevracejí.

Jedna z možností, jak tomuto čelit, je použití RSS čtečky nebo upozorňování na email, přibyde-li do diskuse nový příspěvek. Ovšem ne každý diskusní systém tuto možnost nabízí. Dalším řešením je využití služeb třetích stran jako je Disqus[3] nebo Facebook Comments plugin[4], které na nově příchozí zprávy upozorňují. Vložení na stránky ale vyžaduje zásah provozovatele stránek.

Hlavním problémem je ovšem architektura WWW. Data se nacházejí pouze na serveru (ať už přímo provozovatele webu nebo na serveru služby třetí strany), je tedy relativně snadné jej přetížit, webové příspěvky úmyslně měnit a cenzurovat, případně pokud se provozovatel rozhodne dále stránky neprovozovat, celá diskuse přestane existovat.

4.3. Usenet

Je jedním z nejstarších diskusních systémů na internetu, který se ještě běžně používá (je přibližně o deset let starší než služba WWW). Diskuse v něm jsou logicky rozděleny do hierarchických skupin podle předmětu, např. *sci.math* je skupina diskutující o matematice. Čtečka příspěvků označovaných v Usenetu jako „news“ je dnes součástí většiny e-mailových klientů, a většina serverů poskytuje i webová rozhraní. Jeho hlavní nevýhodou je právě členění do hierarchických skupin,

[1] Samotný AJAX tento problém nevyřeší, pořád je potřeba obnovovat stránku, ale protokol HTTP verze 2 umožňuje neukončit spojení hned po přenesení dat, čehož by se dalo využít pro komunikaci v reálném čase.
[2] WebSocket umožňuje serveru komunikovat s klientem přes full-duplexní TCP spojení.
[3] https://disqus.com
[4] https://developers.facebook.com/docs/plugins/comments

což potvrzuje skupina *alt.** (skupina bez předmětu), která je nejobsáhlejší kategorií v celém Usenetu. Každá skupina (kromě *alt.**) je navíc centralizovaně kontrolována a cenzurována.

Usenet je distribuovaný mezi mnoha, neustále se měnícími konglomeracemi serverů, které si mezi sebou vyměňují své příspěvky pomocí záplavového algoritmu. Připomíná tak peer-to-peer komunikaci, zatímco komunikace mezi čtečkou a serverem je typickým příkladem klient-server architektury.

Server se ovšem také nemusí synchronizovat vůbec a klient tak nemá záruku, že uvidí všechno, co se v dané skupině nachází i na jiných serverech. Dále Usenet neřeší autorizaci a autentizaci uživatele a neumožňuje příspěvky editovat.

5. Analýza požadavků

5.1. Sběr požadavků

Zde se pokusíme vyjmenovat a vypsat požadavky v řeči běžného uživatele internetu, který není vývojářem a ani programátorem a užívá aplikace s grafickým uživatelským rozhraním.

1. Necentralizovaný distribuovaný systém.

2. Možnost archivovat a distribuovat komentáře jakýmkoliv účastníkem diskuse.

3. Možnost vytvořit komentář i při nestabilním připojení k internetu.

4. Jednoduché použití, snadná konfigurace.

5. Jednoznačná identifikace uživatele, který přispívá do diskuse

6. Uživatel bude mít možnost své komentáře upravovat.

5.2. Analýza sebraných požadavků

1. Protokol bude zprostředkovávat distribuovaný způsob archivace a předávání dat v podobě komentářů v prostém textu. Žádný účastník komunikace nemusí mít nikdy všechna data. Jednotlivé uzly komunikace si boudou data synchronizovat. Tato

synchronizace bude probíhat na uvážení každého uzlu. U distribuovaného řešení není možné zajistit, aby všechny uzly měly vždy najednou všechna data. Synchronizace má vždy nějaké zpoždění, závislé například na vytížení daného hardwaru a sítě.

2. Je nutné rozlišit, o jakého účastníka diskuse jde. U klienta, který je připojený například přes mobilní telefon s velikostí úložiště řádově stovky MB až jednotky GB a s nestabilním a často se měnícím připojením k internetu, u takového by bylo na škodu posílat mu vždy všechny příspěvky. Naopak u uzlu, který je na páteřní síti a má velké datové úložiště, u takového je vhodné aby uložil co nejvíce dat.

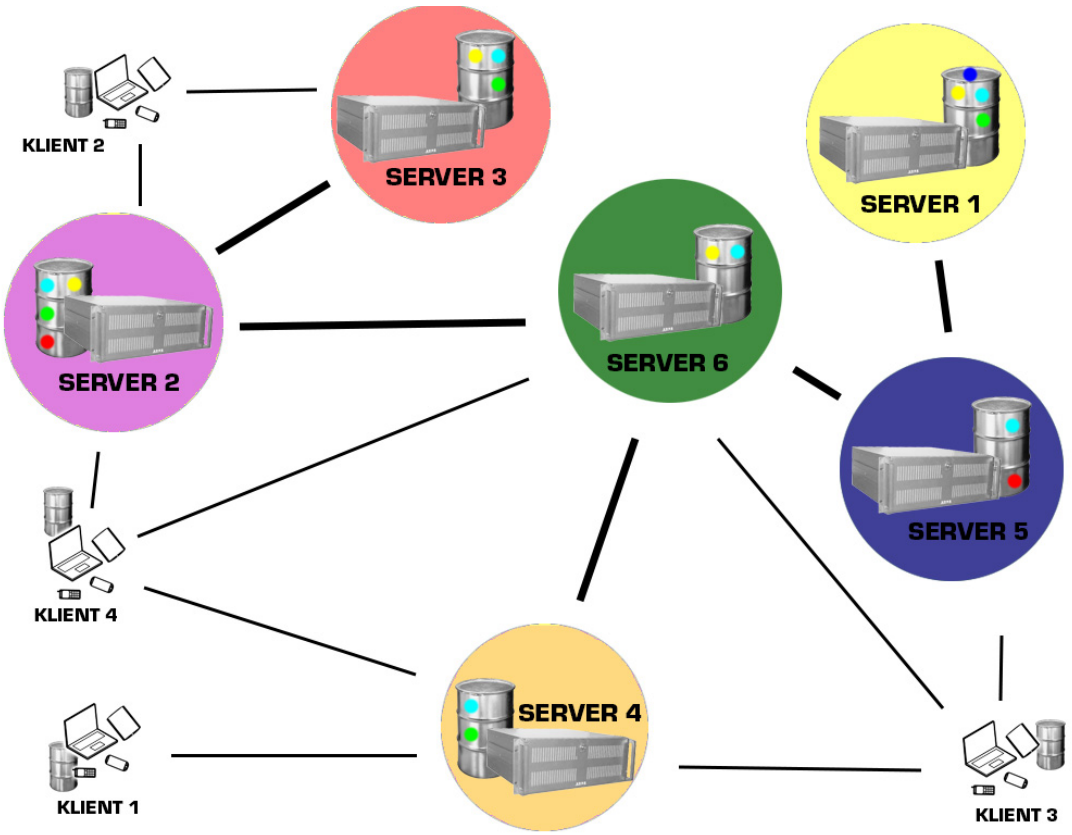
3. Uživatel bude moci číst komentáře, které má ve svém lokálním úložišti bez ohledu na to, zda je připojen k internetu nebo není. Na tyto komentáře bude moci odpovědět, okomentovat je, a dále se nebude muset starat o jejich odeslání na další příslušný uzel či více uzlů. K synchronizaci dojde bez nutnosti aktivní činnosti uživatele, jakmile to konektivita umožní.

4. Bude se jednat o program, který bude zprostředkovávat komentování pouze v prostém textu. Možnosti základního nastavení proto budou bohatě postačovat jen adresa a port serveru, ke kterému se má klient připojovat. Případně seznam adres a portů serverů na které se má klient připojovat. To bude pro prvotní připojení zcela stačit, protože klient od každého serveru dostane jeho seznam adres a portů přístupných serverů.

5. Autentizace uživatele. Bude kontrolována při přijímání nového komentáře a to jak na straně serveru tak na straně klienta. Klient si navíc zkontroluje autentizaci v případě editace stávajícího komentáře, zda uživatel chce editovat opravdu svůj komentář. Číst komentáře si může kdokoliv s tím, že každý uzel si bude u každého právě příchozího příspěvku kontrolovat autentizaci autora a to i u patchů ke komentářům.

6. Autorizace bude prováděna po přijetí každého komentáře před jeho uložením.Neautorizované komentáře se nebudou ukládat. Stejně tak úprava komentářů bude probíhat pomocí patchů. Tedy novým komentářem k opravovanému komentáři, který bude mít náležitý příznak, který označí, jaký komentář upravuje. Pro takovýto patch komentář budou platit stejná autentizační a autorizační pravidla jako pro jakýkoliv jiný komentář.

6. Návrh řešení



Obrázek 1: Základní spojení

Na obrázku 1 je znázorněno základní spojení s naznačením kdo se na koho může připojit a kdo má jaká data. Klient se může připojovat pouze na servery. Server se mohou připojovat na jiné servery. Klient se nemůže připojit na jiného klienta a server se nemůže připojit na žádného klienta. Spojení klienta se serverem je značeno tenkou čarou, spojení serveru se serverem znázorňuje čára tlustá. Sudy značí úložiště. Barevné tečky na sudech znázorňují, jaká různá data má server ve svém úložišti. Každý klient má své vlastní úložiště s diskusemi, potažmo s příspěvky.

6.1. Server

Server zastává funkci distributora příspěvků a úložiště příspěvků. K samotným příspěvkům nic nepřidává, nevytváří je. Server může přijmout příspěvek jak od

klienta, tak od jiného serveru. U každého příspěvku, který mu přijde, ověří, zda odpovídá podpis. Příspěvek si uloží pouze pokud je podpis v pořádku. Server si kromě diskusí ukládá seznam serverů, na které se kdy připojil, aby s nimi mohl spojení navázat znovu. Dále si ukládá u připojených klientů, o jaké diskuse mají zájem a nové příchozí příspěvky jim rovnou přeposílá. Na server není kladen žádný nárok na identifikaci či autentizaci. Jediným identifikátorem serveru je jeho IP adresa a port, případně doménové jméno a port.

6.2. Klient

Klient má hlavní funkci čtení a vytváření komentářů. Klient se připojuje pouze na servery. Počet serverů, na které se klient může připojit, není z pohledu konceptu nijak omezen. Klient navazuje a udržuje spojení se serverem.

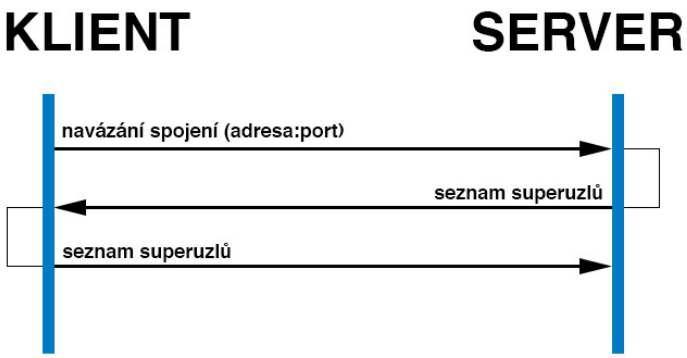
Při vytváření nového komentáře klient vytvoří otisk příspěvku, tento otisk podepíše a pak vše odešle na server. Volitelná položka komentáře je uživatelův veřejný klíč. Pokud klient ví, že na server, na který komentář posílá, již někdy uživatelův veřejný klíč poslal, nemusí ho do komentáře přidávat.

Při obdržení komentáře ze serveru si nejprve zkontroluje, zda podepsaný otisk je otiskem daného komentáře. Pokud ano, uloží ho do svého úložiště včetně podpisu a umožní uživateli zobrazit. Pokud podepsaný otisk neodpovídá komentáři, klient komentář neuloží, ale nikomu již tuto skutečnost neoznamuje. Klient si ukládá seznam serverů, na který byl kdy připojený.

6.3. Spojení klienta a serveru

Spojení inicializuje klient. Spojení se realizuje protokolem TCP. Po úspěšném navázání spojení pošle server klientovi svůj seznam serverů. Tedy serverů, na které byl již někdy úspěšně připojen. Klient si aktualizuje svůj seznam serverů a pošle ho serveru. Pro snazší použití a sdílení konfigurace je vhodné, aby tento seznam byl ukládán v podobě textového souboru, ve kterém bude každá řádka odpovídat jednomu serveru. Stejný soubor by pak používal i server.

Toto spojení zůstane navázané a bude připravené pro posílání zpráv od klienta na server i v opačném směru ze serveru na klienta. Toto navázání spojení je znázorněno na obrázku 2.



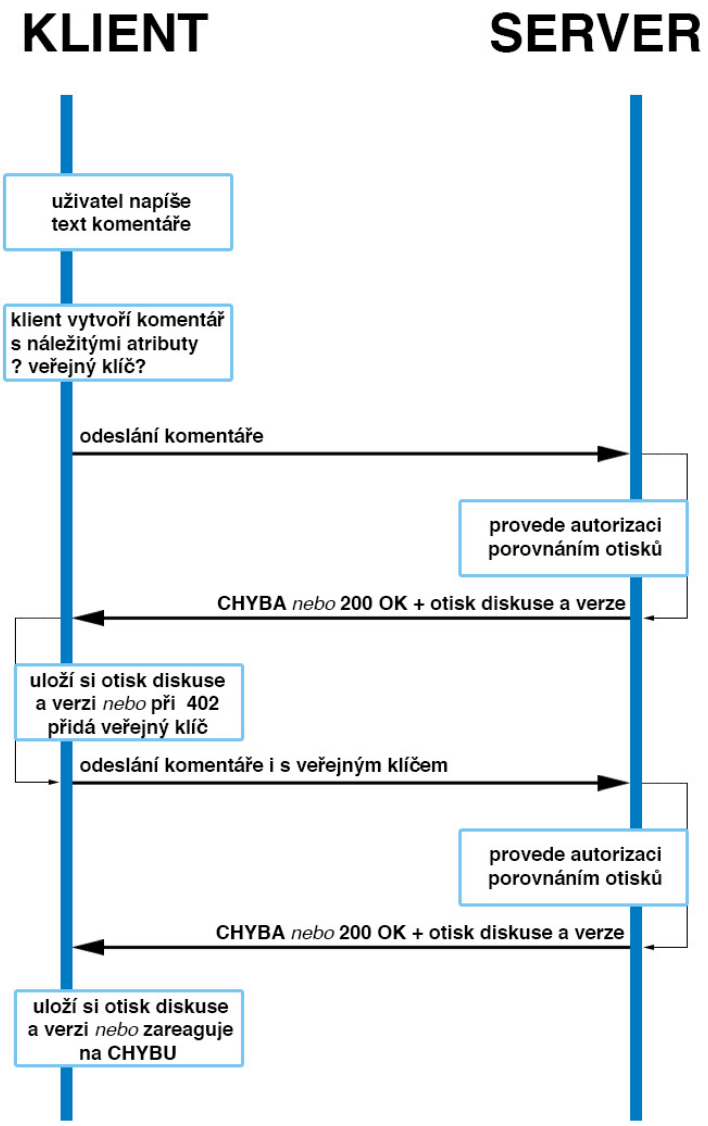
Obrázek 2: Navázání spojení klienta se serverem

Nyní je klient připraven na posílání komentářů na server a na přijímání zpráv a komentářů ze serveru.

Odesílání komentáře, které je znázorněno na obrázku 3, probíhá následovně:

- Uživatel prostřednictvím klienta napíše svůj komentář, přesněji pouze jeho text.
- Klient vytvoří komentář z uživatelem napsaného textu a z atributů, kterými jsou: ID diskuse, číslo patche, ID komentáře, na který uživatel reaguje, podpis, datum. Volitelným atributem je uživatelův veřejný klíč. Pokud klient vyhodnotí, že danému serveru již někdy v minulosti uživatelův veřejný klíč posílal, přidávat ho do atributů nemusí. Klíč je však docela veliký a proto je vhodné ho posílat jen pokud je to nutné, aby se ušetřil síťový provoz.
- Takto vytvořený komentář odešle klient na server.
- Server náležitým způsobem provede autorizaci. Pokud proběhne úspěšně, uloží si komentář do diskuse, diskusi zamkne, přidělí ID komentáře, vytvoří otisk diskuse a zvýší číslo verze.
- Proběhla-li autorizace úspěšně, pošle klientovi kód 200 OK a příslušný nový otisk diskuse a nové číslo verze diskuse. Zároveň tento server odešle nový autorizovaný příspěvek všem účastníkům diskuse, se kterými má spojení.
- Pokud autorizace neproběhla v pořádku, pak pošle server klientovi odpovídající chybový kód, z jakého důvodu se autorizace nezdařila.
- Je-li chybovým kódem 402 Key Required, pak klient k tomu samému komentáři přidá do atributu uživatelův veřejný klíč a znovu ho odešle na

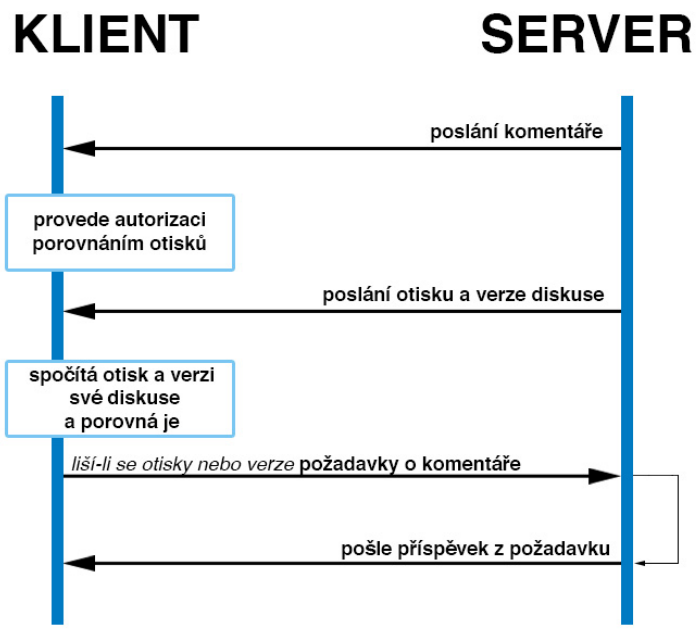
server. Nyní server pokračuje autorizací a stejným způsobem jako při předchozím přijetí příspěvku. Tento proces je znázorněn na obrázku 3.



Obrázek 3: Klient posílá komentář

Přijímání komentáře klientem je velmi podobné, jako příjem serverem, ale je o něco jednodušší. Klient neposílá zpátky žádné chybové kódy. Že komentář přijde ze serveru na klienta v pořádku a nepozměněn je zajištěno protokolem TCP, kterým

spolu klient a server komunikují. Navíc pošle server klientovi po novém komentáři ještě aktuální otisk diskuse a aktuální číslo verze diskuse. Klient si stejným způsobem jako server autorizuje komentář, tedy porovnáním otisků komentáře, ale jen pro své potřeby. Klient si stejně jako server spočítá nový otisk diskuse a novou verzi diskuse s nově přijatým komentářem a porovná ho se zprávou, kterou obdržel od serveru. Pokud se otisky nebo verze liší, pak si klient vyžádá celou diskusi. Schematicky je tento postup vyjádřen v obrázku 4.



Obrázek 4: Klient přijímá komentář

Klient si může libovolně zažádat o seznam příspěvků v diskusi. Systém verzování diskuse musí být dodržený takový, aby číslo verze vždy odpovídalo počtu příspěvků v diskusi a to včetně patchů, protože patch je nový plnohodnotný příspěvek. Klient tedy podle obdržené verze od serveru ihned zjistí, kolik příspěvků mu chybí a ví, o které si má zažádat.

Klient si může zažádat o libovolný příspěvek jakýkoliv server, ke kterému je aktuálně připojený a který diskusí disponuje. Tímto způsobem lze velmi jednoduše zrekonstruovat celou diskusi a překlenout tak případné chyby či vypnutí nějakých serverů.

6.4. Spojení serveru se serverem

Zde bychom mohli narazit na konflikt terminologií. Z pohledu našeho konceptu je pojem server definován v článku 6.1. V TCP, který je zdokumentován v IETF[5] RFC[6] 793[7] je ale vždy popisováno jen spojení mezi klientem a serverem. Proto jsme si pro naše potřeby a lepší porozumění zavedli pojem superuzel v článku 2.3.

Spojení mezi superuzly je nutné, aby mohl být celý systém distribuovaný a abychom překlenuli chyby či vypnutí některých superuzlů. Tímto způsobem navíc bude každá diskuse uložena ve stejnou dobu na různých úložištích a na různých místech současně. Zjednodušeně se tedy dá hovořit o záloze často se měnících dat.

Spojení mezi dvěma superuzly je realizováno pomocí TCP. Každý superuzel je pro TCP serverem, může se tedy na něj připojit kdokoliv. Který ze superuzlů se stane z pohledu TCP klientem a inicializuje spojení je jedno. V tomto konceptu může dojít k propojení dvou stejných superuzlů dvakrát. Takové druhé spojení by se mohlo zdát jako redundantní. Každé z takových dvou TCP spojení mezi dvěma stejnými superuzly může být navázáno přes jiné směrovače a proto je tato redundance vhodná pro zálohování spojení dvou stejných superuzlů.

Po navázání spojení, si superuzly vymění své seznamy superuzlů, podobně jako v případě spojení klienta a serveru. Od této doby se udržuje TCP spojení navázané a z pohledu našeho konceptu již na sebe superuzly vidí a není potřeba nadále rozlišovat, kdo je TCP klient a kdo je TCP server.

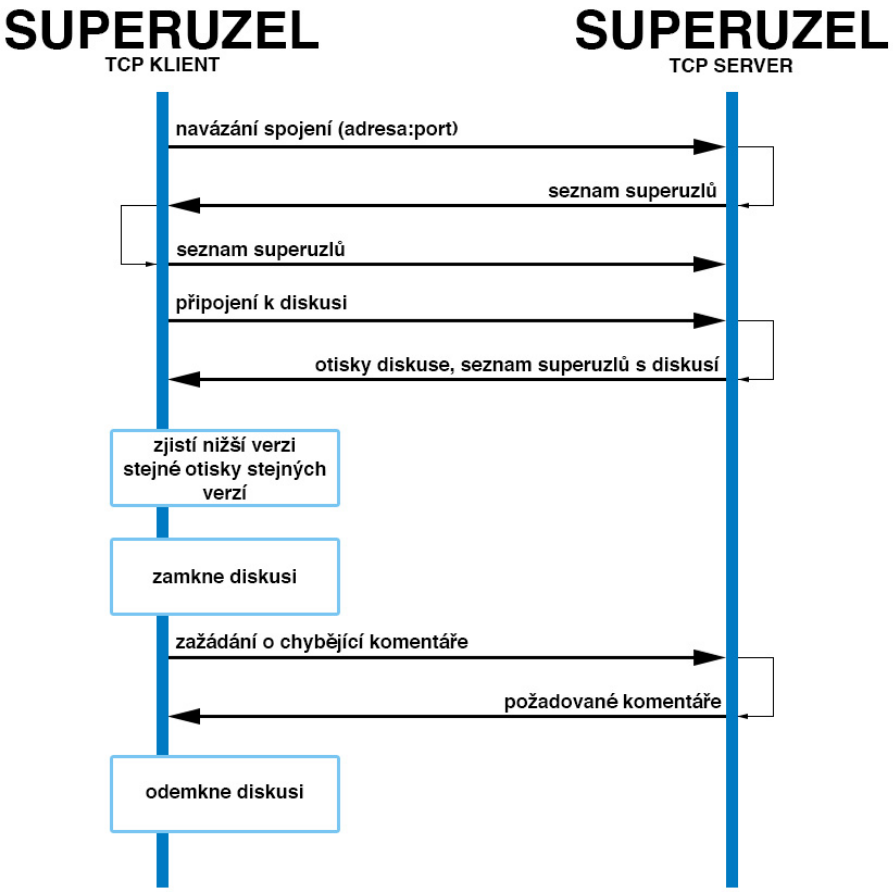
6.4.1. Doplnění diskuse

Připojení do diskuse probíhá tak, že si superuzel pošle požadavek o připojení k diskusi a druhá strana mu odpoví chybovým kódem 404 Not Found, pokud diskusi nemá. Pokud diskusi má, pošle žádajícímu superuzlu otisky diskuse a seznam superuzlů, kteří jsou do diskuse připojeni. Pořadí otisku odpovídá číslu verze diskuse a číslo verze odpovídá počtu komentářů v diskusi. Nejsnazší případ, který může v tuto chvíli nastat, je situace, kdy mají oba superuzly stejnou verzi diskuse a u poslední verze

[5] https://www.ietf.org
[6] https://www.ietf.org/rfc
[7] http://www.rfc-base.org/txt/rfc-793.txt

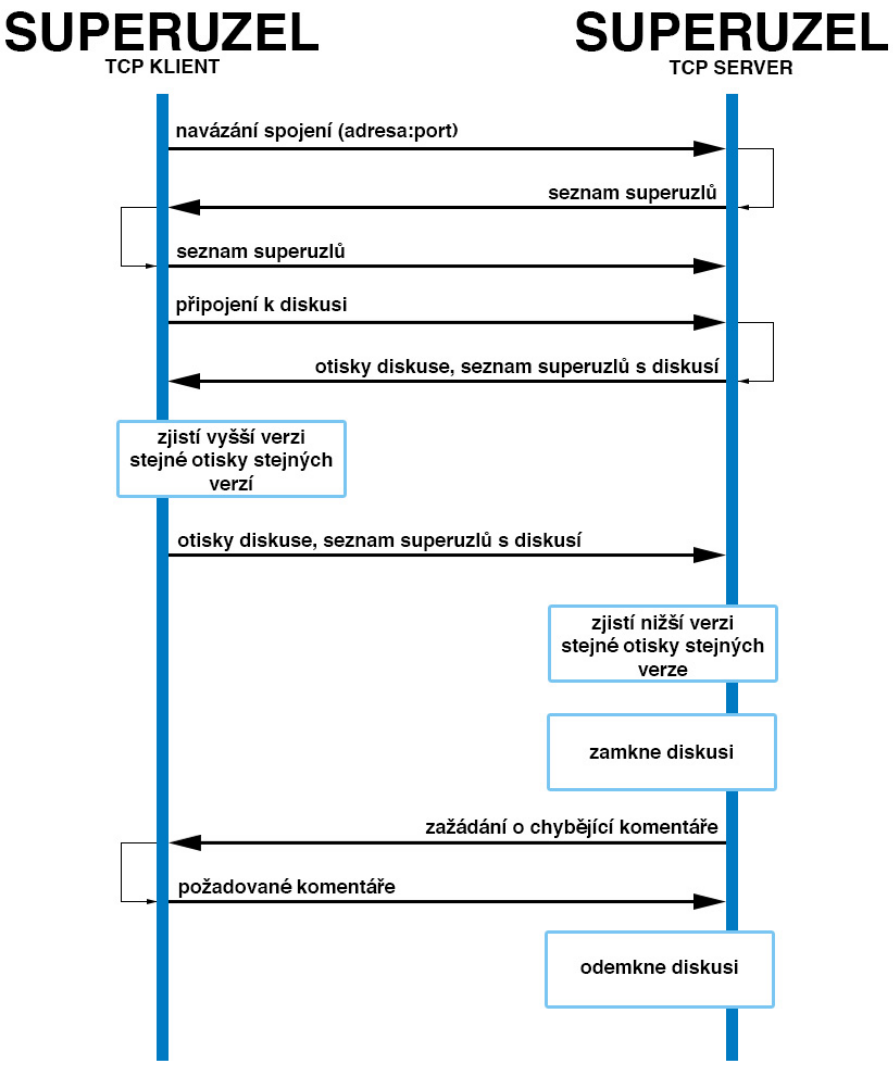
stejný otisk. Pro zjednodušení budeme přepokládat, že shodují-li poslední otisky, shodují se i všechny otisky předchozích verzí.

Pokud se verze liší, pozná to jako první dotazující se superuzel. Má-li dotazující se superuzel nižší verzi diskuse, ověří, zda je otisk jeho poslední verze stejný, jako otisk odpovídající verze superuzlu, kterého se dotazoval. Shodují-li se odpovídající otisky, dotazující superuzel zamkne svou diskusi a od ostatní superuzlů si nechá poslat příspěvky, které mu chybí tak, aby svou diskusi doplnil a synchronizoval. Tento postup zažádání o verzi a doplnění musí opakovat, dokud se verze a otisky neshodují. Poté svou diskusi odemkne. Postup je znázorněn na obrázku 5.



Obrázek 5: Spojení serveru se serverem - doplnění diskuse 1

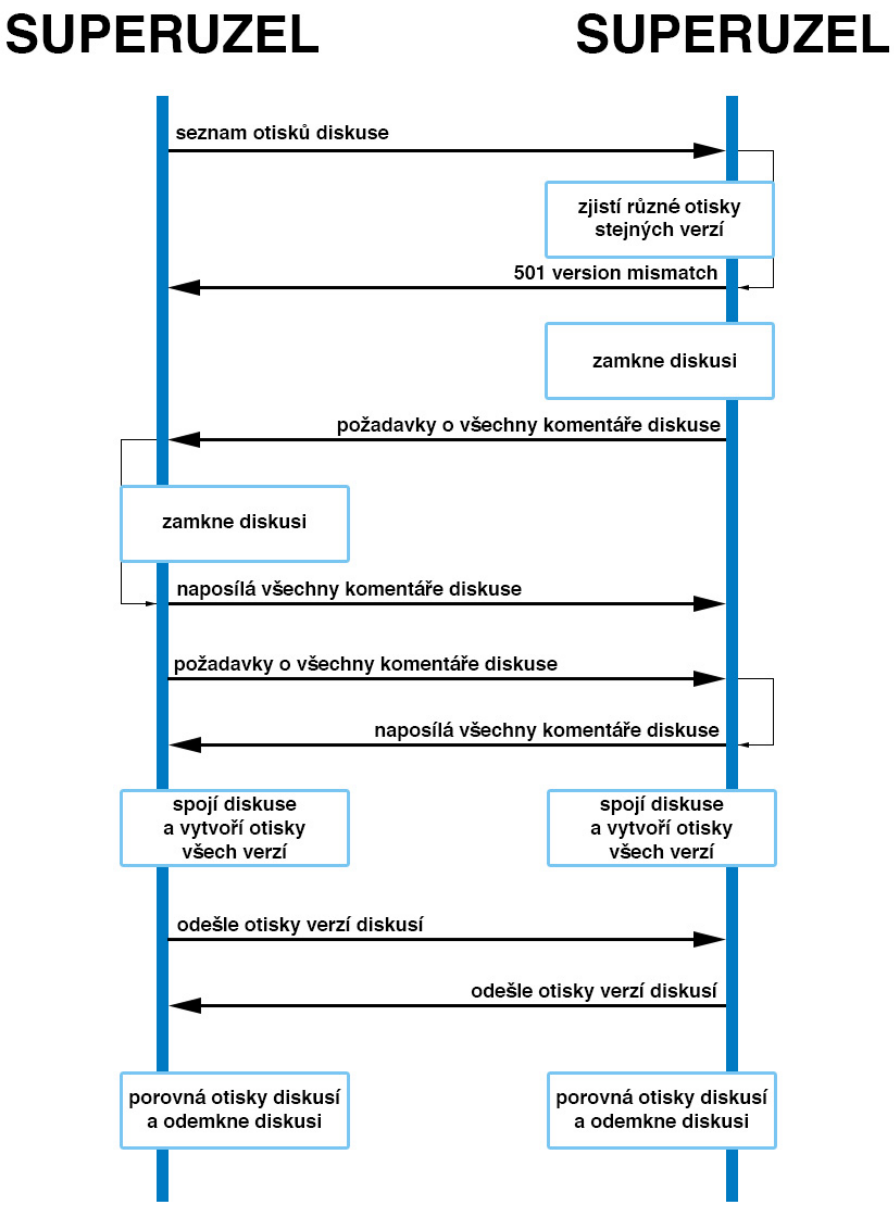
Druhý možný případ nastane v momentě, kdy dotazující se superuzel má novější verzi diskuse než dotazovaný superuzel a odpovídající verze diskusí mají stejné otisky. V tomto momentě pošle dotazující se uzel svůj seznam otisků a následný postup je stejný, jako v předchozím odstavci, s tím, že zamyká a doplňuje diskusi dotazovaný superuzel. Tato situace je znázorněna na obrázku 6.



Obrázek 6: Spojení serveru se serverem - doplnění diskuse 2

6.4.2. Spojení diskusí

Složitější situace nastane v momentě, kdy se odpovídající otisky neshodují. Tuto skutečnost zjistí jako první opět dotazující se superuzel. Dotazovanému uzlu pošle chybový kód 501 Version Mismatch. Oba superuzly zamknou své diskuse. Dotazující se uzel pošle své otisky diskusí a naposílá dotazovanému superuzlu postupně komentáře z celé své diskuse. Dotazovaný uzel také pošle každý komentář ze své diskuse dotazujícímu se uzlu. Poté oba superuzly každý zvlášť obě diskuse spojí. Komentáře budou číslovat podle času. K celé diskusi, taktéž vytvoří odpovídající otisky verzí diskusí. Tyto otisky si vzájemně zkontrolují. Vzájemná kontrola otisků nových diskusí je jen formalitou, protože vznikají ze stejných komentářů a stejným postupem. Poté oba superuzly danou diskusi odemknou. Spojení diskuse je vyobrazeno na obrázku 7.



Obrázek 7: Spojení diskusí

Každý superuzel si udržuje u každé své diskuse seznam, jací klienti a které superuzly, se kterými má aktuálně navázané spojení, sledují tuto samou diskusi. Pokud superuzlu přijde komentář do dané diskuse a superuzel ho úspěšně autorizuje, rozešle tento komentář všem uzlům z tohoto udržovaného seznamu, kromě odesílatele tohoto

komentáře. Obdrží-li superuzel duplicitní komentář, který již ve své diskusi má, bude ho ignorovat a nebude ho rozesílat dále.

6.5. Autentizace, autorizace a podepisování komentářů

Celý systém chceme udržovat distribuovaný. Proto je potřeba, aby se distribuovaně řešily i autentizace, autorizace a podepisování komentářů. Použití nějakého externího systému pro autentizaci, jako je například Kerberos, by sice bylo příjemné, ale při jeho výpadku nebo zrušení by nebylo možné vytvářet nebo upravovat komentáře. Nabízela by se tedy možnost použít více různých autentizačních systémů, kde by při výpadku nebo zrušení jednoho z nich mohli komentovat alespoň uživatelé, autentizovaní jiným systémem. To ale není vhodné řešení, navíc by si uživatelé museli vyřešit registraci na jiném serveru a to jim může zbytečně komplikovat situaci. Potřebujeme tedy řešit autentizaci, autorizaci a podepisování komentářů někde uvnitř našeho distribuovaného systému a pro uživatele co nejjednodušeji.

Autentizaci, autorizaci i podepisování bude řešit asymetrické šifrování, tedy použití soukromého a veřejného klíče s algoritmem RSA[8]. Každý uživatel bude mít svůj soukromý klíč pro podepisování a veřejný klíč, který bude poskytovat ostatním účastníkům k ověření podpisu. Tyto klíče pro uživatele vygeneruje vhodným způsobem klient. Součástí identifikace uživatele bude právě veřejný klíč uživatele.

6.5.1. Podepisování komentářů

Součástí každého komentáře bude označení diskuse, ke které komentář patří. Číslo patche komentáře. Kdo komentář píše, tedy jméno uživatele. Samotný text komentáře. Z tohoto všeho klient vytvoří otisk pomocí vhodného hashovacího algoritmu a tento otisk podepíše svým soukromým klíčem. Podpis pak bude jako další položka v komentáři. Pokud je to první komentář daného uživatele do diskuse, musí klient v komentáři poslat zároveň veřejný klíč uživatele, aby server měl možnost komentář autorizovat. Komentář, na který reaguje a veřejný klíč je v komentáři volitelná položka. Pokud klient pošle uživatelský veřejný klíč do dané diskuse vícekrát než jednou, musí být tento veřejný klíč vždy stejný. Aby byl otisk příspěvku a tedy i celé diskuse konzistentní, zařazuje server veřejný klíč pouze do prvního komentáře daného uživatele.

[8] http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-rsa-cryptography-standard.htm

6.5.2. Autorizace a autentizace

Autorizace a autentizace bude provádět každý účastník komunikace, tedy každý uzel a to vždy při přijmutí každého nového komentáře. Uzel přijme komentář a zjistí, zda je součástí komentáře veřejný klíč autora komentáře. Pokud ano, zkontroluje, zda má k dispozici veřejný klíč k danému autorovi ve svém seznamu. Pokud uzel veřejný klíč nemá, zažádá si o něj uzlu, který mu komentář poslal. Má-li uzel veřejný klíč daného uživatele a i příspěvek obsahuje veřejný klíč, musí se tyto klíče shodovat. Neshodují-li se, uzel, který komentář přijal, pošle zpátky informaci o tom, že se klíče neshodují, uzlu, který mu komentář poslal. Dešifruje podpis veřejným klíčem uživatele, který je autorem komentáře. Vytvoří otisk z příslušných položek komentáře a porovná, zda jsou otisky stejné. Pokud ano, je komentář autorizovaný a tím zároveň autentizovaný uživatel, který ho vytvořil. Pouze takovýto autorizovaný komentář si uzel uloží do svého úložiště. Pokud otisky nesouhlasí, pošle nazpět informaci o tom, že komentář nebyl autorizovaný z důvodu neshody otisků.

6.6. Chybové kódy

200 OK

Když klient pošle serveru komentář a server komentář úspěšně autorizuje a uloží, pak pošle klientovi zpět zprávu 200 OK s novým otiskem a číslem verze dané diskuse, do které klient přispěl komentářem.

300 Message Not Understood

Chyba značí, že uzel nerozumí zprávě nebo zpráva nemá typ.

400 Bad Request

Tímto chybovým kódem odpoví server, pokud klient pošle požadavky, které na serveru nejsou definovány nebo je jakákoliv jiná chyba v požadavku.

401 Unauthorized

Tento chybový kód pošle server, pokud neautorizoval příspěvek z důvodu, že je špatný podpis komentáře.

402 Key Requried

Pokud server nemůže ověřit podpis komentáře z důvodu, že nemá ani u sebe ani v komentáři veřejný klíč uživatele, oznámí to klientovi chybovým kódem 402 key required.

403 Wrong Key

Chybu 403 wrong key vrátí server klientovi, pokud se snaží autorizovat příspěvek, ale má uložený k danému uživateli, autorovi komentáře, jiný veřejný klíč než který klient poslal v komentáři.

404 Not Found

Tato chyba znamená, že server požadovanou diskusi nemá a ani ji nemá žádný server, se kterým je aktuálně spojený.

500 Server Error

Toto je obecná chyba serveru

501 Version Mismatch

Chyba serveru, na kterou superuzel upozorní druhý superuzel, jakmile tuto chybu zjistí. Tato chyba znamená rozdílné komentáře ve stejné diskusi. Tuto chybu řeší souběžně oba superuzly každý zvlášť a to tím, že obě diskuse spojí a vytvoří nové otisky všech verzí.

6.7. Zprávy mezi uzly

Aby bylo možné rozlišit, co který uzel požaduje nebo na co odpovídá nebo co posílá, zavedli jsme dva různé typy zpráv

Request

Označuje požadavek klienta na server. Například pro získání konktrétního komentáře, vložení nového komentáře nebo připojení do diskuse.

Tato zpráva musí obsahovat atributy type s hodnotou request a method s hodnotami get nebo put nebo leave nebo join.

Response

Response je typ zprávy, kterou server dopovídá na požadavek klienta. Obsahuje chybový kód a případně odpověď na požadavek

Tato zpráva musí obsahovat atributy type s hodnotou response, code s třímístným číslem chyby a message s textem zprávy.

Každá tato zpráva má další povinné atributy, které vyplývají z návrhu komunikačního protokolu.

Pro přenos zpráv přes TCP stream si navrhneme jednoduchý protokol. Budeme předpokládat, že zpráva nebude delší než 4 GB, její velikost se tedy vejde do 4 bajtů, které pošleme přes socket jako první, a až poté vlastní obsah zprávy. Po přijetí prvních 4 bajtů tedy budeme vědět kolik dalších bajtů bude ještě potřeba přečíst. Délka bude v kódování Network Order, který používá Big Endian. Toto kódování definuje nejvyšší řád čísla na nejnižší adrese.

S protokolem popsaným výše je spojeno několik problémů:

- binární data jsou pro člověka složitá přečíst i napsat
- hlavička zprávy bez jakékoli redundance není robustní. Vyskytne-li se někde chyba, zpráva se poškodí a nebude možné se z poškození zotavit. Dokonce ani není možné chybu detekovat, přečtení 4 bajtů z dat zprávy může být interpretováno jako příchod dalších 2 GB dat, což může vést k nekonečnému čekání.
- omezení na 4 GB dat.
- nemožnost přidat další data do hlavičky, jako například formát dat.
- jakékoli rozšíření v budoucnu nebude zpětně kompatibilní

V našem případě ale není většina těchto problémů příliš relevantní a většinu z nich můžeme vyřešit použitím textového formátu JSON pro kódování zprávy.

Další možností by bylo si místo hlavičky zvolit nějaký ukončující znak, který se ve zprávě nesmí vyskytnout a číst zprávu dokud na tento znak nenarazíme.

V mnoha internetových protokolech (e-mail, FTP, IRC, HTTP...) se obvykle používá kombinace obou metod, tedy hlavičky ukončené nějakým speciálním znakem,

u kterého ale stačí, že se nebude vyskytovat pouze v hlavičce, protože data jsou už předem určená jejich velikostí.

7. Realizace prototypu

7.1. Vývojové nástroje

Jako programovací jazyk pro realizaci prototypu klienta jsem se rozhodoval mezi Javou a Pythonem. Javou, protože jsme v ní byli vedeni od začátku studia, Pythonem, protože se mi líbí a chtěl jsem si ho vyzkoušet. Nakonec jsem si vybral Python a této volby nelituji.

Vyhodnotil jsem jako nutné použít verzovací systém a repozitář. Běžně používám repozitář Github[9] s nástrojem Git[10] a ovládacím grafickým rozhraním SourceTree[11]. Pro závěrečnou bakalářskou práci jsem použil tyto nástroje, protože jsou zdarma, jsou pro potřeby bakalářské práce dostačující a dobře je znám.

IDE pro Python jsem použil program PyCharm od JetBrains[12], který je od stejných tvůrců jako IDE IntelliJ IDEA pro Javu, které už delší dobu používám a jsem s ním spokojen.

7.2. Implementace

Nižší vrstvu přenosu dat obstarává třída *NodeMixIn*, která má metody *listen* pro spuštění serveru a metodu *connect* pro připojení k serveru.

Jakékoli připojení, ať už ho inicializuje klient nebo server spouští v novém vlákně instanci třídy *NodeHandler*, která zajistí obsluhu spojení.

[9] https://github.com
[10] http://www.git-scm.com
[11] https://www.atlassian.com/software/sourcetree/overview
[12] https://www.jetbrains.org/pycharm/

Třída *NodeMixIn* má dále metody join pro přihlášení uzlu k odběru příspěvků k diskuzi na dané téma, a metodu leave pro odhlášení z diskuze. Pro odeslání nového příspěvku slouží metoda *put* a pro stažení metoda *get*. Metodu *get* se obvykle volá pouze uvnitř třídy *NodeMixIn*.

Tato část je společná serveru i klientovi.

Příchod nové zprávy zjišťuje třída *NodeHandler*, která nejprve prověří, jestli se jedná o požadavek, nebo o odpověď na požadavek, a poté předá zprávu dál třídě *RequestHandler*, nebo třídě *ResponseHandler*, v závislosti na typu zprávy.

Třída *RequestHandler* má metody *handle_join*, *handle_leave*, *handle_get* a *handle_put*, které obstarávají logiku serverové části protokolu jednotlivých požadavků. V metodě *handle_join* posílá seznam uzlů v dané diskusi uzlu, který o join požádal. Pokud diskusi nemá, zavolá join na ostatních spojeních, případně vrátí chybu 404. V metodě *handle_leave* si odebere uzel ze senzmau pro danou diskusi. V metodě *handle_get* pošle uzlu, který poslal požadavek, příspěvek ze svého seznamu příspěvků k dané diskusi. V metodě *handle_put* ověří, že je příspěvěk správně podepsán a pokud je, zařadí si ho do seznamu příspěvků k diskusi a odpoví pořadovým číslem příspěvku v diskusi a heší diskuse. Následně rozešle zprávu *put* všem dalším účastníkům diskuse. Při přijetí přeposlané zprávy *put*, už zprávu dál neodesílá. Dostane-li při odesílání zprávy *put* odpověď 501 Version Mismatch, zavolá metodu *merge*. Po obdržení seznamu hešů příspěvků aktualizuje verzi a heš příspěvku ve zprávě *put* a odešle ho znovu.

Třída *ResponseHandler* má metody *handle_ok*, *handle_bad_request*, *handle_unauthorized*, *handle_key_required*, *handle_wrong_key*, *handle_server_error* a *handle_version_mismatch*.

Třída *Client* má metodu send, která sestaví a odešle nový příspěvek, a metody *join* a *leave*, které posílají požadavky na server o přihlášení k odběru, respektive odhlášení od odběru nových příspěvků.

Část kódu, která je stejná pro klienta i pro server jsme naprogramovali společně praktikou párového programování a nachází se v souboru *common.py*. Jedná se hlavně o třídy *NodeMixIn* a *NodeHandler*.

Samostatně jsem potom naprogramoval třídy *ResponseHandler* a *Client*. Které implementují klientskou část diskusního protokolu.

8. Testování

8.1. Jednotkové testy

Třída *ResponseHandler* je testována pomocí jednotkových testů v souboru *client_tests.py* balíčkem unittest, který je součástí distribuce jazyka Python 2.7.

Dále byla provedena zkouška komunikace k připojenému serveru.

8.2. Provedené akceptační scénáře

8.2.1. Test připojení k serveru:

Klient se připojí k serveru, očekává se text „Connected to *address*“

Zobrazil se text: „Connected to (92.62.234.125, 9000)“

8.2.2. Test přihlášení do diskuse

Klient se připojí do diskuse, očekává se text: „Joined to topic *topic*“

Zobrazil se text: „Joined to topic firsttopic“

8.2.3. Test odeslání příspěvku

Klient pošle nový příspěvek, očekává se text: „Contribution sent to topic *topic* with text *text,* version *version*“

Zobrazil se text: „Contribution sent to topic firsttopic with text Hello World!, version 1“

8.2.4. Test odhlášení z diskuse

Klient se odpojí z diskuse, očekává se text: „Topic *topic* left“

Zobrazil se text: „Topic firsttopic left“

8.2.5. Test editace starého příspěvku

Klient edituje starý příspěvek, očekává se text: „Contribution edited in topic *topic* with text *text*, version *version*.

8.2.6. Test editace neexistujícího příspěvku

Klient edituje neexistující příspěvek, očekává se text: „Error 404 Not Found“

Zobrazil se text: „Error 404 Not Found“

8.2.7. Test editace příspěvku, který nepatří klientovi

Klient edituje příspěvek podepsaný jiným klíčem, očekává se text: „Error 401 Unauthorized“

Zobrazil se text: „Error 404 Unauthorized“

8.2.8. Test spojení diskusí

Klient 1 se připojí k serveru 1, klient 2 se připojí k serveru 2, klient 1 pošle příspěvek do diskuse firsttopic, klient 2 pošle příspěvek do diskuse firsttopic, server 1 se připojí k serveru 2. Očekává se, u klienta 1 text: „New contribution to topic firsttopic with text Hello from client 2“ a u klienta 2 text: „New contribution to topic firsttopic with text Hello from client 1“

U klienta 1 se zobrazil text: „New contribution to topic firsttopic with text Hello from client 2“, u klienta 2 se zobrazil text: „New contribution to topic firsttopic with text Hello from client 1“

9. Závěr

Navrhli jsme protokol pro komentování webových článků dle zadání. Naprogramoval jsem ukázkový prototyp klientské aplikace, která komunikuje se serverem pomocí protokolu navrženého v této bakalářské práci. Na více počítačích jsme si vyzkoušeli fungování tohoto protokolu a prototypů serveru s připojenými klienty.

Řešení je funkční a provedené testy splňují naše očekávání. Protokol se jeví jako vhodný základ pro vytovření robustního nástroje pro distribuované a svobodné diskutování na internetu.

V průběhu implementace a testování vzniklo několik nápadů na zlepšení, které nebyly součástí této bakalářské práce, a kterým bychom se chtěli věnovat do budoucna.

1. Integrování klienta a serveru do jedné aplikace.

2. Vytvoření GUI k celé aplikaci.

3. Nalézt vhodnou ochranu proti nevyžádaným příspěvkům – spamu, například zavedením tagování příspěvků.

4. Navrhnout vyhledávání diskusí.

Implementovaný prototyp a současný návrh protokolu jsou určeny pro vývojáře, kteří by je chtěli vhodně rozšiřovat a přiblížit je tak do verze, kterou by bylo možné oficiálně představit a vydat k testování širší veřejnosti.

Vy vývoji návrhu komentovacího protokolu bych rád pokračoval a rozšiřoval tak jeho možnosti.

Týmová praktika Párového programování, kterou jsem si v bakalářské práci mohl vyzkoušet se mi velmi líbila a vnímám ji jako velký přínos. Práce ve dvou se střídáním psaní mi zlepšila koncetrování se na danou problematiku a chyby z nepozornosti obvykle ihned odhalil druhý vývojář. Také možnost okamžitě s někým konzultovat to, co právě dělám, s někým, kdo dělá přesně to samé, je výborná.

Nevýhodu Párového programování bych viděl jen v případě, že dva vývojáři mají zcela rozlišnou ideologii, co a jak právě tvořit a jsou oba příliš autoritativní. To se nám ale nestalo a společně tvořená práce byla oběma dobrým přínosem.

10. Přílohy

10.1. Extrémní programování

10.1.1. Zrod Extrémního programování

Extrémní programování je agilní metodika vývoje softwaru. Základní popis z počátku devadesátých let dvacátého století je přisuzován trojici Kent Beck, Ward Cunningham a Ron Jeffries. Tento popis vznikl během jejich práce na projektu Chrysler Comprehensive Compensation System, zkratkou přezdívaný C3. Tento projekt byl nasazen v roce 1997 a je často referován v literatuře o Extrémním programování.

Extrémní programování je i metodologií vývoje software. Jednotlivé praktiky během vývoje software jsou zkoumány a vyhodnocovány, zda jsou dostatečně agilní. Pokud se prokáže, že jsou příliš rigidní, jsou buď změněny nebo vyřazeny z metodiky nebo nahrazeny praktikou jinou.

10.1.2. 12 základních praktik extrémního programování

Obchodní praktiky

1. **Plánovací hra.** Zákazník společně s vývojáři postupně určují, jak dosáhnout co nejlepšího přínosu softwaru v co nejkratší čas. V první fázi zákazník sepíše seznam funkčních požadavků, které od systému požaduje. V druhé fázi vytvoří vývojáři společně se zákazníkem z každého funkčního požadavku uživatelský scénář, ve kterém je funkčnost popsána kde jazykem zákazníka funkce popsána. Ve třetí fázi analyzují vývojáři jednotlivé scénáře a určí za jaký čas a za jakou cenu mohou tyto funkce naprogramovat. Ve čtvrté fázi zákazník rozhodne o prioritách jednotlivých uživatelských scénářů a co má být implementováno jako první. Scénáře jsou pak zařazeny do jednotlivých iterací podle zvolených priorit.
2. **Zákazník při vývoji.** K vytvoření použitelného software je potřeba mít k dispozici skutečného uživatele, která bude se softwarem pracovat. Tato osoba musí mít pravomoc měnit požadavky a stanovovat priority.
3. **Vydávání malých verzí.** Vydat novou verzi při vytvoření každé nové funkcionality. Pro vývojáře je tak rychlejší zpětná vazba a pro zákazníka je najednou méně změn.

4. **Metafora.** V projektovém týmu je důležité používat zapamatovatelné a jednoduché pojmy, aby všichni dokázali jednoduše o vyvíjeném systému komunikovat. Metafora je systém názvu a popisů vyvíjeného softwaru.

Týmové praktiky

5. **Párové programování.** Na implementování kódu vždy pracují společně dva programátoři u jednoho pracoviště. Společně tvoří jeden zdrojový kód. Zatímco jeden píše kód, druhý přemýšlí o právě psaném kódu a zda je to optimální řešení daného návrhu. Oba programátoři spolu stále probírají své záměry a postřehy a vzniká tak rychlá zpětná vazba. Programátoři se v psaní střídají. Během produkčního dne se střídá i složení párů.
6. **Společné vlastnictví kódu.** Každý programátor vlastní celý zdrojový kód. Kdokoliv může měnit jakoukoliv část kódu. Vše prochází testy, pokud někdo udělá chybu, hned se projeví, kde chyba je. Společné vlastnictví kódu je nutné pro párové programování a refaktorování.
7. **Standardy kódu.** Pro efektivní spolupráci na společném kódu dodržují všichni společné standardy. Aby se každý programátor orientoval v jakékoliv části kódu. V ideálním případě není z kódu patrné, kdo ho psal. Bez takových standardů by programování v měnících se párech a refaktorování nebylo efektivní.
8. **Udržitelné tempo.** Jde o zvolení vhodné pracovní doby tak, aby programátoři nebyli přetěžovaní. Pokud je programátor časově přetěžovaný, je unavený a zvýší se chybovost jeho práce a zároveň se práce zpomalí. To je pro vývoj software nežádoucí jev. Pracovní tempo může být pro každého programátora individuální, obecně z praxe se má za to, že je vhodné pracovat 40 hodin týdně. Přesčasy jsou nežádoucí, protože hrozí riziko únavy. Pokud je někdy potřeba pracovat déle, dodržuje se pravidlo přesčasů ob jeden týden.

Programovací praktiky

9. **Testování.** Pro každou funkcionalitu napíší programátoři nejprve testy. Samotnou implementaci kódu poté programují proti těmto testům tak, aby testy proběhly 100% úspěšně. Tato praktika se jmenuje Vývoj řízený testy, v anglické literatuře pak akronymem TDD, který znamená Test-Driven Development.
10. **Jednoduchý návrh.** Vždy navrhovat a implementovat jen to, co splňuje právě aktuální požadavky zákazníka. Protože zákazník může své požadavky kdykoliv

změnit, není ani časově ani finančně vhodné programovat funkcionality, které aktuálně nepožaduje. Je velmi možné, že takové funkcionality ani nikdy požadovat nebude.

11. **Průběžná integrace.** Minimálně jednou denně integrovat nový otestovaný kód do produkčního softwaru. Integraci provádí pouze jeden pár programátorů. Po integraci se provedou integrační testy. Případné integrační chyby jsou rychleji dohledatelné a opravitelné, protože se integruje poměrně malá část kódu, který je čerstvý
12. **Refaktorování kódu.** Tento pojem zavedl americký programátor anglického původu Martin Fowler. Jedná se o vylepšování již existujícího funkčního kódu. Vylepšuje se jednak čitelnost kódu pro programátory a jednak se vylepšuje kód pro snazší strojové zpracování. Pokud nějaký programátor vidí kód, který by se dal napsat lépe, napíše ho lépe. Samozřejmostí je následné testování.

Extrémní nejsou jednotlivé praktiky, ale způsob jejich použití a dodržování. Využití zpětné vazby mezi zákazníkem, návrhem a programováním. Dbá se na soustavnou komunikaci mezi jednotlivými účastníky vývoje. Každý den by měl začít krátkou schůzkou, kde si všichni účastníci vymění své poznatky a sdělí problémy, které se pak následně řeší na té rovině, kde vznikly.

10.2. Obsah přiloženého DVD

Na přiloženém DVD na konci této práce jsou soubory:

wimberic_2015bach.docx	text této bakalářské práce ve formátu DOCX
wimberic_2015bach.pdf	text této bakalářské práce ve formátu PDF
readme.txt	návod na spuštění klientské aplikace
requirements.txt	soubor se závislostmi aplikace pro program pip
common.py	soubor se společnou částí aplikace
client.py	soubor s vlastní implementací klienta
client_tests.py	soubor s jednotkovými testy aplikace

Seznam obrázků

Seznam použitých zkratek

AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
XHTML	Extensible Hypertext Markup Language
FTP	File Transfer Protocol
IRC	Internet Relay Chat
WWW	World Wide Web
GUI	Graphical User Interface
IDE	Integrated Development Enviroment
API	Application Program Interface
HTTP	Hypertext Transfer Protocol
RSS	Rich Site Summary
P2P	Peer to Peer
TCP	Transmission Control Protocol
JSON	JavaScript Object Notation
POP3	Post Office Protocol, version 3
SMTP	Simple Mail Transfer Protocol
IMAP	Internet Access Message Protocol
IETF	Internet Engineering Task Force
RFC	Request for Comments
TDD	Test-Driven Development

Literatura

[1] WIMBERSKÝ, Tomáš. *Protokol komentování webových článků – server, bakalářská práce*. ČVUT, 2015.

[2] *API - Disqus* [online]. [cit. 2015-05-20]. Dostupné z: https://disqus.com/api/docs

[3] *Comments* [online]. [cit. 2015-05-20]. Dostupné z: https://developers.facebook.com/docs/plugins/comments

[4] BECK, Kent. *Extreme programming eXplained: embrace change*. Reading, MA: Addison-Wesley, 2000, xxi, 190 p. ISBN 02-016-1641-6.

[5] FOWLER, Martin. *Refactoring: improving the design of existing code*. Boston: Addison-Wesley, c2000, xxi, 431 s. Addison-Wesley Object Technology Series. ISBN 0201485672.

[6] ČADA, Ondřej. *Objektové programování: naučte se pravidla objektového myšlení*. 1. vyd. Praha: Grada, 2009, 200 s. ISBN 978-80-247-2745-5.

[7] SPURNÁ, Ivona. *Počítačové sítě: praktická příručka správce sítě*. Vyd. 1. Kralice na Hané: Computer Media, c2010, 180 s. ISBN 9788074020360.

[8] KUROSE, James F a Keith W ROSS. *Počítačové sítě*. 1. vyd. Brno: Computer Press, 2014, 622 s. ISBN 9788025138250.

[9] HARMS, Daryl D a Kenneth MCDONALD. *Začínáme programovat v jazyce Python*. 2., opr. vyd. Překlad Ivo Fořt, Lubomír Škapa. Brno: Computer Press, 2008, xvi, 456 s. ISBN 978-80-251-2161-0.

[10] KOEGEL BUFORD, John F, Hong Heather YU a Eng Keong LUA. *P2P networking and applications*. Boston: Elsevier/Morgan Kaufmann, c2009, xxi, 415 p. Morgan Kaufmann series in networking. ISBN 0123742145.