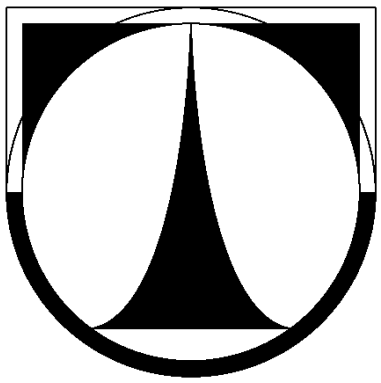
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií



BAKALÁŘSKÁ PRÁCE

květen 2011 **Jiří Sojka**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 Informační technologie
Studijní obor: 1802R007 Informační technologie

Paralelní verze Verse serveru
Parallel version of Verse server

Jiří Sojka

Vedoucí práce: Ing. Jiří Hnídek
Pracoviště: Ústav nových technologií a aplikované informatiky

Čestné prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum: 19. května 2011

. . . . . .

podpis

Poděkování

Na tomto místě bych chtěl poděkovat především své rodině, bez jejichž trpělivosti a podpory bych se k této práci zřejmě nedostal.

Dále bych zde chtěl poděkovat svému vedoucímu Ing. Jiřímu Hnídkovi za ochotu, trpělivost a přístup k vedení mé bakalářskě práce.

Použitý software

Tato práce byla vysázena programem LaTeX pod operačním systémem Fedora 13. Nástroj pro tvorbu zdrojových kódů byl použit Eclipse Galileo ve verzi 3.5.2 s externími pluginy pro komunikaci s SVN a vygenerování dokumentace ke zdrojovým kódům. Testování proběhlo na linuxovém serveru, virtualizovaném pomocí Oracle VM VirtualBoxu verze 4.0.4.

Kontakt

E-mail: jiri.sojka@tul.cz

Anotace

Tato bakalářská práce se zabývá implementací paralelní verze Verse serveru. Protokol Verse je v této době ve fázi implementace nové verze, která by měla splňovat požadavky na real–timovou výměnu 3D dat mezi sdílenými aplikacemi, především v oblasti počítačové grafiky.

Práce se zabývá rozšířením aktuální verze Verse serveru o takzvaný mód master a slave. Server v módu master bude schopen přijmout připojení od klientů i od slave serverů (server v módu slave). Dále bude master server nové klienty přesměrovávat na jeden z připojených slave serverů. Důraz je kladen na bezpečnost, jak při navázání spojení, tak při vzájemné komunikaci mezi servery. Z tohoto důvodu je využita bezpečnostní knihovna OpenSSL.

Klíčová slova: mód master, mód slave, Verse, server

Abstract

This bachelor thesis is focused at the implementation of the parallel version of the Verse server. Verse protocol Verse is nowadays in the stage of implementation of a new version that should meet the requirements for real–time 3D data transfer within the shared applications, particularly in the domain of computer graphics.

Thesis id concerned with the extension of the Verse server's current version by so-called master and slave mode. Server in mode master will be able to accept the connections from clients, incl. slave servers (server in mode slave). In addition, the master server will forward new clients to one of the connected slave servers.

Special emphasis is then placed on the security - both when estabilishing the connection and also on the common communication within the servers. Because of this the OpenSSL security library is used.

Keywords: mode master, mode slave, Verse, server

Obsah

Seznam obrázků

Seznam zkratek

AES Advanced Encryption Standard

API Application Programming Interface

DTLS Datagram Transport Layer Security

IDE Integrated Development Environment

IP Internet Protocol

JDT Java Development Tools

PDE Plug–in Development Environment

RFC Request For Comments

SSL Secure Socket Layer

SVN Subversion

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

URL Uniform Resource Locator

VPN Virtual Private Network

1 Úvod

V oblasti vývoje počítačové grafiky je v této době vyžadován efektivní přenos dat mezi grafiky pracující v týmu, například v grafickém studiu. Protokol, který měl splnit požadavky pro real–timový přenos dat mezi grafickými aplikacemi, je protokol Verse [9]. Na vývoji tohoto síťového protokolu se podílelo Uni–Verse konsorcium v rámci 6. rámcového programu Evropské unie. Konsorcium se sice skládalo z několika významných evropských univerzit a výzkumných institucí (Franhoufer, KTH, Helsinky University of Technology a další), ale po ukončení financování z Evropské unie vývoj protokolu téměř ustal a tento protokol se příliš nerozšířil [9].

Aktuálně na novém návrhu a zároveň implementaci tohoto síťového protokolu pracuje v rámci disertační práce Ing. Jiří Hnídek. Tato bakalářská práce by měla aktuální verzi Verse serveru modifikovat a rozšířit o možnost paralelního zpracování dat. Server by měl být schopen přesměrovávát klienty, kteří se přihlásí k master serveru, na jeden z připojených slave serverů, a tím rozložit zátěž hlavního serveru.

Důležitým bodem bude jak zabezpečené navázání komunikace, tak samotný zabezpečný přenos dat mezi servery. Modifikace by se neměla dotknout klientské části především díky vhodnému návrhu protokolu.

Úvodem této práce je nejprve rešeršní část, v níž je úvod do problematiky Verse serveru a nastínění požadovaných cílů. Dále jsou uvedeny kroky získávání potřebných znalostí pro následnou implementaci paralelní verze Verse serveru.

Součástí je také závěrečné otestování a zhodnocení práce v samotném závěru.

2 Aktuální verze Verse protokolu

V této kapitole jsou popsány základní části Verse protokolu, s nimiž se bylo potřeba seznámit, popřípadě je modifikovat pro dosažení požadované funkcionality.

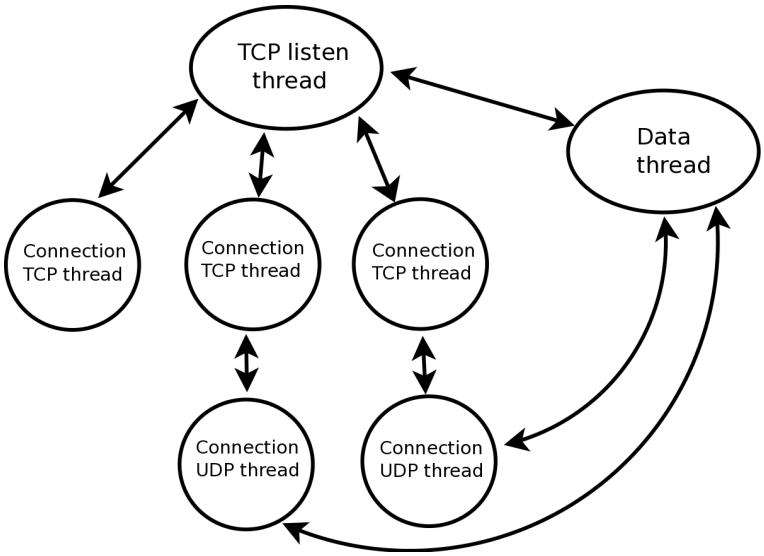
Stará verze Verse protokolu byla implementována do 3D modelovacího a animačního programu Blender. Právě zde se ukázalo mnoho nedostatků. Již samotná implementace byla komplikovaná, nestabilní a nakonec nekompletní. To bylo zapříčeněno především původním ad–hoc návrhem protokolu, jeho možnostmi implementace a nekompletní specifikací.

Nová implementace protokolu Verse si klade za cíl být na rozdíl od staré verze robustní a spolehlivá. Podmínky pro návrh a vlastní implementaci vycházejí z potřeby umožnit přenos dat mezi grafickými aplikacemi v síti Internet v reálném čase. Z této myšlenky vychází celá řada požadavků na protokol Verse [9]. Důležitý je přenos dat s nízkou latencí, tuto vlastnost nabízí transportní prokotol UDP [18], zároveň je ovšem potřeba zajistit alespoň částečnou spolehlivost přenášených dat. Právě za tímto účelem byl vyvinut protokol Verse.

Nový protokol Verse byl od základu přepracován s ohledem na efektivitu, spolehlivost a bezpečnost přenášených dat. Mezi klientem a serverem je umožněna dohoda o vlastnostech datové komunikace. Pro dosažení částečné spolehlivosti přenášených dat byl navržen nový resend mechanismus, který efektivněji využívá přenosové linky a přeposílá pouze aktuální data. Komplexněji je celý protokol Verse popsán v dokumentu [9].

2.1 Aktuální verze Verse serveru

Implementace Verse serveru je vícevláknová aplikace, kde byl kladen důraz na propustnost přenášených dat a maximální výkon. Server v samostatném vláknu naslouchá a detekuje nová spojení od klientů. Pokud je spojení navázáno, vytvoří se pro komunikaci s klientem vlákno nové, ve kterém se provede TCP [13] a následný TLS [8] handshake . Po této autentizaci je dohodnuto nové UDP spojení, ve kterém probíhá samotná komunikace s klientem, včetně DTLS [14] handshaku a handshaku datagramového spojení. Pokud se některý z handshaků nezdaří, jsou systémové prostředky uvolněny pro navázání dalšího spojení [9].

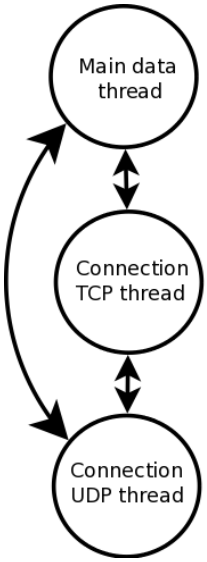


Obrázek 1: Vlákna aktuální verze Verse serveru

2.2 Aktuální verze Verse klienta

V nové verzi Verse klienta došlo k návrhu a implementaci nového API. Verse klient může efektivně komunikovat i s více servery najednou, přičemž komunikace s každým serverem probíhá ve dvou samostatných vláknech. TCP vlákno slouží pro autentizaci a dohodnutí datagramového spojení. Datagramové UDP vlákno je určeno pro samotný přenos dat. Výměna dat mezi vlákny je prováděna pomocí fronty příkazů [9].

Tato práce je zaměřena na samotnou implementaci modifikace Verse serveru, klientská část by proto měla zůstat ve stávající podobě.



Obrázek 2: Vlákna aktuální verze Verse klienta

2.3 Dohadování o URL

Poté, co se klient úspěšně autentizuje na serveru, dochází k vyjednání URL. Klient navrhuje URL, server buď navrhne jiné řešení, nebo potvrdí klientovo. URL musí mít konkrétní tvar: *schéma, ip adresu, port*. Ve schématu se nastavují základní vlastnosti datagramového spojení.

Schéma musí vždy začínat řetězcem *verse*, poté následuje řetězec udávající protokol používaný na transportní vrstvě. V této verzi je podporován transportní protokol UDP (řetězec *udp*). Poslední částí schématu je informace o zabezpečení spojení. Pokud je vyžadována bezpečnost, je ve schématu jako poslední řetězec *dtls*, pokud není vyžadováno zabepečení, je řetězec ve tvaru *none*. Mezi řetězci je vždy pomlčka. Celé schéma je ukončeno ://. Je tedy možné použít dva tvary schémat:

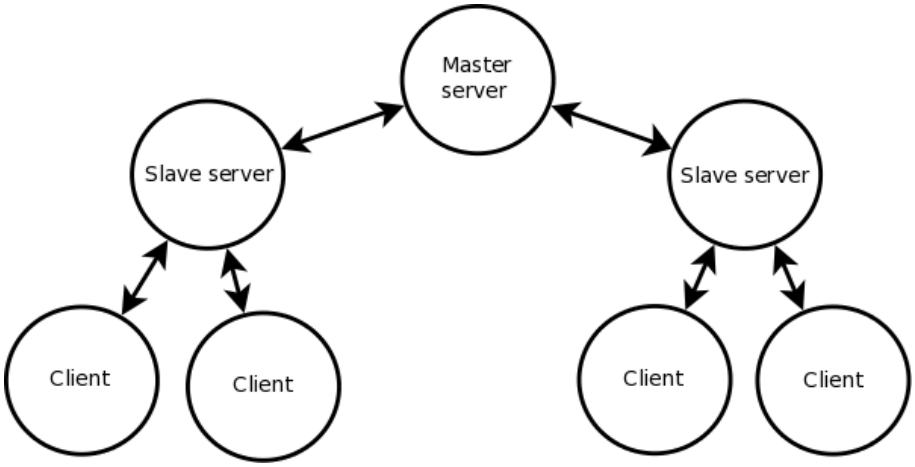
- verse-udp-none://
- verse-udp-dtls://

Část o vyjednávání nejen URL [15] je popsána v dokumentu [9]. Pomocí tohoto mechanismu nebylo obtížné odeslat klientovi URL serveru, na který má být přepojen. Samotné přesměrování klienta je popsáno v kapitole 6.1.3.

3 Motivace

Již v návrhu Verse protokolu se počítalo s robustním nasazením, proto je zřejmé, že budou kladeny vysoké nároky na server ohledně zpracování požadavků od klientů. První možností je neustálá obnova hardwaru serveru pro získání většího výpočetního výkonu, což zcela jistě není efektivní. Druhým případným řešením je pokusit se rozdělit zátěž mezi více serverů, jenž by spolu byly schopny vzájemně komunikovat a rozdělovat požadavky od klientů mezi sebe.

Cílem této bakalářské práce je rozšíření aktuální verze Verse serveru o možnost paralelního zpracování požadavků od klientů. Verse server by měl mít možnost být spuštěn buď v módu master, nebo v módu slave, a to využitím volby (popřípadě parametru) při spuštění na příkazové řádce. Stále ovšem zůstane zachována i původní funkce serveru, pokud se aplikace spustí bez zadané volby.



Obrázek 3: Navrhované schéma komunikace

3.1 Master mód

Verse server v módu master by si měl nadále zachovat vlastní funkci ve smyslu naslouchání pro požadavky klientů, zároveň je ale potřeba naslouchat pro požadavky na připojení od slave serverů. Po navázání zabezpečeného spojení se slave serverem pomocí protokolu TLS a úspešné autentizaci, mohou být noví klienti přesměrovány na daný slave server, který je výbrán podle zvolených kritérií. Samozřejmě pouze pokud slave server disponuje systémovými prostředky pro přijetí nového připojení. Samotná klientská část by neměla být modifikována, a proto by klient neměl poznat rozdíl v komunikaci.

3.2 Slave mód

Při spuštění Verse serveru v módu slave se server nejprve připojí k master serveru. Po připojení bude slave server čekat na data o klientovi od master serveru. V případě dostatečných prostředků pro nového klienta, by měl slave server master serveru potvrdit požadavek o přepojení klienta. Poté bude klient přesměrován na daný slave server. Při přerušení komunikace nebo pádu master serveru, by měl být slave server dále schopen obsluhy připojených klientů.

4 Zabezpečení komunikace

V této době plné počítačových hackerů a crackerů, kteří se snaží využívat programátorských chyb nebo nezabezpečené komunikace, je důležitým bodem bezpečnost. Na veřejných sítích obecně nebývá zajištěna ochrana přenášených dat, proto je potřeba zajistit autentizaci komunikajících stran a zabezpečenou komunikaci. V projektu Verse se za účelem zabezpečené koumunikace používá protokol Transport Layer Security (TLS) a jeho implementace z knihovny OpenSSL [6]. Proto je protokol TLS použit i v paralelním rozšíření.

Za vznikem OpenSSL stojí celosvětová komunita vývojářů, kteří se podílejí na společném plánování a rozvíjení této bezpečnostní knihovny včetně odpovídající dokumentace. Knihovna OpenSSL je open source software a vydává se pod duální licencí (*OpenSSL License* a *SSLeay License*) [6]. Uživatel si může vybrat podle které licence se bude řídit. Obě licence jsou založeny na licenci BSD [16].

4.1 Využití knihovny OpenSSL

OpenSSL poskytuje knihovny napsané v jazyce C, které jsou dostupné pro unixové operační systémy (Solaris, Linux, operační systémy BSD), serverové operační systémy OpenVMS a Microsoft Windows. Tato knihovna podporuje mnoho různých (kryptografických) symetrických šifer (Blowfish, Camelia, RC4, RC5, AES), hashovacích funkcí (MD5, MD2, SHA), asymetrických šifer (RSA, DSA, Diffie–Hellman) a implementuje i kompresní formáty a celé bezpečnostní protokoly (SSLv1, SSLv2, TLS, DTLS). Právě z toho důvodu, že se jedná o open source software, byla tato knihovna použita v této práci.

Zabezpečená komunikace mezi klientem a Verse serverem (autentizace i přenos dat) je v aktuální verzi Verse serveru plně implementována. V této práci je potřebné zajistit zabezpečenou komunikaci mezi master serverem a slave serverem. Pro navázání a průběh komunikace byl zvolen protokol TLS.

4.2 TLS protokol

Tento protokol byl navržen již roce 1999 a je umístěn v hierarchii mezi vrstvou transportní a aplikační. Jeho úkolem je zabránit odposlouchávání nebo změně přenášených dat mezi koncovými uzly. Samotný protokol je rozdělen do dvou částí:

- TLS Handshake protokol – zajišťuje autentifikaci a dohodu o šifrovacím algortimu

- TLS Record protokol – zajišťuje šifrování a zapouzdřuje protokoly vyšších vrstev [8]

4.3 TLS Handshake protokol

Tento protokol zajišťuje vyjednání podmínek pro spojení a způsob komunikace. TLS Handshake je dále rozdělen na části:

- Alert protokol – množina zpráv, sloužící k indikaci chyby
- Change Cipher Spec protokol – slouží jako způsob potvrzení vyjednaného klíče a šifrování
- Handshake protokol – samotný handskake, buď úplný nebo částečný, který vychází z dříve navázaného spojení

4.3.1 Handshake protokol

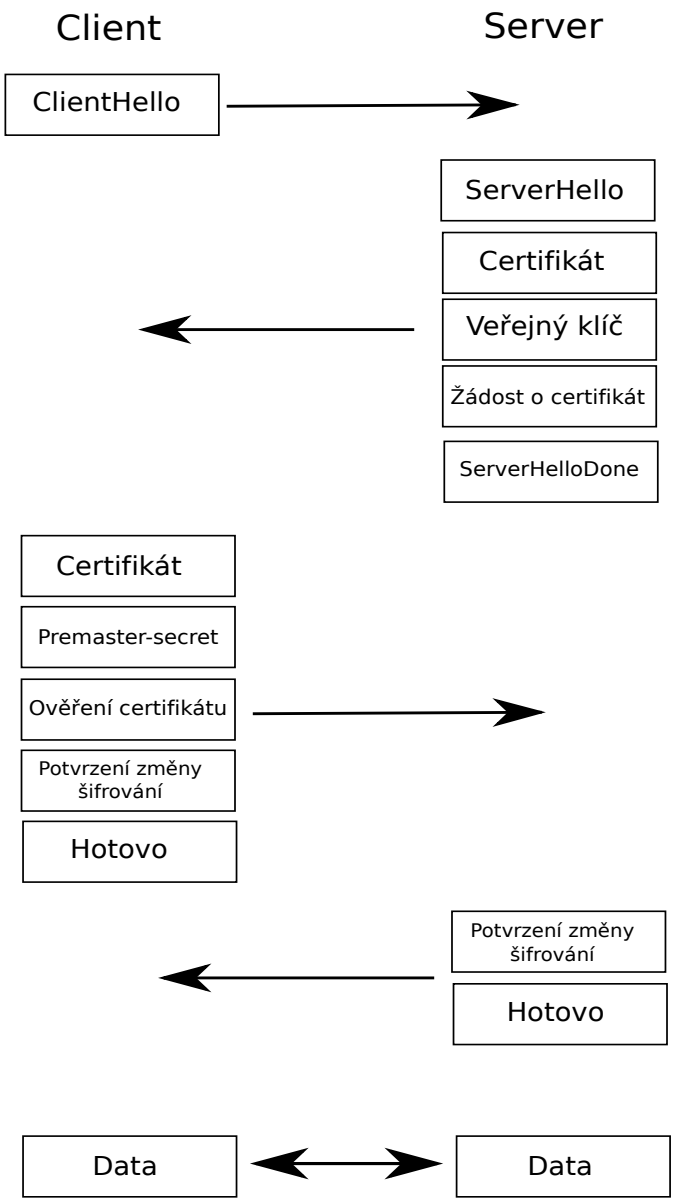
Handshake znamená v překladu potřesení rukou dvou lidí, jež se seznamují. Podobné je to i u Handskahe protokolu, kdy se koncové body navzájem „představují“ a vyměňují si informace o způsobu komunikace, jak je znázorněno na obrázku 4, například:

- identifikátor spojení
- certifikát
- způsob komprese data
- algortimy pro šifrování dat

Spojení zahajuje klient zasláním zprávy *ClientHello*, ve které oznamuje nejvyšší podporovanou verzi TLS, seznam šifrovacích metod, kompresních metod a náhodné číslo.

Server odpoví zprávou *ServerHello*, obsahující zvolenou verzi protokolu, náhodné číslo, šifrovací a komprimovací metodu zvolenou ze seznamu, který nabídl klient. Dále odešle server svůj certifikát (*Certificate*) a ukončující zprávu inicializace *ServerHelloDone*.

Klient odešle zprávu *ClientKeyExchange*, v níž je buď veřejný klíč, nebo je zpráva prázdná v závislosti na zvolené šifře. Také odešle zprávu *ChangeCipherSpec*, ve které upozorňuje, že další komunikace již bude šifrována, a ukončující zprávu *Finished*.



Obrázek 4: Kompletní TLS Handshake

Server se pokusí dešifrovat záverečnou zprávu od klienta a ověřit její hash. Pokud je dešifrování úspěšné, odpoví server zprávou *ChangeCipherSpec* a svou vlastní zašifrovanou zprávou *Finished*. Pokud klient také dešifruje a ověří zprávu, je inicializace ukončena a dále se mohou přenášet data v zašifrované podobě [8].

4.3.2 TLS Record protokol

TLS Record protokol zajišťuje zapouzdření dat z vyšších vrstev. Data rozdělí do bloků o maximální velikosti 16KB, popřípadě i zkomprimuje bezeztrátovou kompresí. Kroky odesílání vypadají následovně:

- rozdělení dat do bloků - fragmentace
- volitelná komprese
- vypočet kontrolního součtu
- zašifrování
- předání zprávy nižší vrstvě pro následné odeslání

podobné, i když samozřejmě opačné, je to s příjmem zašifrované zprávy:

- dešifrování zprávy
- ověření kontrolního součtu
- dekomprimace a sestavení
- předání vyšším vrstvám

Transport Layer Security vychází z rozšířeného protokolu SSL verze 3 [10], ovšem kvůli změnám především v bezpečnosti nejsou tyto protokoly vzájemně kompatibilní. V současné době stále roste počet jak klientských, tak samozřejmě i serverovských aplikací, které nativně podporují TLS. Tento protokol lze dále použít například pro tunelování síťových protokolů a vytváření VPN [21].

Jelikož je protokol TLS již nějakou dobu vyvinut, byly do něj postupným časem přidávány různé druhy rozšíření. Mezi nejznámější patří především šifra AES a takzvaná TLS Extensions, což jsou rozšíření určená především pro mobilní zařízení používající bezdrátové připojení. Více je popsáno v RFC dokumentu [11].

5 Příprava před implementací

Před samotnou implementací paralelní verze Verse serveru bylo samozřejmě nutné získat určité znalosti se soketovým programováním v jazyce C pod operačním systémem Linux [7], s konkrétními funkcemi OpenSSL [3] a dále vývojovým prostředím Eclipse včetně pluginu pro práci s SVN.

5.1 Jazyk C

Jazyk C vznikl již počátkem 70. let 20. století. Dnešní oficiální standard je tzv. ANSI C z roku 1990. V současné době se jedná o oblíbený jazyk pro psaní především systémového softwaru. Jazyk C byl navržen a implementován v operačním systému Unix a téměř celý Unix je v jazyce C napsán. Základní vlastnosti jazyka C jsou:

- univerzální programovací jazyk
- jazyk nízké úrovně (*low level language*)
- úsporné vyjadřování, strukturovaný, case sensitive
- není specializovaný na konkrétní oblast použití
- pro velký počet úloh je efektivnější a rychlejší než jiné programovací jazyky

Jazyk C je „jazyk nízké úrovně“, což znamená, že pracuje pouze se standardními datovými typy (znaky, celá čísla, reálná čísla). Neumožňuje přímo práci s řetezci a ani neobsahuje nástroje pro vstupy a výstupy. Tyto akce je nutné provádět pomocí volání knihovních funkcí, z čehož vyplývá jednoduchost jazyka a nezávislost na konkrétním počítači.

Program, napsaný podle standardu ANSI C a s využitím standardních knihovních funkcí, je téměř 100% přenositelný na libovolný operační systém.

Pro plné využtí jazyka je potřeba se seznámit také s tím, jak je program zpracován od napsání zdrojového textu až po spuštění samotného programu. Základní zpracování programu v jazyce C probíhá několika fázemi [4]:

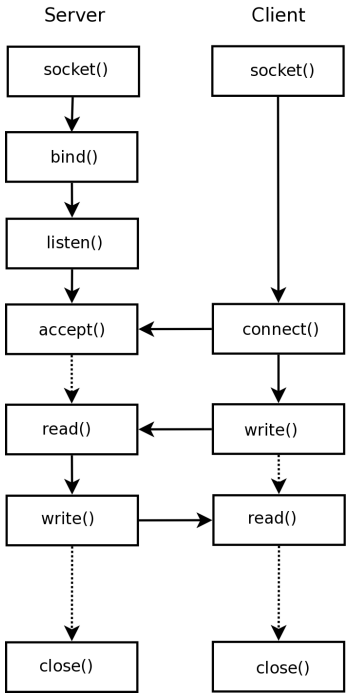
- editor – vytváří zdrojový soubor (.C)
- preprocesor – součást překladače, předzpracovává zdrojový kód (vkládá hlavičkové soubory, vynechává komentáře, nahrazuje makra a konstanty)
- compiler – provádí překlad zdrojového souboru

- linker – sestaví program, přidělí relativním adresám obsolutní (výsledkem je spustitelný soubor nebo knihovna)
- debugger – ladící program, slouží pro nalezení a odstraňování chyb

V této práci byl využit právě jazyk C, protože celá implementace Verse protokolu je napsána v tomto jazyce.

5.2 Soketové programování

Soket jako takový je obecný nástroj pro komunikaci. Poprvé se tento pojem objevil již v operačním systému BSD. Tento nástroj umožňuje komunikovat s ostatními systémy v síti. Na soket se lze dívat jako na koncový bod síťového připojení. Stejné funkce pro využití soketů lze používat pro komunikaci různých protokolů, ovšem v této práci byl potřeba především transportní protokol TCP.



Obrázek 5: Schéma programování soketů [12]

Důležité je, že soketové API je shodné pro všechny Unixové operační systémy (stejné funkce se stejnými parametry). Při použití soketů pod operačním systémem Windows jsou sice různé odlišnosti, ale tato práce byla psána v operačním systému Linux.

Při programování zkušebních aplikací typu klient–server byl zvolen postup pro získání praktických zkušeností především podle seriálu o soketovém programování a jazyku

C/C++[7]. Zde je vhodně krok za krokem popsána základní problematika používání soketů, včetně názorných příkladů, které byly potřeba pro vlastní práci.

Nejprve byla vytvořena jednoduchá aplikace typu klient–server pro osvojení si problematiky samotných soketů. Poté byl do obou částí (server i klient) implementován protokol TLS za využití knihovny OpenSSL.

5.3 Bezpečnost

Při implementaci zabezpečené komunikace mezi klientem a serverem bylo potřeba detailněji prostudovat oficiální dokumentaci k OpenSSL knihovně na oficiálních stránkách [6]. Tato knihovna se stále vyvíjí a zdokonaluje, proto je důležité využívat vždy nové a zároveň spolehlivé funkce, které jsou nabízeny.

Získávání použitelných zkušeností probíhalo především ze studia stávající verze Verse serveru, ve které je knihovna OpenSSL také využita. Dále jsou některé postupy použití popsány v článku [3]. Samozřejmostí bylo se s potřebnými funkcemi seznámit a umět vhodně použít [6].

5.4 Eclipse

Vývojové prostředí Eclipse je open source software napsaný v jazyce Java, který je pro vetšinu lidí známý jako prostředí pro vývoj právě v jazyce Java. Ovšem flexibilní návrh této platformy umožňuje seznam podporovaných programovacích jazyků pomocí dodatečných pluginů rozšířit, například o podporu jazyka C/C++, PHP a dalších.

Eclipse je možné volně stáhnout v základní verzi z oficiálních stránek [19]. V základní výbavě je již k dispozici samotné IDE prostředí, JDT modul pro programování Java aplikací a modul PDE pro programování nových plug–inů do systému Eclipse.

V této práci se toto vývojové prostředí osvědčilo také díky využitelnosti SVN po doinstalování vhodného pluginu.

5.5 Subversion (SVN)

Tématu protokolu Verse se věnuje více lidí, proto bylo vhodné, aby každý pracoval ve své vývojové větvi (branche) a navzájem nedocházelo k přepisování kódu někoho jiného, nebo naopak. Při změně kódu v hlavním větvi (trunk) je důležitý jednoduchý přesun změn do vlastní vývojové větve. Jedno z možných řešení je využití nástroje Subversion (SVN).

Jedná se o systém pro správu a verzování zdrojových kódů. Pro využití tohoto prostředku bylo vhodné do Eclipse doinstalovat plugin subclipse [22], samotný postup instalace je popsán v jednoduchém návodu [20].

Vývoj této práce probíhal také v samostatné vývojové větvi, proto bylo nezbytné porozumět alespoň základním operacím, které poskytuje SVN.

Základní pojmy:

- repository (repozitář) – umožňuje organizaci projektu a správu verzí
- branch (větev) – slouží k organizaci repository, lze si ji představit jako adresářovou strukturu
- revision (revize) – obsahuje pořadové číslo každé změny a informace o tom, co bylo změněno, kdo změnu provedl, poznámku a čas změny
- commit – odeslání změn do repository
- merge – sloučení změn z jedné větve do jiné větve, lze určit rozsah změn, a to volbou intervalu revizí

Vždy po vytvoření určité funkční části zdrojového kódu na lokálním stroji byl proveden commit do repozitáře. Zde si mohl snadno a rychle prohlédnout aktuální verzi i vedoucí práce, což značně zjednodušilo a urychlilo vlastní vývoj.

Naopak pokud provedl vedoucí práce změnu v hlavní větvi, byl provede merge pro aktualizaci zdrojového kódu ve vlastní vývojové větvi.

6 Implementace

V této kapitole je popsána samotná implementace paralelní verze Verse serveru. Aplikace je naprogramována v jazyce C a z knihovny OpenSSL je využita implementace protokolu TLS.

Aby zůstala zachována struktura a přehlednost zdrojového kódu, byly přidány dva nové zdrojové soubory – *vs_master_server.c* (pro zdrojové kódy funkcí, které využívá server v módu master) a *vs_slave_server.c* (pro mód slave) včetně příslušných hlavičkových souborů.

Nejprve bylo potřeba modifikovat spuštění Verse serveru tak, aby měl požadovanou funkcionalitu. Ta je určena volbou z příkazové řádky. Při spuštění Verse serveru bez zadání volby zůstává jeho funkce stejná jako stávající verze Verse serveru. Pokud je potřeba spustit server v takzvaném módu master, spustí se server s volbou „-m“. Pokud je vyžadován mód slave, přidá se volba „-s“, za kterou je potřeba zadat IP adresu master serveru (parametr), k němuž se má slave server připojit. Dále je možné zvolit volbu „-i“, za kterou se vkládá IP adresa, jež je v módu slave nabízena klientům (pokud parametr není zadán, použije se defaultní adresa nastavená ve funkci *vs_load_default_values()*). Jako poslední přidaná volba je „-r“, která nastavuje, zda bude server v módu master zapojovat i sám sebe do rozhodnutí, kam přeposílat klienty.

Pro základní parsování voleb a parametrů je využito funkce *getopt()*, která zpracovává volby a parametry zadané při spuštění a podle kterých je server spuštěn v určitém módu. Pokud by se chtěl uživatel seznámit se všemi možnostmi spuštění serveru, je po zadání volby „-h“ zajišteno vypsání náledující tabulky s popisem volby a případně příslušného parametru:

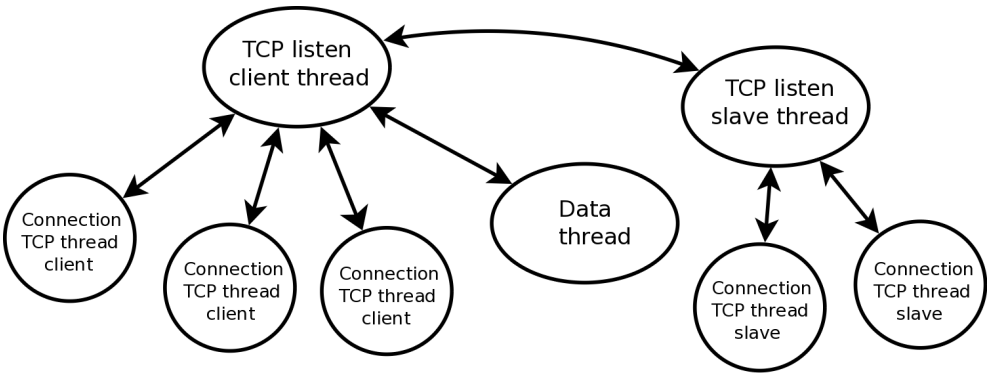
Options:

-h	display this help and exit
-c config_file	read configuration from config file
-s IP_master_server	run server as slave connecting to master server
-i IP_address	IP address which is offer to client
-m	run server as master server
-r	default enable to master recieve client
-d debug_level	use debug level [$none \mid info \mid error \mid warning \mid debug$]

6.1 Master mód

Po spuštění serveru v módu master zůstává zachována původní fukce Verse serveru. Server vytvoří datové vlákno pro zpracování příkazů z front a v hlavním vláknu vytvoří nekonečnou smyčku, ve které naslouchá na zvoleném portu (aktuálně port 12345) pro požadavky na připojení klientů.

V tomto módu je zapotřebí navíc zajistit možnost připojení ostatních slave serverů. Nejprve je vytvořeno za pomoci funkce *pthread_create()* nové vlákno, ve kterém je podle zvolené verze IP protokolu vytvořen voláním funkce *socket()* soket. Tento nový soket slouží pro komunikaci se slave servery. Následně je vytvořena nekonečná smyčka, v níž se zahájí naslouchání na zvoleném portu (aktuálně 4000) pro požadavky na připojení slave serverů. Schéma vytvoření soketu, včetně použití zabezpečeného protokolu, je znázorněno na obrázku 7. Tento soket je následně funkcí *fcntl()* nastaven do neblokovacího režimu.

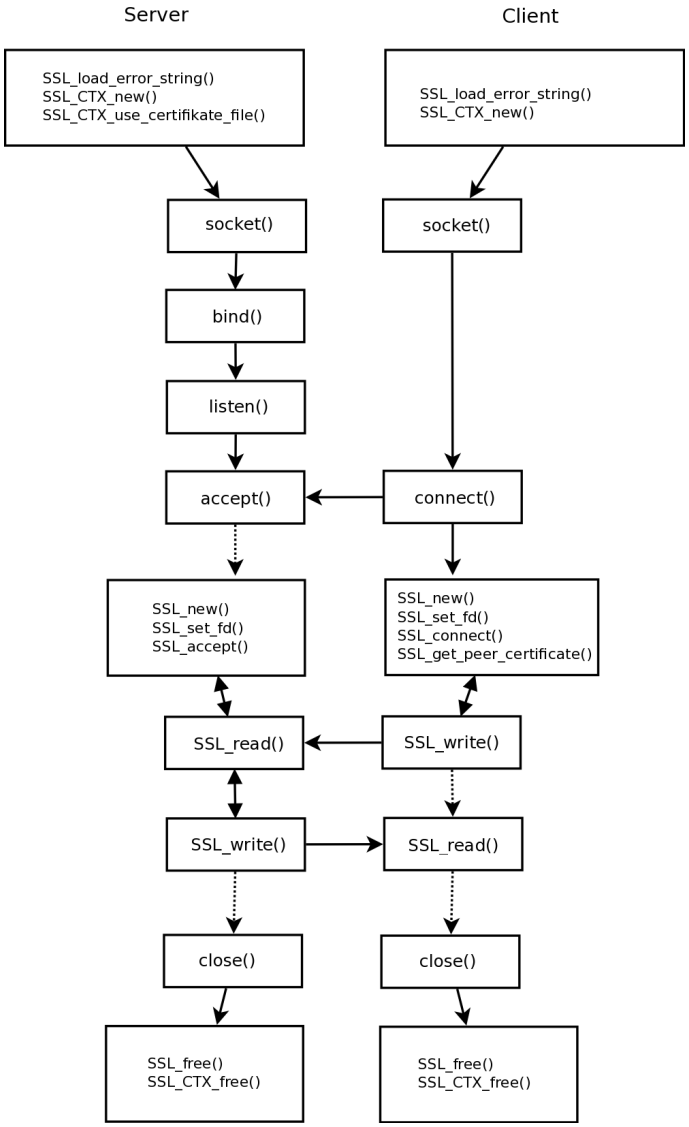


Obrázek 6: Vlákna master serveru

Teprve v tomto vláknu se alokuje potřebná paměť pro strukturu MASTER_CTX jež je popsána v časti 6.1.4, včetně paměti pro pole struktur SLAVE_INFO. Struktura SLAVE_INFO je popsána v části 6.1.5. Následně dojde k inicializaci struktury MASTER_CTX. Funkcí *vs_master_init_ctx()* jsou nastaveny výchozí hodnoty pro zabezpečené připojení a za pomoci systémového volání *listen()* server zahájí naslouchání na zvoleném portu. Dále je v každé struktuře v poli struktur SLAVE_INFO nastaven atribut *is_free* na konstantu YES, která označuje, že je tento prvek pole volný pro nové připojení slave serveru.

6.1.1 Nastavení zabezpečeného spojení

Dále je potřeba zajistit úvodní nastavení pro možnost navázání zabezpečeného spojení se slave servery. V této práci je využito zabezpečení pomocí knihovny OpenSSL [6].



Obrázek 7: Schéma programování soketů s využitím SSL funkcí [12]

Samotné nastavení je realizováno ve funkci *vs_master_init()*, která slouží mimo jiné také pro vytvoření komunikačního soketu. Zde jsou volány funkce v tomto pořadí:

- *SSL_library_init()* – zaregistruje dostupné SSL/TLS šifry
- *SSL_load_error_string()* – zajišťuje poskytování chybových zpráv
- *OpenSSL_add_all_algorithms()* – přidá všechny podporované šifrovací algoritmy do tabulky (sloužící pro volbu šifrování)
- *SSL_CTX_new(TLSv1_server_method())* – vytvoří nový objekt SSL_CTX jako rámec pro vytvoření SSL/TLS spojení, vstupním parametrem je funkce, jež určuje

že se aplikace bude chovat jako server a podporuje Transport Layer Security ve verzi 1

- *SSL_CTX_use_certificate_file()* – načte certifikát uložený v souboru (cesta k certifikátu je zadána jako vstupní parametr)
- *SSL_CTX_use_PrivateKey_file()* – načte soukromý klíč (cesta k certifikátu je zadána jako vstupní parametr)
- *SSL_CTX_check_private_key()* – kontrola privátního klíče

Po úspěšném provedení těchto funkcí je server připraven na příjem slave serverů pomocí zabezpečeného spojení. U každé funkce je kontrolována (tak jako v celém projektu) návratová hodnota. Pokud nastala chyba při vykonávání jedné z funkcí, je chyba vypsána na výstup a funkce *vs_master_init_ctx()* vrací hodnotu -1, která indikuje chybu. Pokud proběhlo vše v pořádku, tato funkce vrací hodnotu 0.

6.1.2 Přijetí slave serveru

V tuto chvíli je master server schopen přijmout připojení jak od klientů, tak od slave serverů. Požadavek na spojení je v obou případech detekován pomocí systémového volání *select()*. Master server je schopen přijmout připojení od slave serveru pouze v případě, je-li alespoň jedna struktura SLAVE_INFO volná (atribut *is_free* má hodnotu YES). Pokud je struktura v poli volná, je vytvořeno nové připojení (*accept()*), navázáno zabezpečené spojení (*SSL_accept()*) včetně TLS Handshaku, jenž je znázorněn na obrázku 4, a zároveň je nastaven atribut *is_free* na konstantu NO.

Pokud není master server schopen přijmout nové připojení od slave serveru (například pokud je již připojen maximální počet slave serverů), přesto přijme příchozí TCP spojení (pro obsloužení systémového volání), které je následně ihned ukončeno funkcí *close()*.

V případě, že master server detekuje přerušení komunikace nebo ukončení spojení se slave serverem, je atribut *is_free* ve struktuře pro daný server nastaven opět na YES a master server je připraven pro připojení dalšího slave serveru.

6.1.3 Přijetí klienta na straně master serveru

Server v módu master naslouchá na určitém portu pro požadavky klientů na připojení. Pokud se klient pokusí připojit, naváže se zabezpečené spojení a klient je požádán o zadání přihlašovacích údajů. Po úspěšné autentizaci klienta dochází k vyjednávání

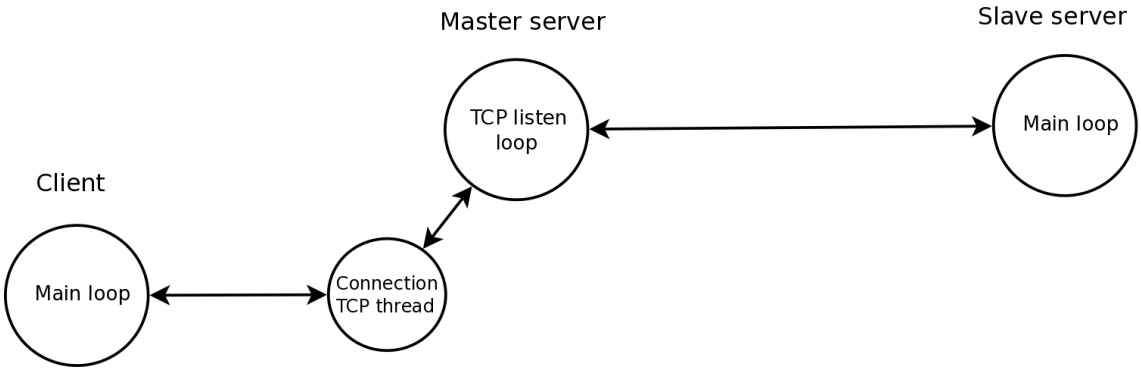
URL mezi master serverem a klientem. Součástí výsledného URL je IP adresa a číslo portu. Tuto adresu a port použije klient k návázání datového UDP spojení se serverem. Během této fáze vyjednávání se je volána funkce *vs_master_negotation()*.

V této funkci se jednoduše prochází pole struktur SLAVE_INFO a zjišťuje se, kolik je připojených slave serverů a kolik klientů bylo na který slave server přesměrováno. Podrobněji je problematika výběru vhodného slave serveru popsána v kapitole 6.2. Pokud doposud není připojen žádný slave server, master server přijme automaticky klienta sám na sebe.

V případě, že je vybrán vhodný slave server, dojde mezi master serverem a slave serverem k výměně zpráv, jenž mají strukturu popsanou v části 6.4:

- master server odešle zprávu ve které zasílá slave serveru informace o klientovi: cookie (jak master serveru tak klienta), user id a avatar id
- pokud je slave server schopen přijmout klienta, potvrdí master serveru obě cookie a ve zprávě pošle IP adresu včetně portu, na kterém očekává do 30 vteřin připojení od klienta
- master server zkontroluje zprávu (cookie), pokud je vše v pořádku, vloží do zprávy o vyjednání URL klientovi IP adresu slave serveru

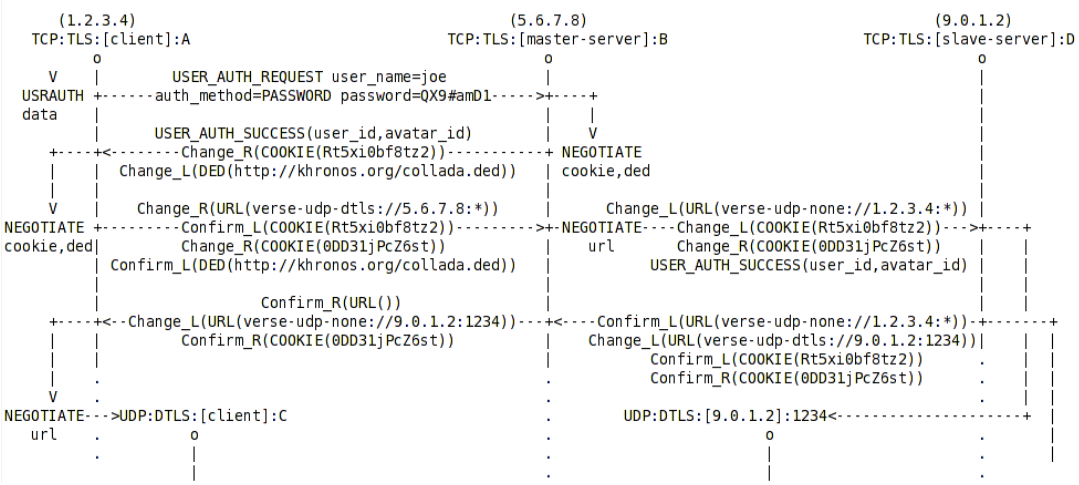
Pokud si oba servery mezi sebou vymění korektní údaje, je na slave serveru vytvořeno UDP vlákno, ve kterém server čeká na spojení s klientem. Tím, že byl klient již ověřen na master serveru, je vynechána celá TCP komunikace mezi klientem a slave serverem a klientovi je umožněno ihned zahájit samotnou UDP komunikaci.



Obrázek 8: TCP spojení klienta a master serveru, komunikace mezi master serverem a slave serverem

Důležité je, že na master serveru dochází jak k ověření klienta, tak k vygenerování user id i avatar id, která jednoznačně identifikují klienta na serveru. Pokud je klient

ověřen, lze ho bezpečně přesměrovat na slave server. Chování slave serveru je popsáno v části 6.3. Dojde-li během komunikace mezi servery k chybě, nebo není slave server schopen přijmout klienta, automaticky nabídne master server klientovi vlastní IP adresu a port, na němž očekává připojení. Pokud byl již tedy klient autentizován, nestane se, že by byl odmítnut.



Obrázek 9: Předání zpráv mezi master serverem a slave serverem

6.1.4 Struktura MASTER_CXT

Nová funkcionalita přinesla nutnost uchovávat nová data. Proto byla navržena struktura MASTER_CTX, pro níž se alokuje paměť pouze při spuštění serveru v módu master.

struktura MASTER_CTX :

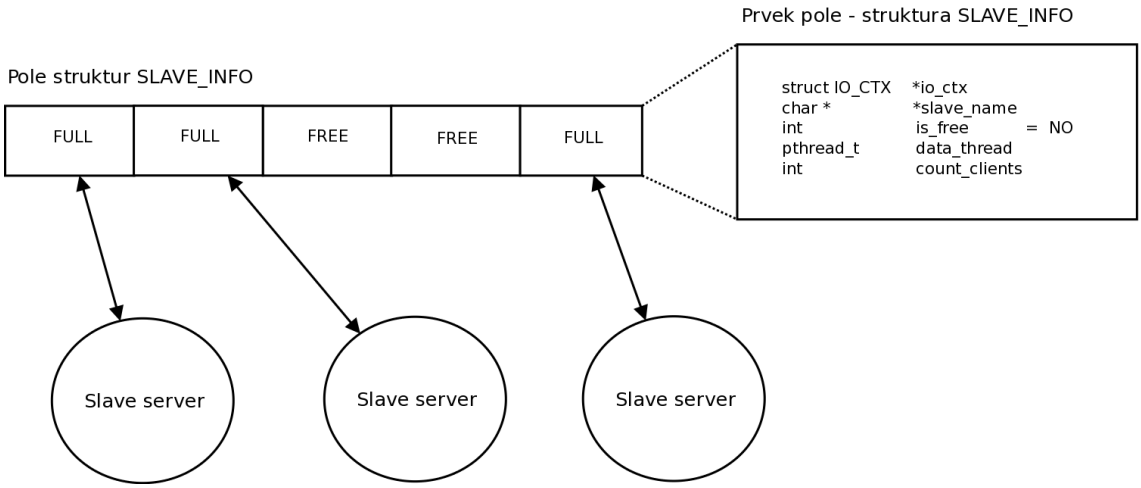
struct IO_CTX	io_ctx	informace o samotném spojení (IP adresa, port)
SSL_CTX	*tls_ctx	SSL kontext pro zabezpečené spojení
char	*hostname	název pro vlastní UDP spojení
unsigned short	max_qeue	maximální délka příchozí fronty
int	stream_protocol	verze protokolu
char	*public_cert_file	cesta k certifikátu s veřejným klíčem
char	*private_cert_file	cesta k certifikátu s privátním klíčem
int	max_slaves	maximální počet připojených slave serverů
struct SLAVE_INFO	*slave_info	pole s daty o slave serverech viz 6.1.5
socklen_t	lenght	délka struktury sockaddr_in

6.1.5 Struktura SLAVE_INFO

Je potřebné, aby si master server uchovával data o připojených slave serverech. Za tímto účelem byla vytvořena struktura s názvem SLAVE_INFO. Server si v módu master alokuje potřebnou pamět pro pole těchto struktur a pokud se nějaký slave server připojí, informace o tomto připojení jsou uloženy do struktury v tomto poli. Velikost pole, a tím i maximální počet připojených slave serverů, je nastaveno konstantou MAX_SLAVES_CONNECT. Každý prvek pole tedy obsahuje infromace právě o jednom připojeném slave serveru.

Infromace, které je potřeba vést o slave serverech:

struktura SLAVE_INFO :		
struct IO_CTX	io_ctx	informace o samotném spojení (IP adresa, port)
char	*slave_name	název slave serveru
int	is_free	identifikátor použitelnosti pro nové připojení
pthread_t	data_thread	identifikační číslo vlákna pro slave server
int	count_clients	počet klientů, kteří byli přesměrovány na slave server



Obrázek 10: Pole struktur SLAVE_INFO

Prvky v poli jsou dynamicky obsazovány a uvolňovány, podle aktuální situace. V případě, že master server detekuje odpojení slave serveru, je struktura uvolněna pro další příchozí spojení.

6.2 Metoda přidělování klientů

Tato práce se zaměřuje především na vzájemnou komunikaci mezi servery ve chvíli, kdy probíhá vyjednávání o URL mezi serverem a klientem. Pro rozdělení komunikační zátěže mezi připojené slave servery byl proto v této práci navržen jednoduchý mechanismus. Sofistikovanější návrh pro přesměrování klientů je popsán v kapitole 7.1.2. Do budoucna se také počítá s implementací mechanismu pro sdílení informací mezi servery nejen o klientech, ale také o samotných datech. Popis této problematiky je v kapitole 7.1.1.

Master server si u každého připojeného slave serveru uchovává informaci o počtu přesměrovaných klientů. V aktuální implementaci je slave server schopen přijmout pouze klienta, kterého k němu přesměroval master server. Při každém přepojení klienta inkrementuje master server proměnnou *count_clients* ve struktuře SLAVE_INFO.

Při rozhodování, kterému serveru se má klient přepojit, se v cyklu projde pole struktur SLAVE_INFO a zjistí se, jaký server má nejméně klientů. Ukazatel na tuto strukturu je zaznamenán a následně využit pro komunikaci mezi servery. Klient je tedy přepojen na server s nejmenším počtem připojených klientů

Volba z příkazové řádky „-r“ nastavuje, zda bude master server rozkládat zátěž mezi servery včetně sám sebe. Pokud ano, nejprve dojde k nalezení slave serveru s nejmenším počtem klientů a poté je tento počet porovnán s počtem klientů připojených k samotnému master serveru. V případě zjištění, že nejméně klientů má master server, je klient připojen k samotnému master serveru.

Pokud je server spuštěn bez volby „-r“, přeposílá master server klienty automaticky na slave server s nejmenším počtem klientů. Server je i nadále schopen přijmout klienty, například pokud doposud není připojen žádný slave server.

6.3 Slave mód

V módu slave má server jako primární úkol připojení k master serveru. V tomto případě je slave server v roli klienta, který se snaží připojit k příslušném portu na master serveru. Je proto potřeba znát IP adresu a port serveru, ke kterému se má server připojit (IP adresa je zadána za jako parametr za volbu „-s“, port je nastaven konstantou). Po inicializaci počátečních hodnot a po úspěšném navázání spojení pomocí funkce *connect()* s master serverem očekává slave server data o klientovi.

Slave server má ve své podstatě po připojení k master serveru obdobnou funkci jako klasický Verse server. Hned při spuštění také vytvoří datové vlákno, které bude zpracovávát data od klientů. Nemá pouze vlákno, jež v původní verzi sloužilo k naslouchání

pro klienty.

Z počátku implementace byla vytvořena nová struktura, která měla uchovávat potřebné údaje pro slave server. Postupem času se ovšem ukázalo, že prvotně navržená struktura pouze kopírovala stávající strukuturu vs_ctx. Z tohoto důvodu nebylo potřeba pro server v módu slave vytvářet strukturu novou.

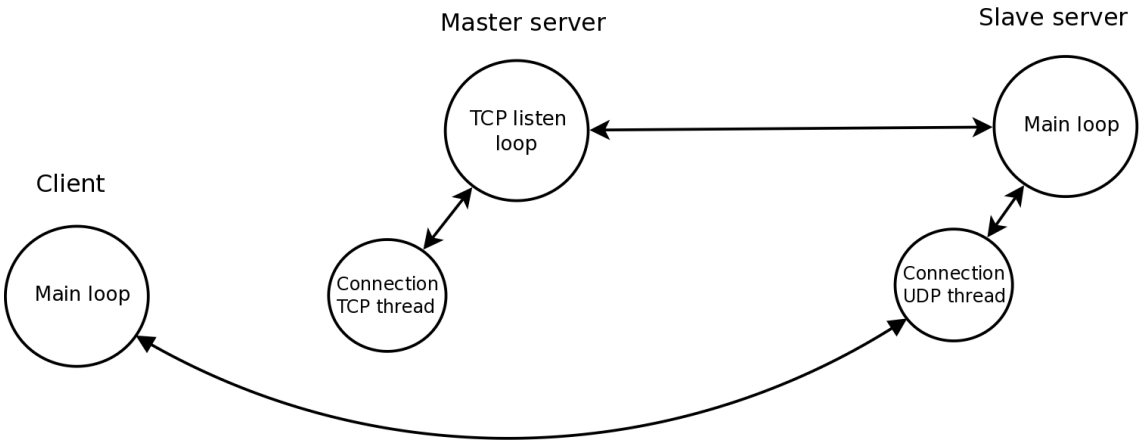
6.3.1 Komunikace s master serverem

Klient se vždy připojuje nejprve k master serveru. Ten od něj získá potřebné infromace během přihlášení a vygeneruje cookie, user id a avatar id. Tyto informace potřebuje znát i slave server. Proto bylo dílčím úkolem využít již implementovaného rozhraní pro zasílání potřebných údajů mezi servery. Stuktura zpráv je popsána v kapitole 6.4.

Po úspěšné výměně zpráv, která je znázorněna na obrázku 9, si k sobě slave server uloží klientskou cookie pro následnou autentizaci komunikace s klientem. Zároveň zná user id a avatar id, které budou potřeba pro přihlášení klienta k uzlu na serveru.

6.3.2 Přijetí klienta na straně slave serveru

V této chvíli má slave server dostatek potřebných údajů pro bezpečné přijetí klienta (především klientskou cookie). Vytvoří se tedy vlákno, jenž slouží přímo pro vlastní UDP komunikaci.



Obrázek 11: Přepojení klienta z TCP spojení s master serverem na UDP spojení se slave serverem

Jak bylo popsáno v kapitole 6.1.3, není nutné na slave serveru opět ověřovat klienta. Tento proces by byl nejen zbytečný, ale také neefektivní.

V původním návrhu klient komunikuje se serverem pomocí dvou vláken. Nejprve se pomocí TCP spojení klient ověří a poté dojde ke vzájemné dohodě o navázání spo-

lečného UDP spojení. Server tedy vytvoří vlákno, ve kterém čeká na UDP připojení klienta a i když se klient připojí a probíhá zde vlastní datová komunikace, TCP spojení je stále aktivní.

V této práci je klient připojen ke slave serveru pouze pomocí UDP spojení. Mohlo by se zdát, že je tím omezena nějaká funkcionalita na straně klienta nebo serveru, ale po prostudování vlastní komunikace mezi serverem a klientem a po konzultaci s vedoucím práce bylo zjištěno, že se TCP spojení sice stále udržuje, ale žádná data mezi koncovými body v tomto vlákně vyměňována nejsou, a ani se s tím do budoucího rozšíření nepočítá. Proto je na straně serveru komunikační UDP vlákno plně dostačující.

6.4 Struktura a předávání zpráv

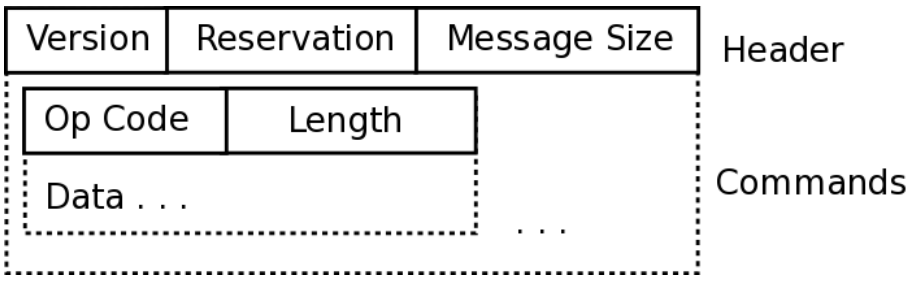
Po návázání zabezpečeného TCP spojení se mezi servery posílají zprávy, které obsahují data a příkazy potřebné k domluvě o přepojení klienta. Každá zpráva má svoji hlavičku, za kterou následují příkazy [9].

Hlavička obsahuje:

- *Version* – verzi protokolu
- *Reservation* – rezervaci pro další položky
- *Message Size* – velikost zprávy včetně hlavičky v bytech

Příkazy se skládají z:

- *Op Code* – specifikuje další strukturu příkazu
- *Length* – délka příkazu včetně *Op Code* a *Length* v bytech



Obrázek 12: Struktura zprávy

6.4.1 Odeslání zprávy

Zprávu je potřeba vytvořit, poté zabalit do zásobníku a odeslat. Nejprve se postupně do zprávy vloží požadované příkazy a data. Pro označení posledního příkazu je vložen příkaz CMD_RESERVED_ID. Dále jsou příkazy zabaleny funkcí *v_pack_stream_system_commands()* do zásobníku (bufferu). Hlavička zprávy obsahuje informaci o celkové velikosti zprávy. Tato velikost je ovšem známa až po uložení všech potřebných údajů, proto se hlavička zprávy vyplňuje vždy až jako poslední. Nakonec je i hlavička zprávy přibalena funkcí *v_pack_message_header()* na začátek zásobníku, který je následně pomocí funkce *v_SSL_write()* odeslán příjemci. Ještě před odesláním je zpráva zobrazena v konzoli využitím funkce *v_print_send_message()*.

6.4.2 Příjem zprávy

Po příjmu dat se musí zpráva nejprve složit, následně zpracovat. Data jsou přijímány funkcí *v_SSL_read()*. Nejprve je voláním funkce *v_unpack_message_header()* rozbalena hlavička zprávy, následně je rozbalena i příkazová část pomocí funkce *v_unpack_message_system_commands()*. V tuto chvíli má slave server složenou celou příchozí zprávu. Ta je zobrazena funkcí *v_print_receive_message()* v konzoli. Pro zpracování zprávy je potřeba přečíst přijaté příkazy. Tento mechanismus je implementován v cyklu, kdy se prochází jednotlivé příkazy v přijaté zprávě. Postupně se načte příkaz a za ním následující data, která jsou zpracována. Příkazy jsou načítány do chvíle, kdy je načten příkaz CMD_RESERVED_ID, který značí konec. Přijatá data jsou uložena, popřípadě ověřena (například u cookie). Pokud jsou všechny přijaté informace korektní, je umožněno složit a odeslat odpověď.

7 Testování a možnosti rozšíření

Součástí zadání bylo také paralelní variantu Verse serveru otestovat a zkontntrolovat požadovanou funkcionalitu. Je možné otestovat kompletní funkcionalitu na lokálním stroji, ale pro alespoň částečné přiblížení se realitě byla použita virtualizace serveru pomocí VirtualBoxu. Celý projekt má podporu protokolu IP verze 4 i IP verze 6.

Master server byl spuštěn virtuálně, také pod operačním systémem Linux (Fedora 13 – Goddard). Na tomto operačním systému běží firewall, který je možné buď příkazem:

```
/etc/init.d/iptables stop
```

pro použití protokolu IP ve verzi 4, popřípadě pro protokol IP ve verzi 6 příkazem:

```
/etc/init.d/ip6tables stop
```

zcela vypnout (což se samozřejmě na klasickém serveru nedoporučuje), nebo iptables nastavit příkazy:

```
iptables -I INPUT -p tcp -- dport 12345 -j ACCEPT
iptables -I INPUT -p tcp -- dport 4000 -j ACCEPT
iptables -I INPUT -p udp -- dport 20000:20009 -j ACCEPT.
```

Pro nastavení ip6tables jsou příkazy obdobné:

```
ip6tables -I INPUT -p tcp -- dport 12345 -j ACCEPT
ip6tables -I INPUT -p tcp -- dport 4000 -j ACCEPT
ip6tables -I INPUT -p udp -- dport 20000:20009 -j ACCEPT.
```

Popis nastavení iptables (ip6tables) je v souboru README, který je součástí výsledného projektu na přiloženém CD.

Je vyžadováno, aby byl jak server, tak klient vždy spuštěn z hlavního adresáře projektu (v projektu jsou využity relativní cesty). Server v módu master je spuštěn příkazem:

```
./bin/verse_server -m,
```

popřípadě je možné přidat volbu „-r“, pro funkci, jenž je popsána v kapitole 6.2

```
./bin/verse_server -m -r.
```

Po spuštění rerveru v módu slave je vždy potřeba zadat IP adresu severu, k němuž se připojuje:

```
./bin/verse_server -s (IP adresa),
```

také je možné konfigurovat IP adresu rozhraní, na kterém bude slave server očekávat připojení klientů:

```
./bin/verse_server -s (IP adresa serveru) -i (IP adresa pro klienty).
```

Parametr „IP adresa pro klienty“ je doporučeno vždy uvádět. Je možné, že slave server očekává připojení na jiném rozhraní než komunikuje s master serverem.

Po spuštění serveru v módu master je vypisována v časových intervalech jedné vteřiny zpráva upozorňující na očekávené připojení klienta i slave serveru. V případě připojení serveru je vypsána IP adresa slave serveru a port, z něhož požadavek přišel.

Hlavní činitel komunikace je klient. Při pokusu o připojení (vždy na master server) je požádán o zadání uživatelského jména a hesla. Po úspěšné autentizaci lze v konzoli sledovat výměnu zpráv mezi master serverem a slave serverem. Poté komunikuje klient již pouze se slave serverem.

Pokud dojde k ukončení master serveru, slave server detekuje ztrátu spojení. Ukončí soket, na kterém probíhala společná komunikace, a dále slouží k obsluze doposud připojených klientů. Není ovšem již schopen příjmu nových klientů.

Funkcionalita serverů byla otestována s maximálním počtem slave serverů. V případě pokusu o připojení dalšího slave serveru je požadavek akceptován a následně toto spojení ihned uzavřeno.

Otestováno bylo také uvolňování prvků v poli poté, co master server detekoval ztrátu spojení se slave serverem. Po odpojení jednoho slave serveru bylo umožněno připojení dalším serverům.

Využití výkonu procesoru a paměti zůstávalo na stejné úrovni.

7.1 Možnosti rozšíření

Tato práce rozšiřuje funkci Verse serveru. Během implementace byl vedoucím práce určen cílový bod, do kterého bylo potřeba dojít. Dále ovšem zůstaly určité nedostatky, které bude potřeba v budoucnu zlepšovat. V této kapitole jsou stručně popsány dva nejdůležitější body. Řešení těchto problémů by mohlo být zadáním dalších univerzitních prací.

7.1.1 Sdílení dat mezi servery

Aktuálně jsou klienti přepojováni na jiný server za účelem rozložení zátěže, ale změnu dat, kterou klienti provedou na určitém serveru, se již klienti připojeni na jiný server nedozvědí. Je potřebné vhodně navrhnout a implementovat sdílení dat a informací o klientech mezi servery. Ale nejen to. Již při připojení slave serveru k master

serveru by si měly mezi sebou vyměnit informace o aktuálních datech. Bude nutné sdílet informace o datech centrálně právě na master serveru. Proto se předpokládá využití master serveru především za účelem sdílení aktuálních dat a distribuce o jejich změnách. Dále bude vhodné centrálně udržovat přehled o klientech a jejich připojování ke slave serverům včetně jejich odpojování. Samotnou komunikaci mezi klienty a servery by obstarávaly slave servery.

Za tímto účelem by bylo vhodné vytvořit vždy mezi master a slave serverem ještě další vlákno, ve kterém by probíhala samotná komunikace o změně dat. Ovšem samotný mechanismus pro sdílení dat je netriviální záležitost.

7.1.2 Návrh nové metody přidělování klientů

V této práci jsou klienti přepojováni na server, který má nejméně klientů. Tento způsob také nemusí být ideální, protože je posuzující metrika vypočítána pouze z jedné hodnoty (jednoduchého mechanismu). Tuto problematiku by bylo nejvhodnější řešit podobně jako je to u směrovacích protokolů. Bylo by vhodné posuzovat více faktorů, jako jsou například: rychlost připojení slave serverů a doba odezvy, ale také geografická poloha serverů, kterou je možné u IP adres ve verzi 4 zjistit (více informací o geolokaci pomocí IP verze 4 je popsáno v dokumentu [13]). Dále bude potřeba navrhnout a implementovat vhodný algoritmus, který vyhodnotí konkrétní faktory a určí, jaký server bude pro klienta nejvhodnější.

8 Závěr

Tato bakalářská obsahuje popis modifikace stávající verze Verse serveru, za účelem rozložení komunikační zátěže. Práce se zabýva především rozšíření funkcionality Verse serveru o možnost spuštění serveru v požadovaném módu. Po zhodnocení aktuální verze protokolu Verse a samotného Verse serveru (kapitola 2) vyvstaly určité motivační požadavky na tuto práci (kapitola 3).

Nová verze protokolu Verse je robustní síťový protokol, určený především pro real–timové sdílení dat v grafických aplikacích. Z tohoto důvodu jsou na Verse server kladeny vysoké nároky na obsluhu klientů. Jednou neefektivní možností je neustálé zvyšování výpočetního výkonu na straně serveru, druhou možností, jež se zároveň stala cílem této práce, je rozložení komunikační zátěže mezi více serverů a tím paralelně zpracovávat požadavky.

V práci je kladen důraz na zabezpečenou komunikaci mezi všemi koncovými uzly. Tohoto požadavku bylo dosaženo využitím protokolu TLS dostupného v open source knihovnách OpenSSL (kapitola 4).

Po úvodním získání potřebných zkušeností v programování jazyce C pod operačním systémem Linux, zabezpečené komunikace mezi sokety a samotném vývojovém prostředí (kapitola 5), je popsána samotná implementace obou nových módů Verse serveru, jež dostaly pracovní označení *master* a *slave* (kapitola 6).

Verzi Verse serveru, jež je přiložena na CD a která vznikla během této práce, je možné spustit v požadovaném módu. Server v módu master slouží jako centrální prvek, k němuž se připojují klienti i slave servery. Jeho funkce je především přesměrování klientů na vhodý slave server. V módu slave se server připojuje k master serveru a dále slouží především pro zpracování požadavků od klientů, kteří byli přesměrováni z master serveru. Oba módy byly otestovány na virtualizovaném serveru (kapitola 7). Po závěrečném testování byla za pomoci pluginu v Eclipse vygenerována odpovídající dokumentace.

Obecně se dá říci, že vývoj nového komunikačního protokolu je komplikovaný a náročný proces. Z počátku je potřeba dbát na kvalitní návrh, poté zvolit vhodnou implementaci, ve které se mnohdy objeví nějaký problém. I z tohoto důvodu jsou viditelné určité nedostatky, které bude v budoucnu potřeba zlepšovat (kapitola 7.1). Zde se jedná především o udržování aktuálních dat mezi všemi připojenými servery.

Nová verze Verse protokolu si klade za cíl být efektivní a robustní protokol. Tato práce, jež je implementována v jazyce C a podporuje protokol IP verze 4 i verze 6, se doufejme jednou stane jedním z použitelných dílů, ze kterých se protokol Verse skládá.

Seznam literatury

[1] STONES, R., MATHEW, N. *Linux: začínáme programovat.* Přeložil Jan Škvařil. Praha: Computer Press, 2000. ISBN 80-7226-307-2.

[2] STEVENS, W., *Programování sítí operačního systému UNIX.* Přeložila Jana Viktorinová. 1. vydání. Brno: SCIENCE, 1994. ISBN 80-901475-3-4.

[3] BALLARD, Kenneth. IBM [online]. 2004 [cit. 2011-05-04]. *Secure programming with the OpenSSL API.* Dostupné z WWW: <http://www.ibm.com/developerworks/linux/library/l-openssl.html>.

[4] HERAUT, P., *Učebnice jazyka C,* 3. vydání. České Budějovice: KOPP, 1998. ISBN 80-85828-21-9.

[5] HERAUT, P., *Učebnice jazyka C 2. díl*, 1. vydání. České Budějovice: KOPP, 1998. ISBN 8-85828-50-2.

[6] OpenSSL [online]. 2009 [cit. 2011-05-01]. *OpenSSL: The Open Source toolkit for SSL/TLS.* Dostupné z WWW: <http://www.openssl.org>.

[7] Root [online]. 2003 [cit. 2011-04-21]. *Seriál Sokety a C/C++.* Dostupné z WWW: <http://www.root.cz/serialy/sokety-a-cc>.

[8] DIERKS, T., ALLEN, C. RFC editor [online]. January 1999 [cit. 2011-04-28]. *TLS Protocol.* Dostupné z WWW: <http://www.rfc-editor.org/rfc/rfc2246.txt>.

[9] HNÍDEK, Jiří., *Síťový protokol pro grafické aplikace.* Liberec, 2010. 171 stran. Disertační práce. Technická univerzita v Liberci.

[10] FREIER, A., et. al. RFC editor [online]. April 2011 [cit. 2011-05-17]. *The SSL Protocol Version 3.0.* Dostupné z WWW: <http://www.ietf.org/id/draft-mavrogiannopoulos-ssl-version3-04.txt>.

[11] BLAKE-WILSON, S., et. al. RFC editor [online]. June 2003 [cit. 2011-04-29]. *Transport Layer Security (TLS) Extensions.* Dostupné z WWW: <http://www.rfc-editor.org/rfc/rfc3546.txt>.

[12] DOSTÁLEK, Libor. Cpress [online]. 2002 [cit. 2011-05-04]. *SSL.* Dostupné z WWW: <http://www.cpress.cz/knihy/tcp-ip-bezp/CD-ssl/ssl2.htm>.

[13] POSTEL, J. RFC editor [online] September 1981. [cit. 2011-04-26]. *Transmission Control Protocol.* Dostupné z WWW: <http://www.ietf.org/rfc/rfc793.txt>.

[14] RESCORLA, E., MODADUGU, N. RFC editor [online] April 2006 [cit. 2011-04-26]. *Datagram Transport Layer Security.* Dostupné z WWW: <http://www.ietf.org/rfc/rfc4347.txt>.

[15] BERNERS-LEE, T., et. al. RFC editor [online] December 1994 [cit. 2011-04-27]. *Uniform Resource Locators (URL).* Dostupné z WWW: <http://www.ietf.org/rfc/rfc1738.txt>.

[16] LINFO [online]. 2005 [cit. 2011-05-14]. *BSD License Definition.* Dostupné z WWW: <http://www.linfo.org/bsdlicense.html>.

[17] MUIR, J. A., OORCHOT, P. C. Carleton [online]. April 2006 [cit. 2011-05-14]. *Internet Geolocation and Evasion.* Dostupné z WWW: <http://www.ccsl.carleton.ca/ jamuir/papers/TR-06-05.pdf>.

[18] POSTEL, J. RFC editor [online] August 1980 [cit. 2011-05-15] *User Datagram Protocol.* Dostupné z WWW: <http://tools.ietf.org/rfc/rfc768.txt>.

[19] Eclipse [online]. 2011 [cit. 2011-05-15]. Dostupné z WWW: <http://www.eclipse.org/>.

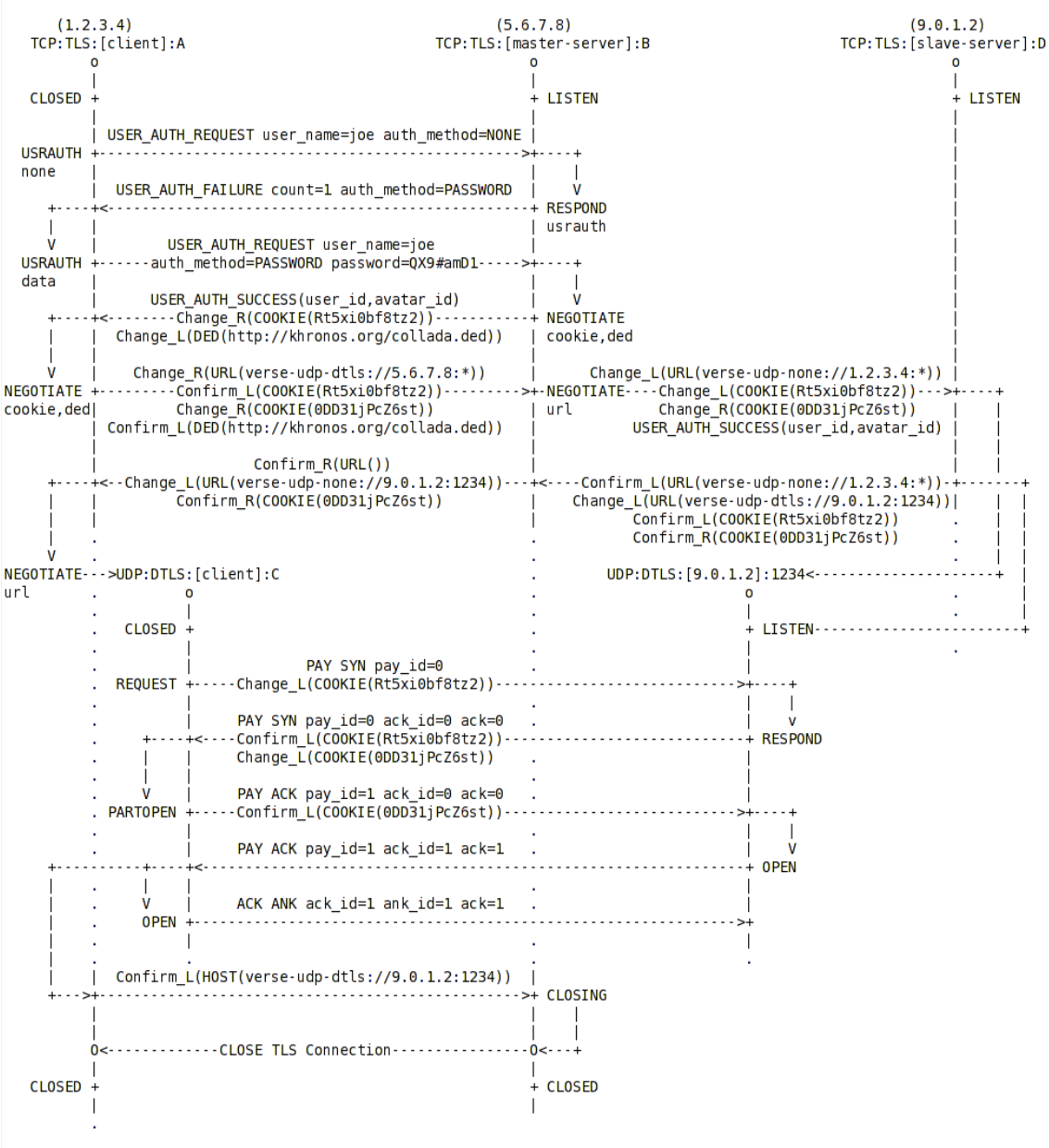
[20] Eclipse instalace [online]. 2010 [cit. 2011-05-15]. Dostupné z WWW: <http://java.vse.cz/Java/EclipseInstalace>.

[21] VPN Protocols [online]. 2010 [cit. 2011-05-16]. Dostupné z WWW: <http://www.vpnc.org/vpn-standards.html>.

[22] Subclipse [online]. 2009 [cit. 2011-05-16]. Dostupné z WWW: <http://subclipse.tigris.org>.

Přílohy

Příloha A - Komunikace



Obrázek 13: Kompletní znázornění komunikace mezi klientem, master serverem a slave serverem

Příloha B - Struktura příkazů

Pro dohadování se mezi master serverem a slave serverem jsou posílány ve zprávách příkazy. Potřebných příkazů je celkem pět a mají následující význam:

- Change_L – odesílatel navrhuje nastavení vlastnosti na své straně
- Change_R – odesílatel navrhuje nastavení vlastnosti na straně příjemce
- Confirm_L – příjemce potvrzuje odesílateli, že si může nastavit danou vlastnost
- Confirm_R – příjemce potvrzuje, že si nastavil danou vlastnost
- USER_AUTH_SUCCESS – slouží k odeslání user id a avatar id k master serveru, zároveň je potvrzena úspěšná autentizace klienta na straně master serveru

Za indentifikátorem vlastnosti se nachází v případě příkazů *Change_L* a *Change_R* navrhovaná hodnota:

- URL – dohadované URL
- Cookie – cookie

V případě příkazů *Confirm_L* a *Confirm_R* se jedná o hodnoty potvrzované.

Příloha C - Přiložené CD

Na přiloženém CD se nachází:

- bakalářská práce ve formátu pdf
- komprimovaný soubor s paralelní verzí Verse serveru (parallel-server.zip)
- příslušná dokumentace