UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Test komunikace dvou programů komunikujících TCP protokolem spuštěných na win32 platformě a na linuxové platformě

Michal Poul

Bakalářská práce
2012

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2011/2012**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal Poul**
Osobní číslo:	**I09238**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Test komunikace dvou programů komunikujících TCP protokolem spuštěných na win32 platformě a na linuxové platformě.**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Student vyrobí jednoduché programy TCP klient a TCP server pro win32 platformu a linuxovou platformu. Nejlépe, aby programy byly portovány (technologie QT nebo i jiné technologie, ale je nutné vyrobit podobné, identické programy). Programy TCP klient a TCP server budou komunikovat s malou zátěží, s větší zátěži a s vysokou zátěži co se týká množství přenesených dat. Student v tabulkách a graficky v dostupných nástrojích bude analyzovat a srovnávat rychlost komunikace programů bežících na různých platformách. Programovací jazyk je na výběru studenta.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Velký průvodce protokoly TCP/IP a systémem DNS, L. Dostálek, A.Kabelová, Computer Press
Mistrovství C++, Stephen Prata, Computer Press
OpenGL Průvodce programátora, D.Shreiner, M.Woo, J.Neider, T.Davis, Computer Press
www.MSDN.com

Vedoucí bakalářské práce: **Ing. Jaroslav Štroch**
Katedra informačních technologií

Datum zadání bakalářské práce: **16. prosince 2011**
Termín odevzdání bakalářské práce: **11. května 2012**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 30. března 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 11. 5. 2012 Michal Poul

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce panu Ing. Jaroslavu Štrochovi za jeho odbornou pomoc a cenné rady, které mi pomohly při zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu během absolvování studia.

Anotace

Cílem této práce je otestovat rozdíly v komunikaci protokolu TCP na Windows a Linux platformách. Aplikace klient a server jsou napsány v jazyce C++ s použitím Qt framworku pro snadnou portaci na jiné platformy a tudíž zajištění identických aplikací. Komunikace je testována s malou, vetší a vysokou zátěží, výsledky jsou vyhodnoceny a graficky znázorněny.

Klíčová slova

Qt, TCP, C++, testování komunikace

Title

Communication test between two programs which are communicating through TCP protocol running on Win32 and Linux platform

Annotation

The objective of this work is to test the differences in communication of TCP protocol on Windows and Linux platforms. Client and server applications are written in C++ with using the Qt framework for easy porting to other platforms and to ensure identical applications. Communication is tested with a small, a larger and high load. The results are evaluated and graphically displayed.

Keywords

Qt, TCP, C++, communication test

Obsah

Seznam zkratek

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
VoIP	Voice over Internet Protocol
MTU	Maximum transmission unit
OS	Operating system
MS	Microsoft

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem této práce je otestovat TCP komunikaci mezi Windows a Linux platformou. Pro tvorbu aplikace klient a server byl zvolen programovací jazyk C++ s použitím Qt frameworku, který umožnuje snadnou portaci aplikací mezi platformami a tím zajištění podobných aplikací.

Obsahem prvních dvou kapitol je teorie o testovaném protokolu TCP a seznámení s technologiemi použitých při tvorbě testovacích aplikací.

Tématem kapitoly čtyři je popis aplikací klient a server. Seznámení se způsobem, jak je řešena samotná komunikace pomocí protokolu TCP a seznámení s třídami obou aplikací.

V kapitole pět a šest je popsáno prostředí testování, tedy jaké operační systémy byly použity na jednotlivých platformách. Dále průběh testování a způsob zpracování výsledků.

Předposlední kapitola obsahuje výsledky testování komunikace.

2 Protokol TCP

TCP je síťový protokol patřící do transportní vrstvy referenčního modelu ISO/OSI. Pro porovnání oproti protokolu IP z nižší vrstvy, který se stará o přenos dat mezi libovolnými počítači v internetu, se TCP stará o přenos mezi konkrétními aplikacemi na daných počítačích.

TCP je spojově orientovaný protokol, tzn. vytváří mezi dvěma aplikacemi spojení, po jehož dobu mohou obě strany nezávisle v obou směrech přenášet data. Přenášené bajty jsou číslovány. Pokud se nějaké data ztratí nebo poškodí, jsou opětovně poslána. Integrita dat je zabezpečena kontrolním součtem.

Aplikace komunikující přes protokol TCP se tedy nemusejí starat o to, jestli data byla ztracena nebo poškozena, ale musejí brát ohled na inteligentní útočníky, kteří mohou data změnit a následně i přepočítat kontrolní součet. Obrana proti těmto útokům je například šifrovací protokol SSL.

Sousedící protokol na transportní vrstvě je UDP. Ten oproti TCP nezaručuje doručení dat a je značně jednoduší. TCP lze také nazvat jako spolehlivý a UDP jako nespolehlivý protokol. Každý protokol má své specifické využití. TCP se používá u aplikačních protokolů, kde není přípustná ztráta nebo poškození dat, jako jsou např. HTTP, FTP. Naopak protokol UDP je vhodný u tzv. real-time aplikací, kde jde o včasné doručení a opakovat odeslání kvůli ztrátě nemá smysl. Jako příklady používající protokol UDP lze uvést stream videa, online hry a VoIP.

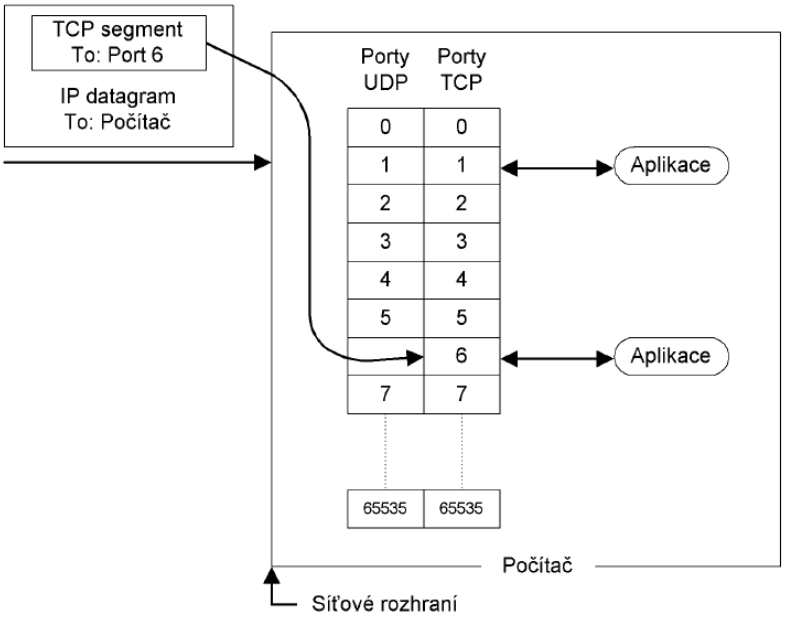
(1)

2.1 TCP port

Protokol IP používá pro identifikaci IP-adresu, což je adresa síťového rozhraní na daném počítači. Adresa protokolu TCP se nazývá port, je to dvojbajtové číslo, tudíž může nabývat hodnot 0 až 65535. U čísel portů se často udává za lomítkem typ protokolu, tedy TCP nebo UDP. Oba protokoly mají vlastní sadu portů, takže např. port 80/tcp nemá nic společného s portem 80/udp. Svoje číslo portu má jak klient, tak server. U serveru by mělo číslo portu být všeobecně známo, aby se klienti mohli připojit. Naopak u klienta to není nutné a většinou je požádáno konkrétní OS o přidělení volného portu.

Porty menší než 1024 jsou označovány jako privilegované, tzn. k jejich použití je potřeba administrátorské oprávnění. Privilegované porty jsou nejčastěji používány servery.

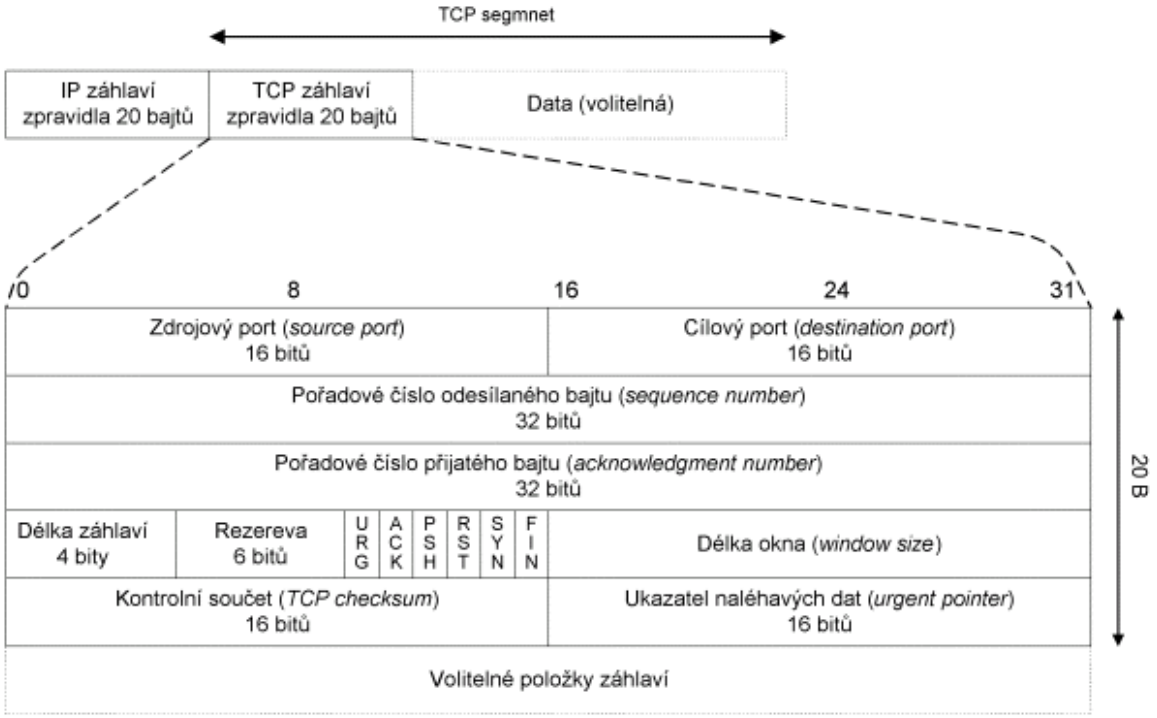
(1)



Obrázek 1 – TCP porty

Zdroj: (1)

2.2 TCP segment



Obrázek 2 – TCP segment

Zdroj: (1)

TCP segment je základní jednotka přenosu TCP protokolu. Lze se potkat i s pojmenováním TCP paket. Např. aplikace potřebuje přenést 3 GB dat, ale TCP segment může být maximálně dlouhý 65535 bitů mínus délka TCP-záhlaví. Z tohoto důvodu se musí 3 GB rozdělit na segmenty. Proto se přeneseně místo TCP paket říká TCP segment. (1)

Vysvětlení pojmů k obrázku (Obrázek 2 – TCP segment):

- **zdrojový port** (source port) – port odesílatele segmentu,
- **cílový port** (destination port) – port adresáta segmentu,
- **pořadové číslo odesílaného bajtu** – pořadové číslo prvního bajtu TCP-segmentu v toku dat (32 bitů), číslování začíná od náhodně zvoleného čísla,
- **pořadové číslo přijatého bajtu** – číslo následujícího bajtu, který je příjemce připraven přijmout, příjemce potvrzuje, že přijal vše do tohoto bajtu,
- **délka záhlaví** – vyjadřuje se v násobcích 32 bitů,
- **délka okna** – přírůstek pořadového čísla bajtu, který bude ještě příjemcem akceptován,
- **ukazatel naléhavých dat -** může být nastaven pouze v případě, že je nastaven příznak URG. Přičte-li se tento ukazatel k pořadovému číslu odesílaného bajtu, pak ukazuje na konec úseku naléhavých dat,
- **URG** – TCP-segment nese naléhavá data,
- **ACK** – TCP-segment má platné pořadové číslo přijatého bajtu,
- **PSH** – TCP-segment nese aplikační data,
- **RST** – odmítnutí spojení,
- **SYN** – odesílatel začíná s novou sekvencí číslování, tedy TCP-segment nese pořadové číslo prvního odesílaného bajtu,
- **FIN** – odesílatel ukončil odesílání,
- **kontrolní součet** – počítá se z TCP segmentu a několika informací z IP hlavičky, kontrolní součet vyžaduje sudý počet bajtů, proto se při lichém počtu na konec doplňuje bajt.

(1)

2.2.1 Zapouzdření TCP segmentu

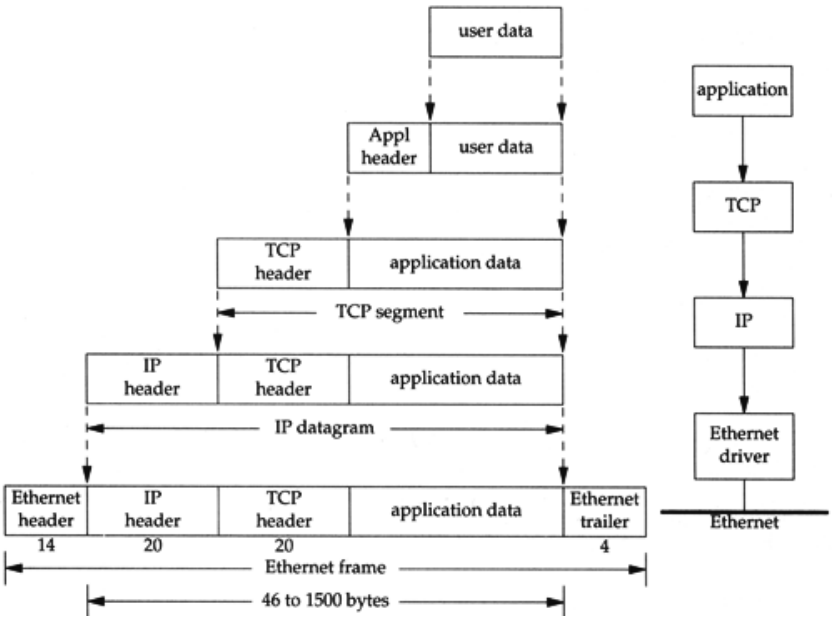
Zapouzdření je znázorněno na obrázku (Obrázek 3Obrázek 3), kde jsou na začátku aplikační data, která se mají odeslat přes TCP protokol. K nim si typicky připojí aplikace nějakou hlavičku, aby přijímací aplikace rozpoznala co s danými daty udělat.

Následně se aplikační data vkládají do TCP segmentu. Pokud je velikost dat vetší, jak 65535 bitů mínus délka TCP-záhlaví nebo je překročena domluvená velikost, tak je zapotřebí provést segmentaci, tedy rozdělit data do více TCP segmentů. TCP segmenty se poté vkládají do IP-datagramů. IP-datagram se vkládá do linkového rámce.

Při vkládání IP-datagramu do linkového rámce může nastat situace, že velikost IP-datagramu je větší než maximální velikostí přenášeného linkového rámce (MTU).

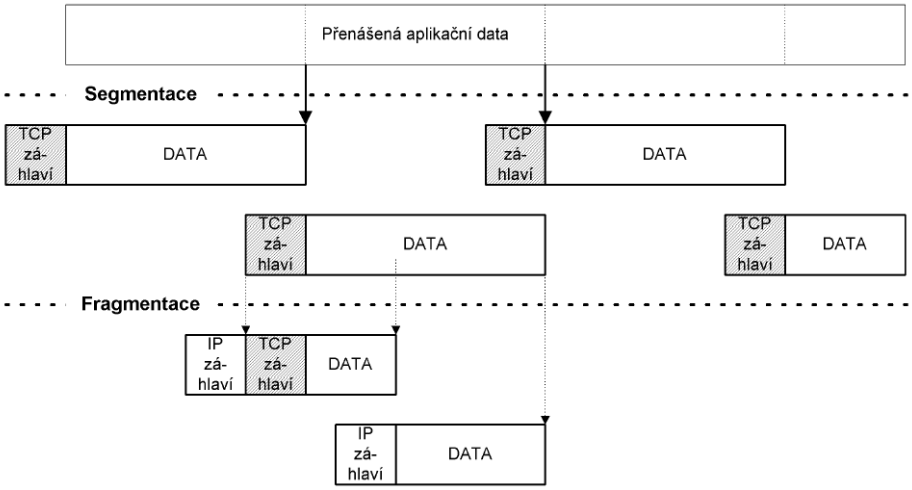
V takovém případě musí protokol IP provést fragmentaci, což znamená rozdělení na menší části. Fragmentace může způsobovat nadměrné přenosy, když se nějaký fragment ztratí a jelikož se protokol TCP stará o spolehlivé doručení a zároveň je protokolem vyšší vrstvy, musí si vyžádat přenos celého TCP segmentu, i když se ztratil jediný fragment. Proto je cílem se fragmentaci vyhnout vytvářením TCP segmentů o takové velikosti při které nebude nutná.

(2)



Obrázek 3 – Znázornění zapouzdření TCP segmentu

Zdroj: http://flylib.com/books/en/3.223.1.18/1/

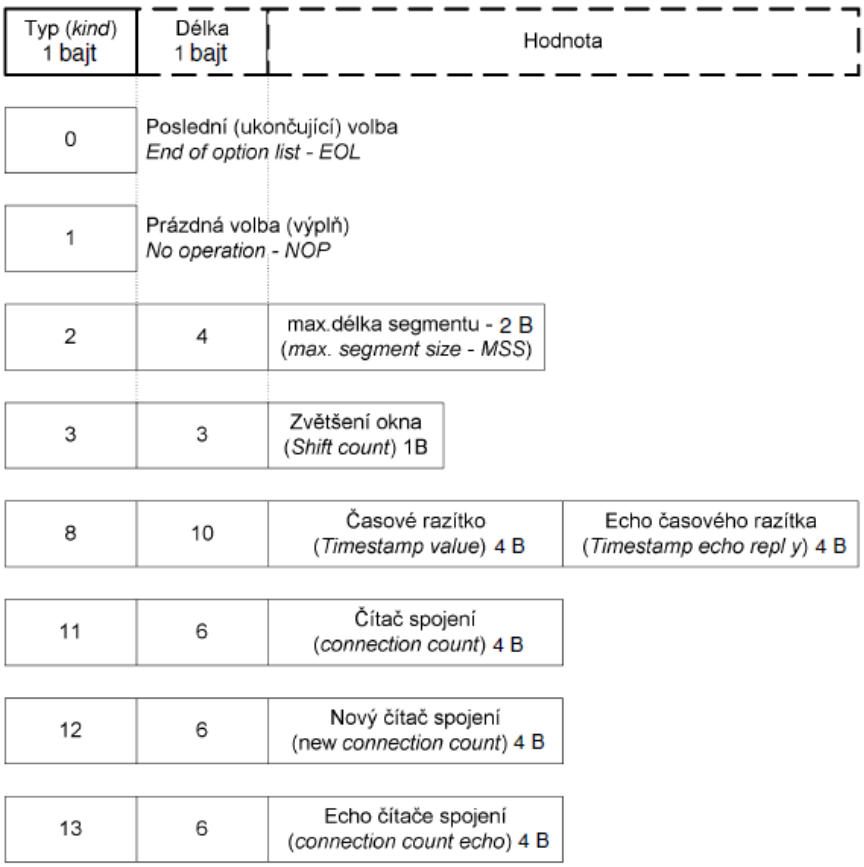


Obrázek 4 – Segmentace a fragmentace

Zdroj: (1)

2.2.2 Volitelné položky TCP záhlaví

Jak je vidět na obrázku (Obrázek 2) TCP segment se skládá z povinných položek záhlaví (tvoří pevnou délku 20 B) a volitelných položek (viz Obrázek 5). Volitelná položka je tvořena z typu, délky volitelné položky a hodnoty. Délka celého TCP záhlaví musí být dělitelná čtyřmi, pokud tomu tak není, záhlaví se doplňuje volitelnou položkou NOP. (1)



Obrázek 5 – Volitelné položky TCP záhlaví

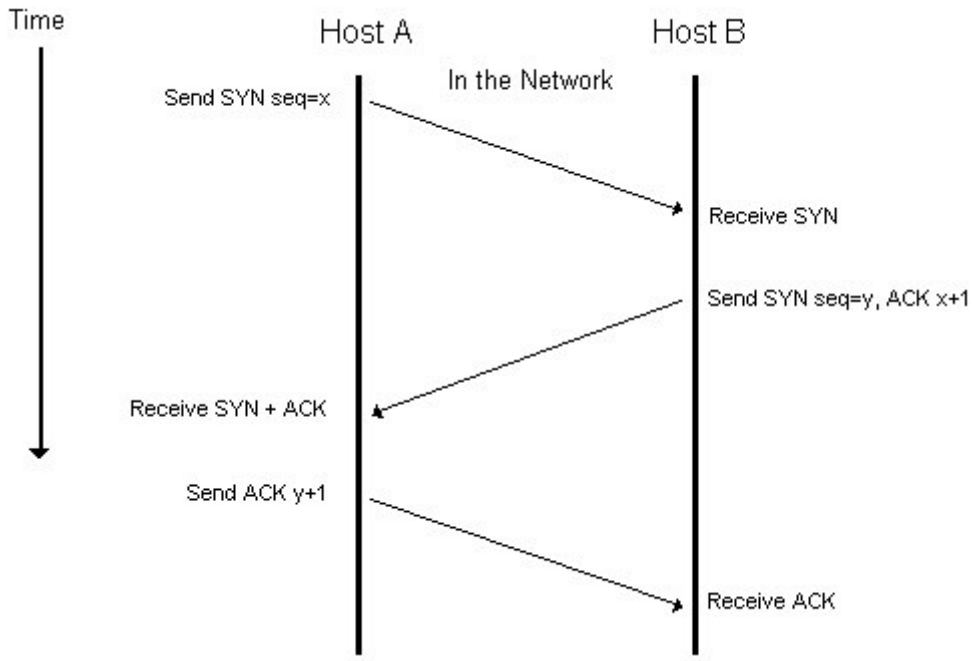
Zdroj: (1)

Položku maximální délka segmentu, lze použít pouze při navazování spojení a slouží k nastavení maximální velikosti segmentů během spojení, aby se vyhnulo IP fragmentaci.

2.3 Navazování spojení

TCP je spojově orientovaný protokol, tedy dvě strany navazují spojení. Jedna strana spojení očekává, nazýváme ji server, naopak druhá strana spojení navazuje, kterou nazýváme klient. Protokol TCP umožnuje i spojení obou stran současně, ale v praxi se tato možnost moc nepoužívá. (1)

Navázání spojený probíhá prostřednictvím výměny zpráv známý jako „*three-way handshake*“.



Obrázek 6 – Navazování spojení

Zdroj: (3)

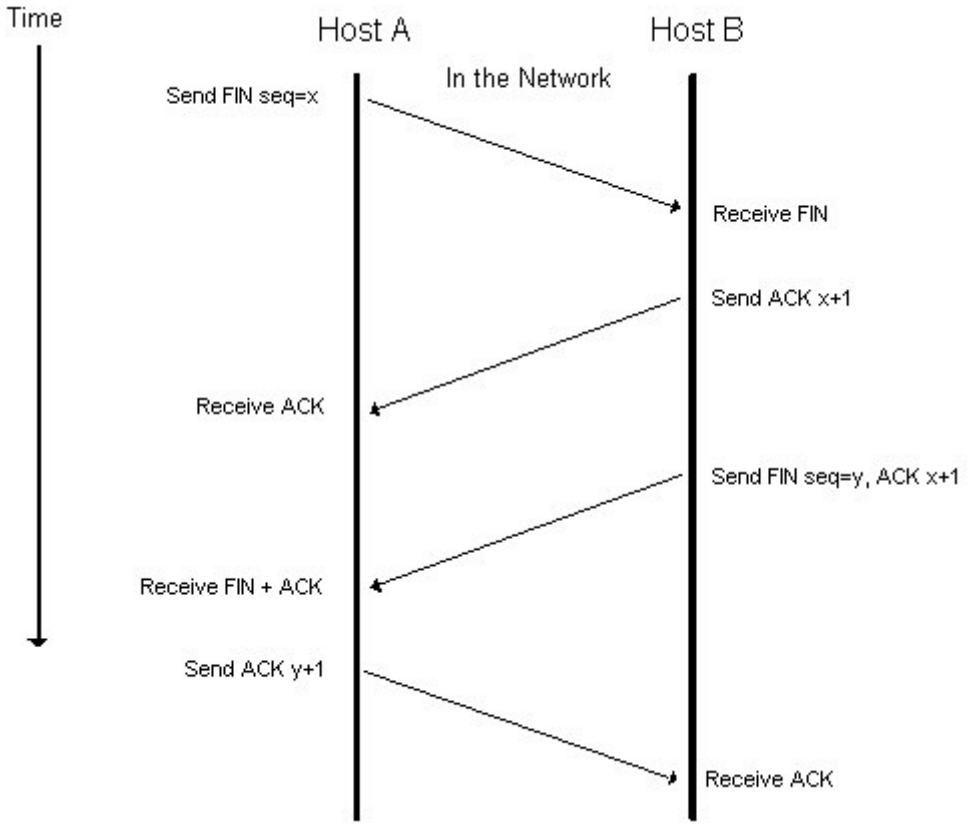
Průběh navázání spojení je zobrazen na obrázku (Obrázek 6). Řekněme, že Host A je klient a chce navázat spojení se serverem Hostem B. Na začátku klient odešle TCP segment s nastaveným řídícím bitem *SYN* a náhodně vygenerovaným startovacím pořadovým číslem odesílaného bajtu (tzv. ISN) v intervalu 0 až 2^{32}-1. Toto číslo je na obrázku (Obrázek 6) značeno jako proměnná x. Pořadové číslo nemůže být během spojení znovu vygenerováno, pouze v TCP segmentech s řídícím bitem *SYN*. (1) (3)

Po nějaké době přijme TCP segment Host B a zpracuje ho. Následně vytvoří vlastní TCP segment s řídícím bitem *SYN* a vygeneruje si vlastní pořadové číslo odesílaného bajtu, na obrázku (Obrázek 6 – Navazování spojení) značeno jako proměnná y. Jelikož Host B přijal nějaké data, musí je potvrdit nastavením řídícího bitu ACK a hodnotu pořadí dalšího očekávaného bajtu, tedy *x+1*. Od tohoto segmentu bude v každém nastaven řídící bit ACK. Takto sestavený TCP segment odešle. (3)

Host A obdržel TCP segment s řídícím bitem *SYN* a *ACK* od Hosta B. Nyní tento segment, ještě potvrdí segmentem s řídícím bitem *ACK* a hodnotou *y+1* a spojení je úspěšně navázáno. (3)

2.4 Ukončování spojení

Spojení navazuje zpravidla klient, ale ukončení spojení může iniciovat libovolná strana a to která první odešle TCP segment s řídícím bitem *FIN*. K ukončení spojení je potřeba výměny čtyř segmentů.



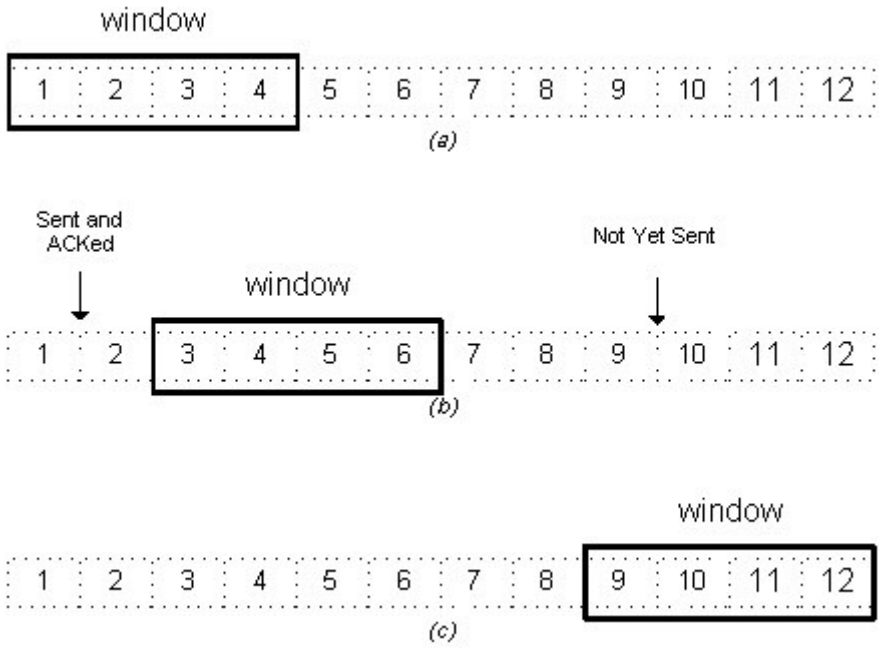
Obrázek 7 – Ukončení spojení

Zdroj: (3)

Jak je zobrazeno na obrázku (Obrázek 7), Host A ukončuje spojení odesláním segmentu s řídícím bitem *FIN*. Tímto Host A provádí aktivní ukončení, tzn. nemůže nadále odesílat aplikační data (segmenty s řídícím bitem *PSH*). Host B po přijmutí daného segmentu vstupuje do pasivního ukončení a potvrzuje ho segmentem s řídícím bitem *ACK*. Host B oproti Hostovy A může nadále odesílat data, dokud sám neukončí spojení. Tento stav se nazývá „*polo uzavřené spojení*“. Host B ukončí spojení obdobně jako Host A, tedy odešle segment s řídícím bitem FIN. Host A ho přijme a nakonec ho ještě potvrdí segmentem s řídícím bitem ACK. (1)

2.5 Technika okna

Používá se při přenosu velkého množství dat. Klient může odesílat data druhé straně, aniž by přenos musel být potvrzen až do velikosti okna. (3)



Obrázek 8 – Technika okna

Zdroj: (3)

Vysvětlení příkladu na obrázku (Obrázek 1). Při navázání spojení se klient a server dohodli, na velikosti okna 4 B. Klient chce, ale odeslat 12 B dat. Klient může odeslat maximálně 4 B dat a čeká na potvrzení (Obrázek 8 – a). Klient obdrží potvrzení (*ACK*) pro bajt 1 a 2, tudíž může odeslat bajt 5 a 6 a čekat opět na potvrzení (Obrázek 8 – b). Podobným principem dokud neodešle všech 12 B.

3 Qt

Qt je multiplatformní framework pro tvorbu aplikací s grafickým uživatelským rozhraním, ale také konzolových např. pro servery. Qt se snaží držet přístupu „*napsat jednou, zkompilovat kdekoliv*“. Qt má velice dobře zpracovanou dokumentaci. (4)

„Framework je softwarová struktura, která slouží jako podpora při programování. Cílem frameworku je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání.“ (5)

Qt je framework programovacího jazyka C++, ale podporuje spoustu jiných jazyků jako např. Python, Perl, Java, C#. (6)

3.1 Historie

Qt bylo vytvořeno v roce 1999 společností Trolltech, která Qt v roce 2008 prodala firmě Nokia. V březnu roku 2011 Nokia ohlásila prodej práv na provoz podpůrných služeb a prodej licencí pro komerční projekty vytvořené pomoci Qt společnosti Digia, avšak společnost Nokia ujistila, že zůstane hlavním vývojářem. (6)

3.2 Licence

Qt je licencováno pod dvěma licencemi:

- **Qt GNU LGPL[1] licence verze 2.1** – vhodný pro vývoj aplikací open source
- **Qt komerční licence** – pro vývojáře komerčních aplikací

(7)

3.3 Podporované Platformy

- Windows – Microsoft Windows XP, Vista a 7,
- Linux/X11 – X Window System 32bit a 64bit (Linux, HP-UX, Solaris, AIX, ...),
- Mac – Apple Mac OS X 10.6 "Snow Leopard" a 10.5 "Leopard" x86_64,
- Vestavěné linux platformy – Embedded Linux QWS (ARM) (PDA, Smartphone)
- Windows CE 5.0 – Qt pro Windows CE (ARMv4i, x86, MIPS),
- Symbian - QT pro Symbian/S60 5.0 platformu,
- Maemo – Maemo 5 (Linux, ARM, X11), plná podpora není zaručena.

(6)

[1] http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

3.4 Signály a Sloty

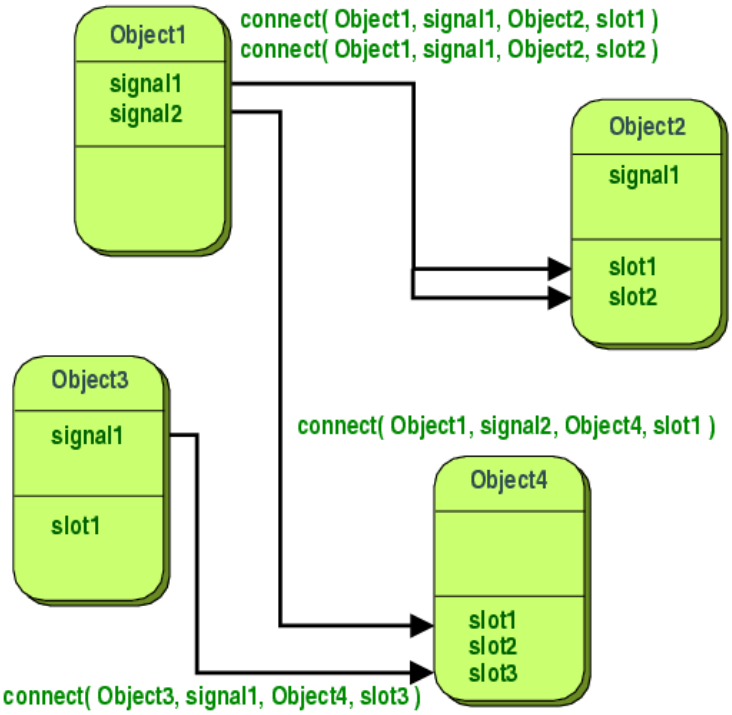
Signály a sloty se používají pro komunikaci mezi objekty. Jsou ústředním mechanismem Qt frameworku a pravděpodobně část, která se liší od jiných frameworků.

V programování GUI aplikací, když změníme jeden objekt, tak často chceme, aby tato změna byla jinému objektu oznámena. Ale hlavně chceme, aby jakýkoliv objekt byl schopný komunikovat s kterýmkoliv jiným. A právě toto nám umožnují signály a sloty.

Signál je vyvolán, když nastane patřičná událost. Qt framework má tyto signály definované u každé třídy či grafické komponenty pokud má co oznámit. Signály v těchto třídách oznamují nějakou vnitřní změnu. Například třída použitá v aplikaci klient *QTcpSocket* má definovaný signál *connected()*, který je vyvolán při úspěšném navázání TCP spojení.

Slot je klasická metoda, tedy lze normálně volat jako každá jiná metoda. Slot se liší od metody v možnosti napojit na slot signál, tedy je vyvolán signál a provede se na něj napojený slot.

(8)



Obrázek 9 – Signály a Sloty

Zdroj: (8)

Napojení signálů a slotů se provádí pomocí funkce:

```
connect(objekt1,SIGNAL(signal()),objekt1,SLOT(slot()));
```

3.5 Qt Creator

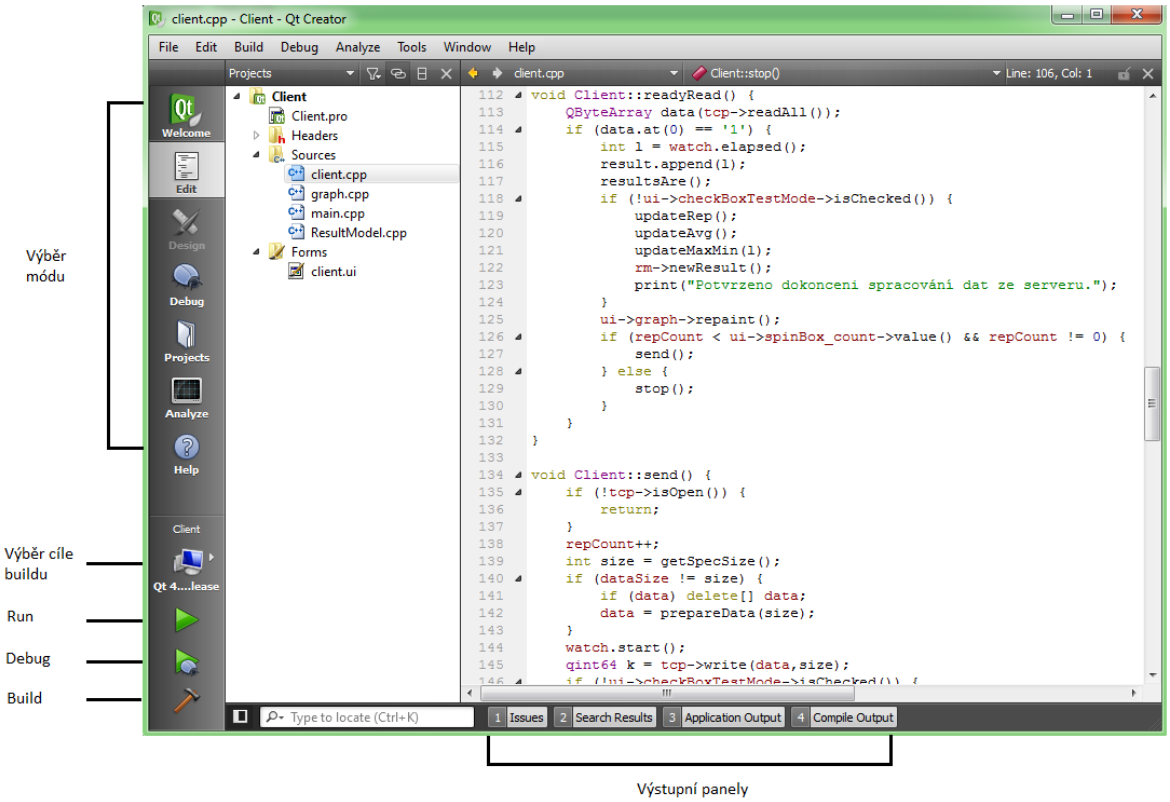
Qt Creator je multiplatformní vývojové prostředí (IDE), jak již název napovídá je určený pro vývojáře pracující s Qt frameworkem. Qt Creator je vyvíjen stejně jako Qt Framework společností Nokia.

Qt Creator se zaměřuje na poskytování funkcí, které pomáhají novým uživatelům Qt rychleji se zdokonalovat, ale také zvýšit produktivitu zkušených Qt vývojářů. (9)

Mezi vlastnosti Qt Creatoru patří:

- Editor kódu v C + +, podpora QML a ECMAScript
- Nástroje pro rychlou navigaci v kódu
- Zvýrazňování syntaxe a doplňování kódu
- Podpora refaktorování zdrojového kódu – např. přejmenování proměnné v celém projektu
- Kontextová nápověda
- Zobrazení vztahu závorek

(9)



Obrázek 10 – Ukázka Qt Creatoru

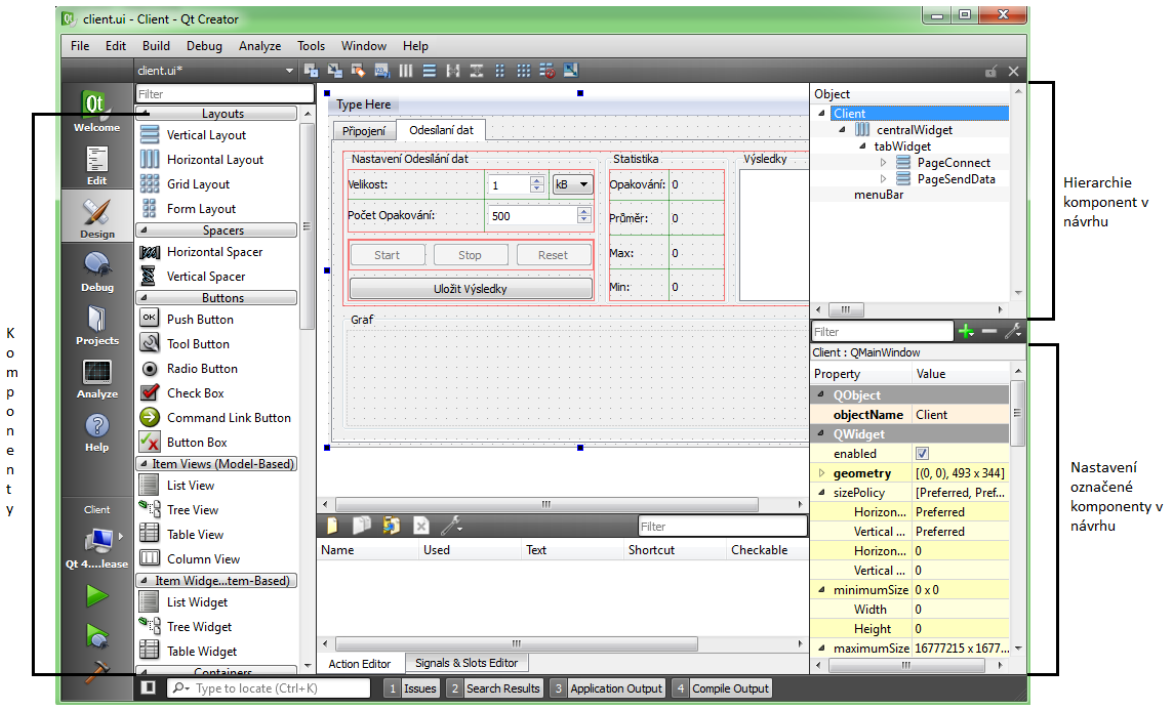
Zdroj: Autor

3.6 Qt Designer

Qt Designer je nástroj pro návrh grafického uživatelského rozhraní Qt aplikací. Grafické rozhraní aplikací mohou být napsány ručně pomocí zdrojového kódu nebo pomocí Qt Designeru graficky navrženy. Qt Designer je součástí Qt Creatoru.

Qt Designer je užitečný pro nováčky, ale také pro pokročilé jako pomocník pro urychlení vývoje. Qt Designer používá tzv. „*component-based*“ architekturu, což v praxi znamená, že jsou k dispozici předpřipravené grafické komponenty (viz Obrázek 11), které lze jednoduše přetáhnout a umístit na daný grafický návrh. Je k dispozici i možnost tvorby vlastních komponent. (4)

Tvorba vlastních komponent lze docílit např. děděním nějaké základní připravené komponenty. Tuto základní komponentu vložíme do návrhu Qt Designeru a následným pravým kliknutím se objeví možnost povýšit danou komponentu na naši vlastní rozšířenou komponentu v daném projektu. Tento způsob byl použit při tvorbě grafu v aplikaci klient (viz 4.1).



Obrázek 11 – Ukázka Qt Designer

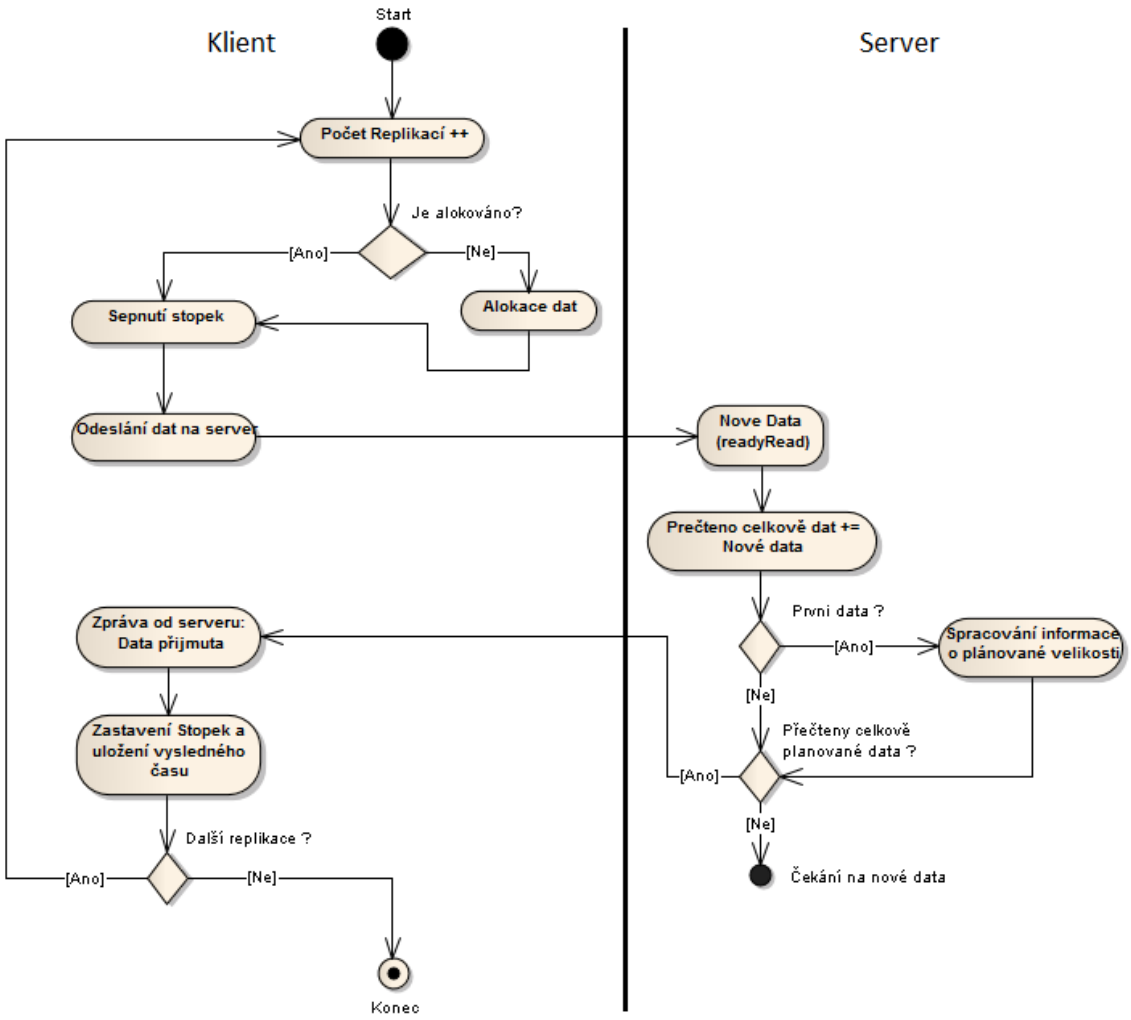
Zdroj: Autor

Celý návrh v Qt Designeru je reprezentován souborem *.ui, který lze spolu s projektem převést do jazyka C++ a následně zkompilovat ve výslednou aplikaci s grafickým rozhraním. (4)

4 Aplikace

Jak už název práce napovídá, jedná se o test dvou programů, tedy klient a server. Aplikace jsou napsány v jazyce C++ s použitím Qt frameworku. Jako vývojové prostředí je použit Qt Creator. Aplikace klient má grafické uživatelské rozhraní navržené v Qt Designeru, naopak aplikace server je pouze konzolová.

Qt je použito z důvodu snadné portace na jiné platformy a tím zajištění identických aplikací. Zmíněný důvod není jediný, Qt nabízí velice propracovaný modul pro síťové programování tzv. *QtNetwork module*. Ten obsahuje třídy *QTcpSocket* a *QTcpServer* pro snadnou implementaci TCP klienta a serveru.



Obrázek 12 – Průběh testování komunikace

Zdroj: Autor

Na obrázku (Obrázek 12 – Průběh testování komunikace) je zobrazen průběh procesu klienta a serveru při testování komunikace. Ještě před zahájením testu je nutno úspěšně navázat TCP spojení. Po navázání spojení si v klientovy navolíme potřebnou hodnotu přenášených dat a počet opakování nebo také jinými slovy počet replikací.

Po spuštění procesu se nejprve zkontroluje, jestli je alokována potřebná velikost dat k odeslání. Pokud ne, dynamicky se alokuje pole znaků (char) o požadované velikosti. V jazyce C++ jeden znak zabírá 1 B paměti.

```
char *temp = new char[1024];  //alokace 1kB dat
```

Poté se obrazně řečeno sepnou stopky a začíná se měřit testovaný čas přenosu. Hned na to se začnou data odesílat serveru. Data se po doručení ukládají na serveru do interního bufferu. Jakmile v bufferu jsou nějaké data, je vyvolán signál *readyRead()* a slot na něj napojený začne buffer zpracovávat. Při prvním zavolání *readyRead()* je z dat vytažena informace o plánované velikosti dat. Celková data odesílaná z klienta mohou být větší než interní buffer serveru, tudíž je buffer postupně naplňován přicházejícím daty z klienta a vyprazdňován neboli zpracován voláním signálu readyRead(). Server tedy musí mít nějakou proměnnou, do které při zpracování bufferu přičítá velikost přečtených dat a porovnává ji s vytaženou informací o plánované velikosti. Pokud se rovnají, požadované data jsou kompletně přenesena a zmíněná proměnná se vynuluje.

```
void ConnectionThread::readyRead() {
    QByteArray data = socket->readAll(); //čtení dat z bufferu
    readed += data.size(); //navýšení proměnné readed o velikost
                           přečtených dat
    count++;              //kolikáte čtení bufferu
    if (readed == size) { // proměnná „size“ je plánovaná velikost
        socket->write("1"); //odeslání klientovy „data přijmuta“
    }
}
```

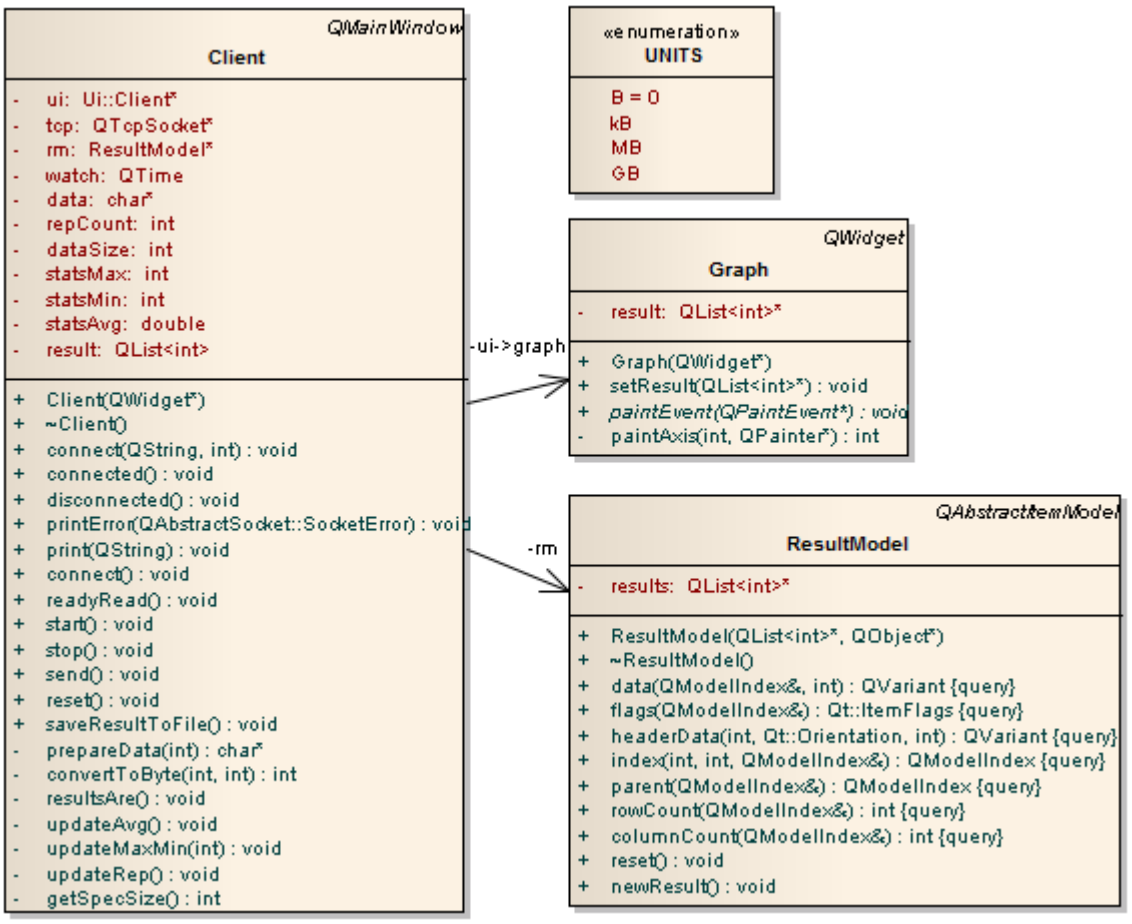
Po přenesení dat sám server odešle informaci klientovy, že požadované data přečetl. Klient po obdržení této informace zastaví stopky a uloží si výsledný čas. Následně zkontroluje, jestli má testování opakovat.

4.1 Klient

Klient obsahuje třídy zobrazené na obrázku (Obrázek 13).

Třída *Client* je hlavní okno aplikace. Obsahuje ukazatel na instanci třídy *QTcpSocket*, která slouží pro TCP komunikaci. Nabízí standartní signály oznamující stav. Signál *connected()* oznamuje úspěšné navázání komunikace a signál *disconnected()* zase ukončení spojení. Nejdůležitější signál je však *readyRead()*, který oznamuje existenci nových dat v interním bufferu. Tento signál je napojen na stejně nazvaný slot *readyRead()*, který data z bufferu přečte. V aplikaci klient se pouze očekávají data ze serveru, které signalizující dokončení a potvrzení zpracování odesílaných dat, tudíž se data přečtou a zkontroluje se první bajt dat, jestli obsahuje potvrzující znak, pokud ano jsou zastaveny stopky a zpracován čas. Třída

Client uchovává datovou strukturu *QList<int>* (results), do které se vkládají výsledné časy testování.



Obrázek 13 – Class diagram aplikace Klient

Zdroj: Autor

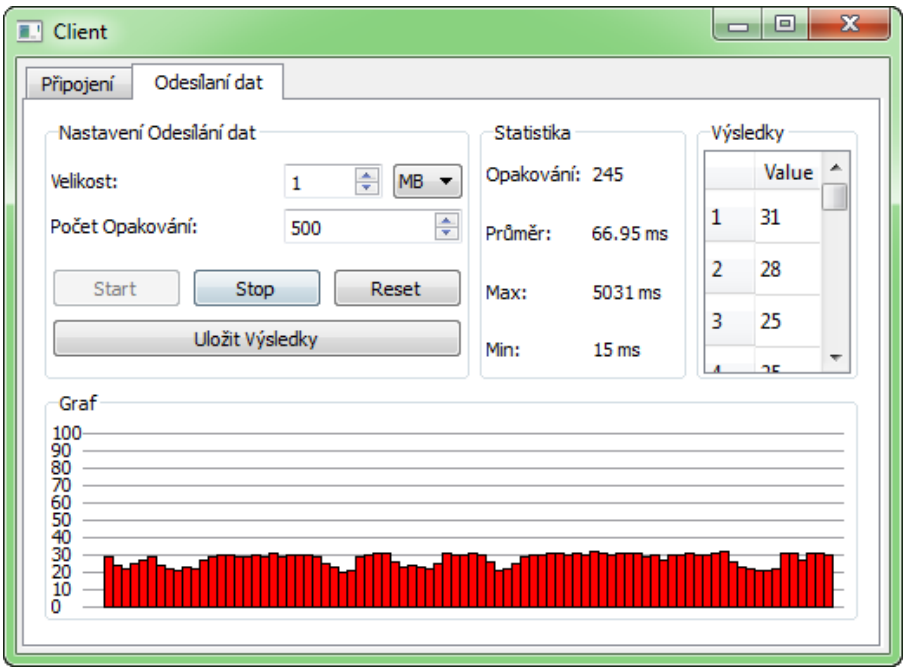
Další třída v aplikaci klient je *ResultModel*. Slouží jako datový model pro komponentu *QTableView*. Jak už podle názvu vyplývá, jedná se o tabulku. Tato komponenta slouží k zobrazení výsledných časů. Třída *ResultModel* dědí abstraktní třídu *QAbstractItemModel*, tato třída slouží jako základní rozhraní pro položkové modely. Třída *ResultModel* musí implementovat všechny abstraktní metody třídy *QAbstractItemModel*. Jedna z nejdůležitějších metod je metoda *data*, jejíž první parametr je typu *QModelIndex*, což je třída určující souřadnice v objektu, v našem případě políčka tabulky. Tato metoda je volána pro každou položku a přes právě zmíněný parametr rozeznáme pozici, tudíž můžeme pro danou pozici vrátit potřebná data, která se zobrazí.

Další parametr je role zobrazení dat. Výstupní parametr je *QVariant*, je to speciální třída, která umí uchovat mnoho druhů datových typů. Třída *ResultModel* uchovává ukazatel na datovou strukturu *QList<int>* s výslednými časy ze třídy *Client*. Jak je vidět v ukázce metody níže. Třída *ResultModel* je vytvořena z důvodu ukázky použití tvoření vlastních modelů a úspory paměti, protože kdyby byla použita jiná varianta např. komponenta

QTableWidget, kde je model implementován s vlastní datovou strukturou pro prvky, což znamená zbytečné uložení totožných dat na dvou místech.

```
QVariant ResultModel::data(const QModelIndex &index, int role) const {
    if (!index.isValid())              //index je neplatný
        return QVariant();
    if (role != Qt::DisplayRole)       //pokud není zobrazovací role
        return QVariant();
    if (index.row() >= results->size()) { //ošetření konce výsledků
        return QVariant();
    }
    if (index.column() == 0) {         //pouze první sloupec
        return QVariant(results->at(index.row()));
            //podle čísla řádku tabulky vracíme hodnotu
    }
    return QVariant();
}
```

Poslední je třída *Graph*, která dědí třídu *QWidget*. *QWidget* je základní třída pro grafické prvky uživatelského rozhraní. Řeší přijímání událostí z klávesnice, myši a další události v okenním systému. Avšak hlavní úlohou je reprezentování sama sebe na obrazovce. K tomu slouží metoda *paintEvent()*, která je právě reimplementována ve třídě *Graph*. Tato metoda je volána vždy, když je potřeba vykreslit nebo překreslit okno aplikace. Třída *Graph* uchovává ukazatel na datovou strukturu *QList<int>* s výslednými časy z třídy *Client*. Pokaždé když je volána metoda *paintEvent()*, tak je datová struktura s výslednými časy zpracována a vykreslena jako např. graf na obrázku (Obrázek 14). Aktualizace grafu při novém získaném času je zajištěna překreslením voláním metody *repaint()* ve třídě *Graph*.



Obrázek 14 – GUI aplikace Klient

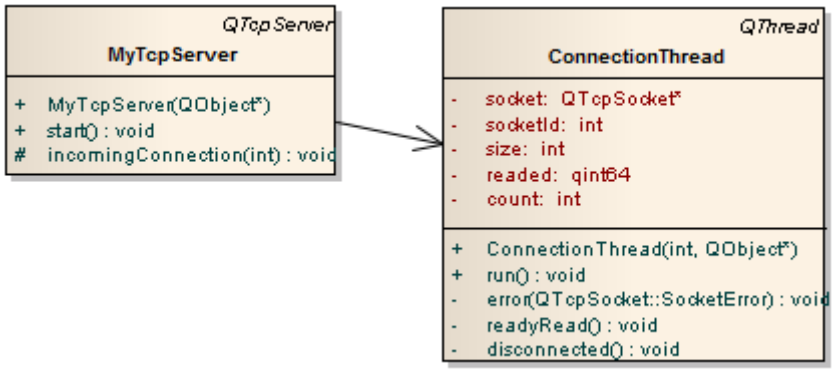
Zdroj: Autor

4.2 Server

Server obsahuje třídy zobrazené na obrázku (Obrázek 15).

Třída *MyTcpServer* dědí třídu *QTcpServer*. Tato třída zajišťuje naslouchání na určitém portu pro přijímání nových TCP připojení. Naskytuje se tu otázka, proč je tato třída děděna do nové třídy. Je to z důvodu reimplementace metody *incomingConnnection(int)*. Tato metoda je zavolána v případě nového připojení. Má vstupní parametr tzv. *socketDescriptor*, je to číselná identifikace připojení. Klasicky tato metoda vytvoří novou instanci třídy *QTcpSocket*, do které nastaví daný *socketDescriptor* a emituje signál newConnection(). V aplikaci server je však tato metoda reimplementována z důvodu vytvoření příchozího spojení v novém vlákně, což je třída *ConnectionThread*.

Třída *ConnectionThread* dědí třídu *QThread*. *QThread* je základní třída pro vytvoření vlákna. Obsahuje abstraktní metodu *run()*, která musí být implementována v potomkovi. Po spuštění vlákna je zavolána právě metoda run() a po jejím dokončení se vlákno zastaví.



Obrázek 15 – Class diagram aplikace Server

Zdroj: Autor

Řešení metody *incomingConnection()*:

```
void MyTcpServer::incomingConnection(int handle) {
    ConnectionThread *thread = new ConnectionThread(handle, this);
    connect(thread, SIGNAL(finished()), thread, SLOT(deleteLater()));
    thread->start();
}
```

Nejprve se dynamicky vytvoří instance třídy *ConnectionThread* se vstupními parametry. První parametr je již zmíněný *socketDescriptor* a druhý je ukazatel na objekt, který vlákno vytvořil[2], v tomto případě instance třídy *MyTcpServer*. Dále je napojen signál *finished()* na slot *deleteLater()*, toto je velice zvláštní případ napojení signálu a slotu, protože je to v rámci totožného objektu. Signál *fnished()* je deklarovaný v *QThread* a je vyvolán, když je dané vlákno dokončeno. Slot *deleteLater()* označí daný objekt k dealokování. Toto

[2] Toto slouží k zajištění hierarchie rodičů a potomků objektů, např. když je smazán rodič, jsou automaticky smazáni jeho potomci.

napojení tedy zajistí dealokaci daného vlákna po dokončení. A nakonec je vlákno spuštěno a začne probíhat metoda *run()*.

```
void ConnectionThread::run() {
    socket = new QTcpSocket();
    if (!socket->setSocketDescriptor(socketId)) {
        emit error(socket->error());
        return;
    }
    connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()),
Qt::DirectConnection);
    connect(socket, SIGNAL(disconnected()), this, SLOT(disconnected()),
Qt::DirectConnection);
    exec();
}
```

V metodě *run()* se nejprve dynamicky alokuje *QTcpSocket()*. Poté se do něj nastaví *socketDescriptor*. Následně jsou napojeny signály *readyRead()* a *disconnected()* na stejně pojmenované sloty. Jako poslední velice důležitá část této metody je příkaz *exec()*, který spustí smyčku událostí a tím je zajištěno nedokončení metody, tzn. vlákno není ukončeno. Jelikož vlákno není ukončeno, může reagovat na příchozí data pomocí signálu *readyRead()* až do ukončení spojení. Při ukončení spojení je vyvolán signál *disconnected()*, který je napojen na stejně pojmenovaný slot. Ve slotu *disconnected()* je příkaz *exit(0)*, při jehož vyvolání se ukončí smyčka v metodě *run()*, metoda se dokončí a vlákno je ukončeno. Tím se vyvolá signál *finished()*, který je napojen na slot *deleteLater()*. Tento slot, jak bylo zmíněno, označí toto vlákno ke smazání.

Toto je ukázka jednoho z možných řešení spravování připojení na serveru. Každé připojení spravuje vlastní vlákno a tím je docíleno souběžné zpracovávání časově náročných operací vyvolaných v každém připojení. Nevýhoda tohoto řešení spočívá v paměťové režii vláken. Např. navázalo by spojení tolik klientů, až by došla operační paměť a aplikace server by se zhroutila.

5 Prostředí testování komunikace

Testování bylo prováděno na virtualizačním nástroji VirtualBox. Tento nástroj byl spuštěný na hostitelském počítači (viz 5.4).

Ve VirtualBoxu jsou vytvořeny čtyři virtuální stroje. Dva s Windows platformou, tedy jeden pro klientskou aplikaci a druhý pro serverovou aplikaci. Obdobně byly vytvořeny dva pro Linux platformu.

Důvodem použití virtualizace při testování, je zajištění co nejvíce identického prostředí pro obě platformy a vyloučení nepříznivých vlivů.

5.1 VirtualBox

VirtualBox je multiplatformní virtualizační aplikace, která lze nainstalovat na Intel nebo AMD založený počítač, ať už se systémem Windows, Mac, Linux nebo Solaris. Rozšiřuje možnosti počítače tak, že dokáže spustit více operačních systému ve stejnou dobu. (10)

VirtualBox vyvinula společnost Innotek GmbH, kterou zakoupila společnost Sun Microsystems, ta je nyní součástí Oracle Corporation. (11)

VirtualBox je nabízen ve dvou licencích:

- **VirtualBox Personal Use and Evaluation License (PUEL)** – pod touto licencí je nabízen předkompilovaný binární kód, neboli klasický instalační balíček pro daný OS. Tato licence je určena pro osobní použití nebo pro zkušební účely.
- **VirtualBox Open Source Edition (OSE)** – v podobě zdrojových kódů s licencí GNU General Public License (GNU/GPL), verze 2.

(11)

5.1.1 Virtuální síťování

VirtualBox nabízí osm virtuálních PCI Ethernet karet pro každý virtuální stroj. Pro každou kartu lze nastavit, jakým typem hardwaru bude virtualizována a módem, kterým bude reprezentována vzhledem k fyzickému hardwaru počítače. (10)

K výběru typu virtualizace hardwaru je na výběr z několika možností. Při testování byla na všech virtuálních strojích použita karta „*Intel PRO/1000 MT Desktop (82540EM)*“.

Jako síťový mód byla použita varianta vnitřního síťování. Tento typ omezuje síťovou komunikaci pouze mezi virtuálními stroji na stejném počítači. Vnitřní sítě se tvoří automaticky, když jsou potřeba a jsou identifikovány čistě podle názvu. Pokud je více než jedna virtuální síťová karta se stejným názvem vnitřní sítě, tak pomocný VirtualBox ovladač automaticky tyto karty spojí přes síťový switch. Tento síťový switch se chová jako běžný Ethernet switch s podporou broadcast/multicast rámců. (10)

5.2 Windows platforma

Operační systém pro Windows platformu je použit *Windows 7 Home Premium SP1 x64*.

5.3 Linux platforma

Operační systém pro Linux platformu je použit *Ubuntu 10.04 Lucid Lynx x64*.

5.4 Hostitelský počítač

Testy probíhaly na laptopu Asus G53J s operačním systémem *Windows 7 Home Premium SP1 x64* a hardwarovou specifikací:

- Procesor – Intel Core I7 740QM 1,73 GHz
- RAM – 6 GB DDR3

6 Testování komunikace

Testování je rozděleno na sedm částí podle velikosti přenášených dat (viz Tabulka 1). V každé části je otestována komunikace mezi všemi kombinacemi propojení Windows a Linux platformy. Komunikace je tedy testována mezi totožnými, ale také různými platformami. Je brán ohled, na jaké platformě je umístěn klient a server. Celkově je tedy sedm částí podle velikosti dat a v každé části porovnání čtyř kombinací.

Tabulka 1 – Použité hodnoty při testování

Velikost	Počet opakování (replikací)
1 kB	1000
10 kB	1000
100 kB	1000
1 MB	1000
10 MB	500
100 MB	500
500 MB	200

6.1 Zpracování výsledků

Výsledky testování komunikace jsou zpracovány v tabulkovém procesoru Excel od společnosti Microsoft. MS Excel je použit z důvodu podpory funkcí pro výpočet směrodatné odchylky, aritmetického průměru, maximálních a minimálních hodnot, ale hlavně díky nástroji pro snadnou tvorbu grafů.

Postup zpracování začne vygenerováním souboru CSV (viz 6.1.1) s výslednými hodnotami. Tento soubor se generuje v aplikaci klient pomocí tlačítka „*Uložit výsledky*“ (viz Obrázek 14), kde se následně v dialogovém okně vybere umístění souboru. MS Excel přímo nabízí import zmíněného souboru. Po importu dat jsou vypracovány výpočty kumulovaného průměru (viz 6.1.2), směrodatné odchylky (viz 6.1.3) a jsou určeny maximální a minimální hodnoty. Tyto výpočty jsou následně navzájem porovnány ve všech kombinacích propojení platforem vzhledem k velikosti odesílaných dat.

Při zpracování bylo použito značení:

„Typ Platformy Klient“_„Typ platformy Server“

Např. WIN_WIN, WIN_LINUX,…

6.1.1 Soubor CSV

Je souborový formát sloužící pro výměnu tabulkových dat. Skládá se z řádků, kde jsou data z jednotlivých sloupců odděleny středníkem.

Tento typ souboru generuje aplikací Klient. Obsahuje výsledné hodnoty testování, jelikož se jedná o jeden sloupec dat, je každá hodnota na novém řádku.

6.1.2 Kumulovaný průměr

Je to postupný průměr počítaný pro každé číslo v daném souboru čísel. Pro číslo x je kumulovaný průměr vypočten, jako vypočet průměru od prvního čísla až po číslo x včetně.

Kumulovaný průměr je použit jako datový podklad ve výsledných grafech, z důvodu přehlednějšího zobrazení výsledku. Kdyby byly v grafu použity reálné výsledné hodnoty, vznikl by nepřehledný, nic neříkající graf.

6.1.3 Směrodatná odchylka

Směrodatná odchylka je kvadratický průměr odchylek hodnot znaku od jejich aritmetického průměru. Vypovídá o tom, jak moc se od sebe liší typické případy v souboru zkoumaných čísel. Pokud je odchylka malá, jsou si prvky souboru většinou navzájem podobné, a naopak velká odchylka signalizuje velké vzájemné odlišnosti. (12)

Směrodatná odchylka se značí řeckým písmenem σ a platí:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} = \sqrt{(\frac{1}{N}\sum_{i=1}^{N}) - \bar{x}^2}$$

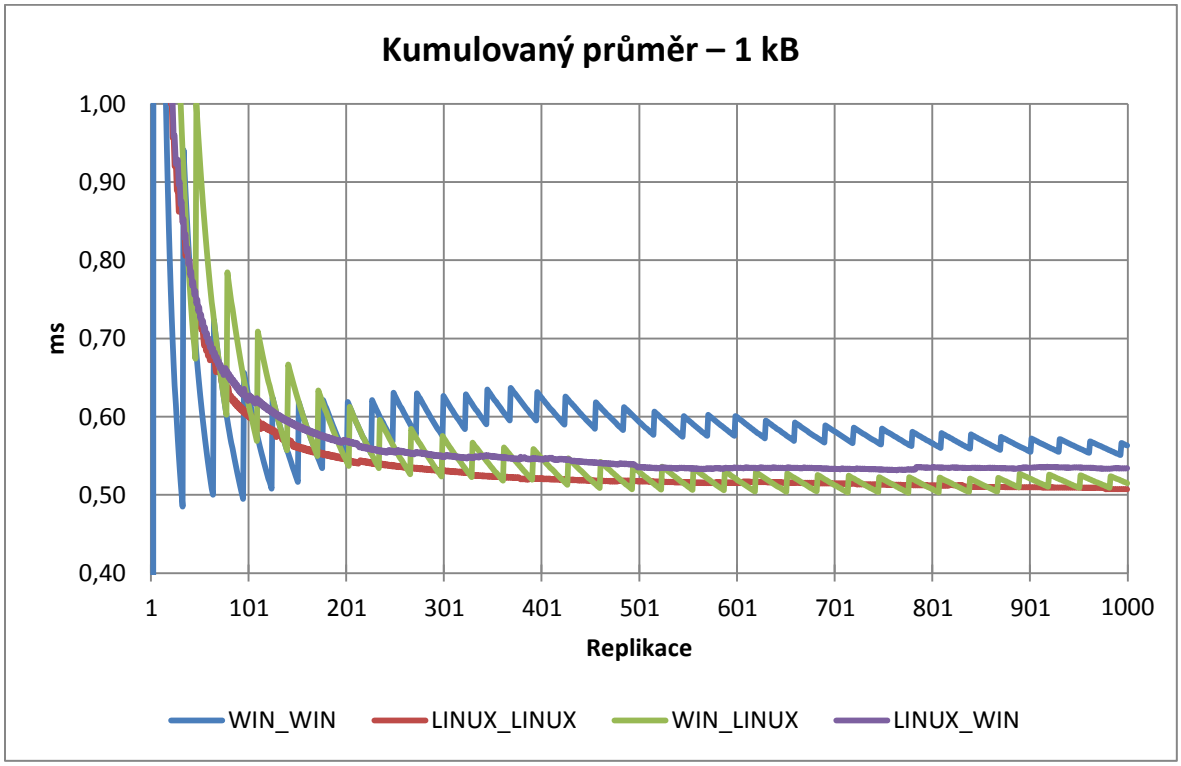
Směrodatná odchylka je použita ve výsledcích testování jako ukazatel rozdílů v naměřených hodnotách, jinými slovy ukazatel stability.

7 Výsledky testování

7.1 1 kB

Tabulka 2 – Výsledná Statistika – 1 kB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	0,56	2,92	0	16
LINUX_LINUX	0,51	0,54	0	4
WIN_LINUX	0,52	2,79	0	16
LINUX_WIN	0,53	0,53	0	5



Obrázek 16 – Graf kumulovaného průměru – 1kB

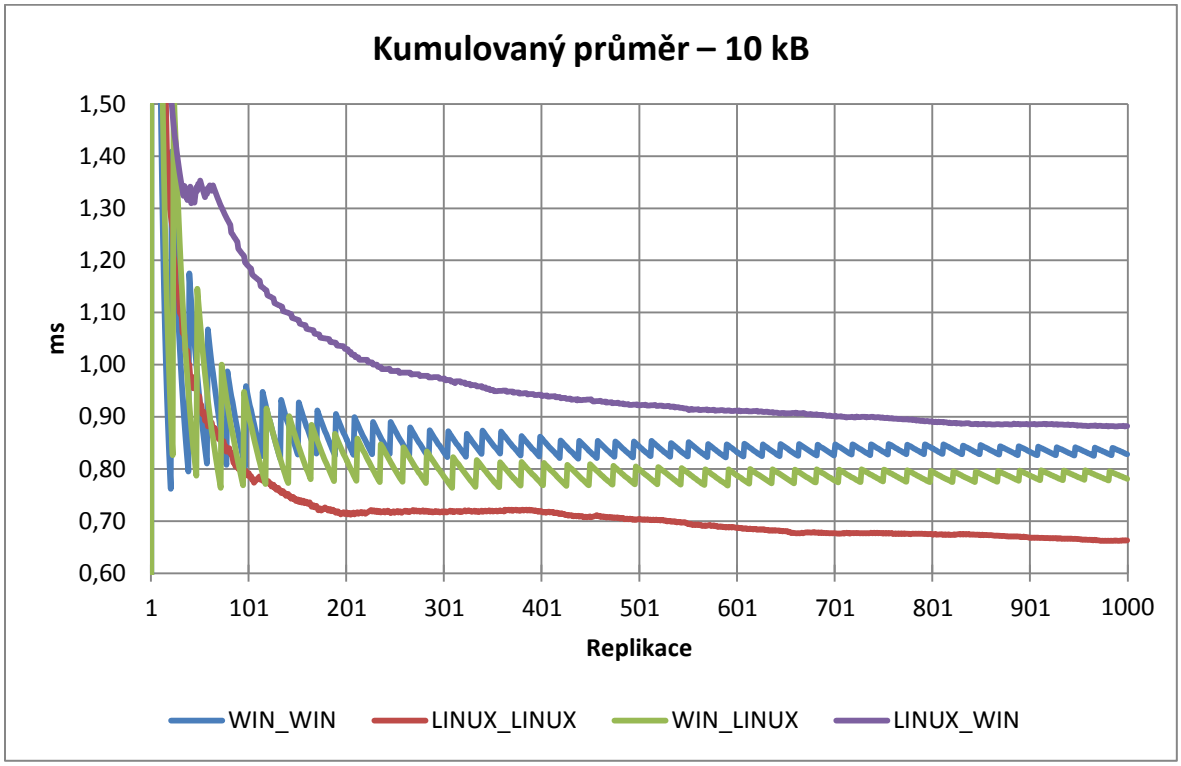
Zdroj: Autor

Rozdíly ve výsledných průměrných hodnotách jsou zanedbatelné. Jak je vidět z grafu a také podle směrodatné odchylky lze přiřadit závislost na umístění klienta. V měřeních, kde je klient na platformě Linux, jsou výsledky stabilnější, než na platformě Windows.

7.2 10 kB

Tabulka 3 – Výsledná statistika – 10 kB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	0,83	3,50	0	16
LINUX_LINUX	0,66	0,54	0	6
WIN_LINUX	0,78	3,69	0	19
LINUX_WIN	0,88	0,43	0	5



Obrázek 17 – Graf kumulovaného průměru – 10 kB

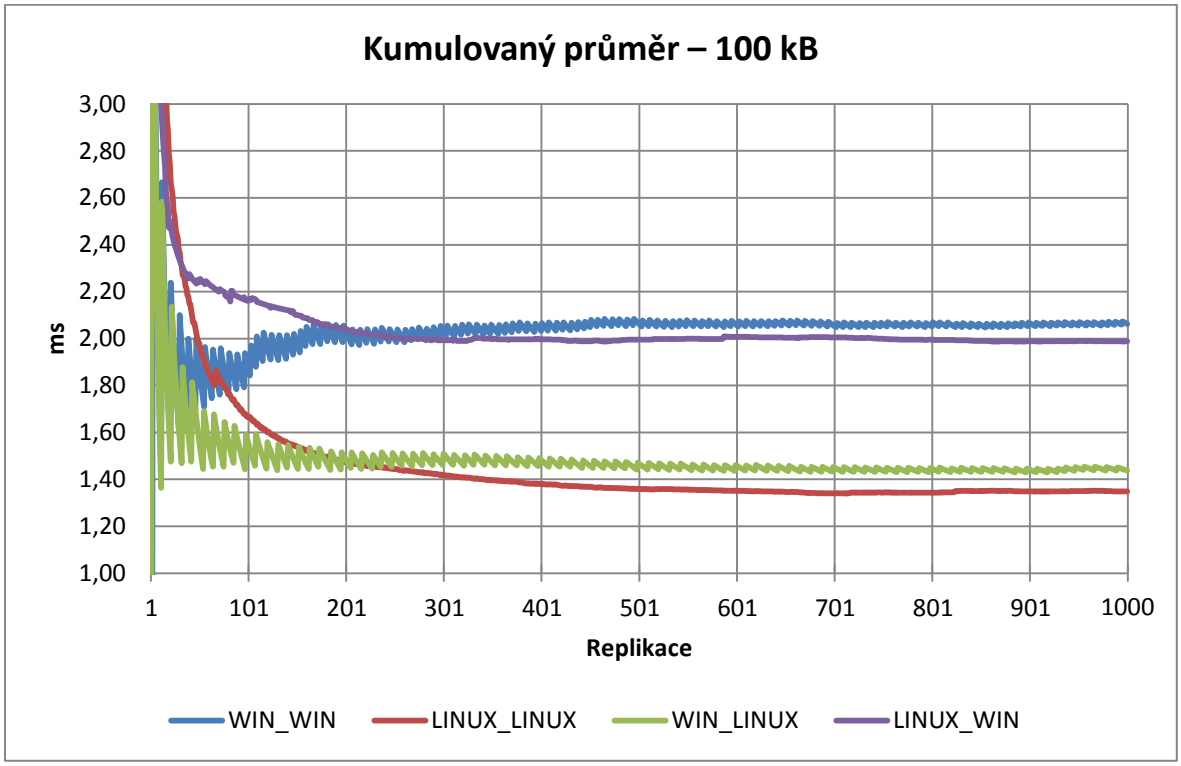
Zdroj: Autor

Podobné výsledky jako v předchozí části (7.1). Potvrdila se závislost umístění klienta na stabilitě. U měření klient na Windows platformě je patrný vyšší nárůst směrodatné odchylky.

7.3 100 kB

Tabulka 4 – Výsledná statistika – 100 kB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	2,06	5,29	0	16
LINUX_LINUX	1,35	0,64	1	11
WIN_LINUX	1,44	4,52	0	16
LINUX_WIN	1,99	0,51	1	9



Obrázek 18 – Graf kumulovaného průměru – 100 kB

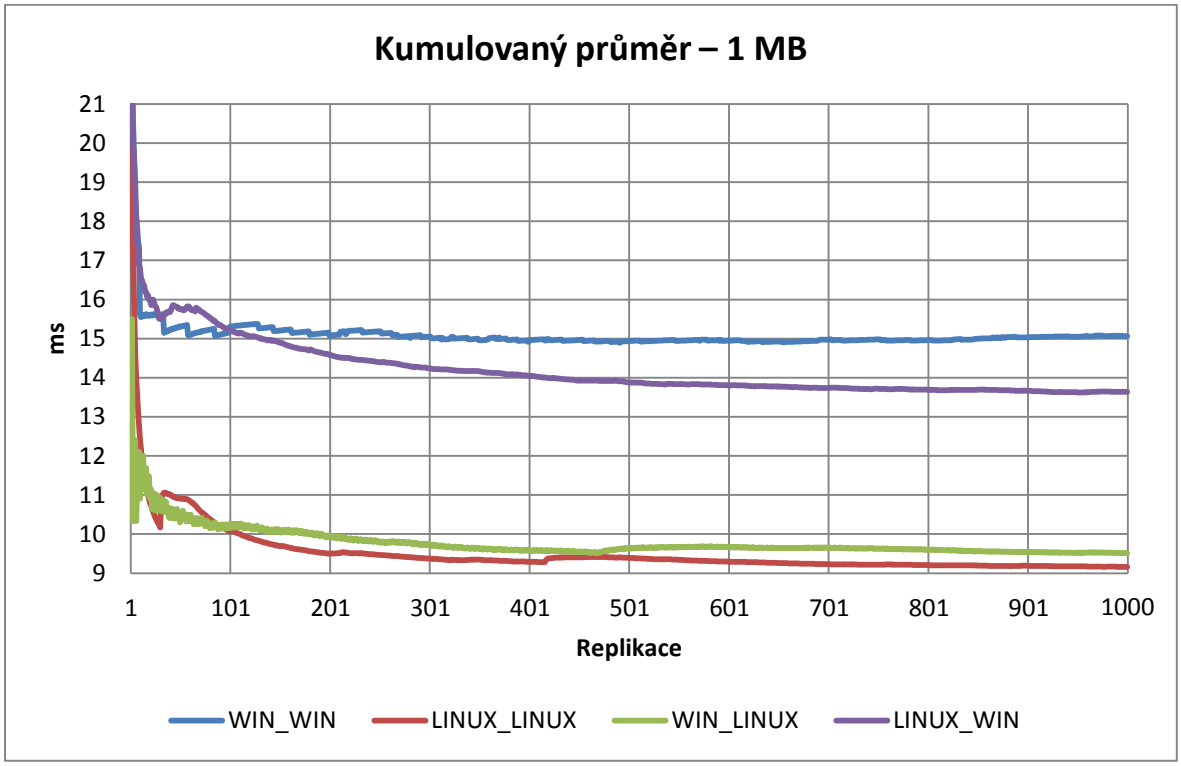
Zdroj: Autor

Zde začíná být viditelná závislost výsledného průměru na umístění serveru. Opět nárůst směrodatné odchylky u měření s klientem na Windows platformě.

7.4 1 MB

Tabulka 5 – Výsledná statistika – 1 MB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	15,06	4,53	0	32
LINUX_LINUX	9,16	1,47	8	32
WIN_LINUX	9,52	7,63	0	16
LINUX_WIN	13,64	1,51	11	28



Obrázek 19 – Graf kumulovaného průměru – 1 MB

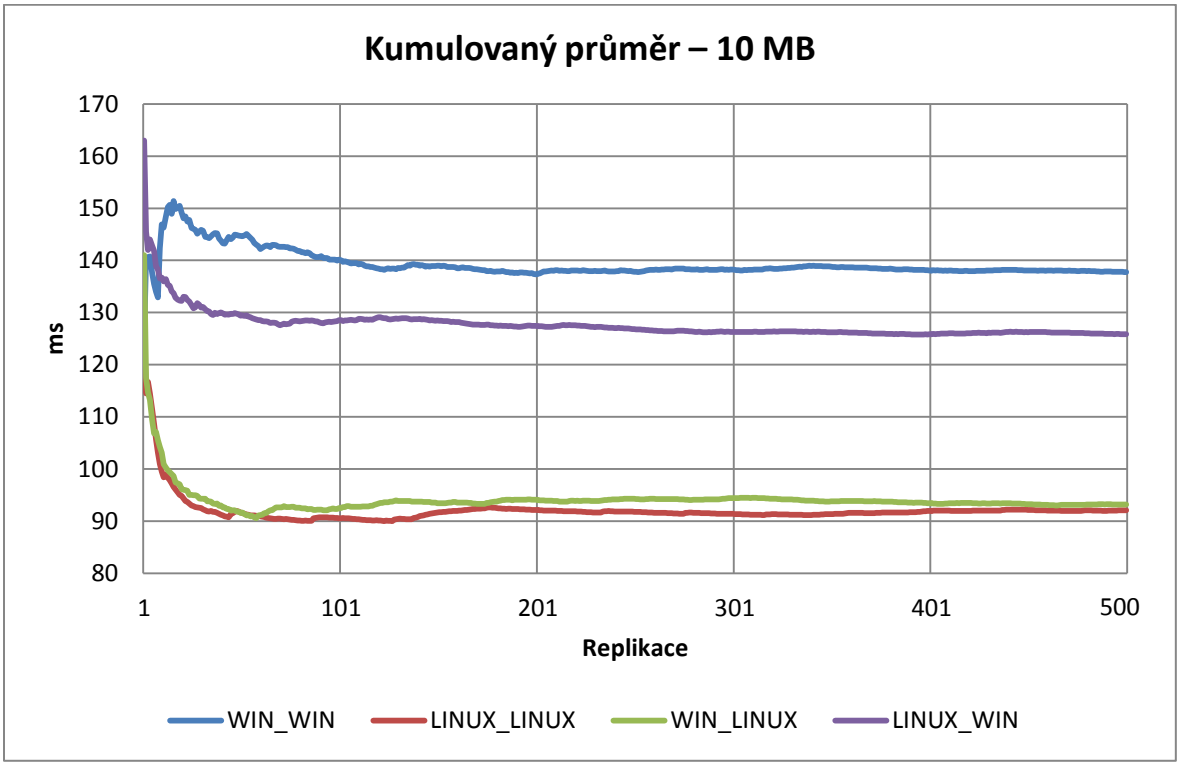
Zdroj: Autor

Potvrzena závislost průměru na umístění serveru. Směrodatné odchylky vyšší u klienta na Windows platformě.

7.5 10 MB

Tabulka 6 – Výsledná statistika – 10 MB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	137,75	15,76	109	218
LINUX_LINUX	92,08	8,23	84	143
WIN_LINUX	93,16	9,84	78	141
LINUX_WIN	125,83	9,40	107	163



Obrázek 20 – Graf kumulovaného průměru – 10 MB

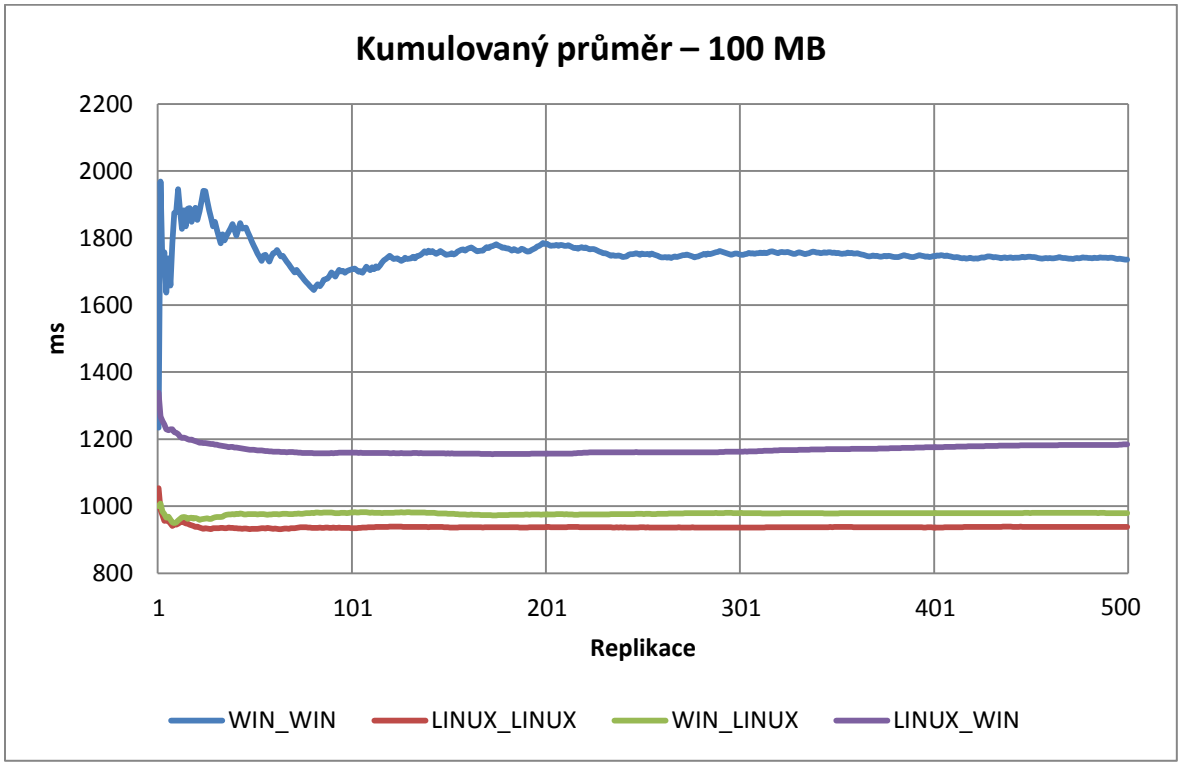
Zdroj: Autor

Směrodatné odchylky jsou vyrovnanější, kromě měření klient a server na Windows platformě, kde je vidět vyšší nárůst.

7.6 100 MB

Tabulka 7 – Výsledná statistika – 100 MB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	1735,75	569,62	1109	2719
LINUX_LINUX	938,16	38,99	869	1069
WIN_LINUX	979,22	44,01	875	1109
LINUX_WIN	1184,75	52,12	1096	1555



Obrázek 21 – Graf kumulovaného průměru – 100 MB

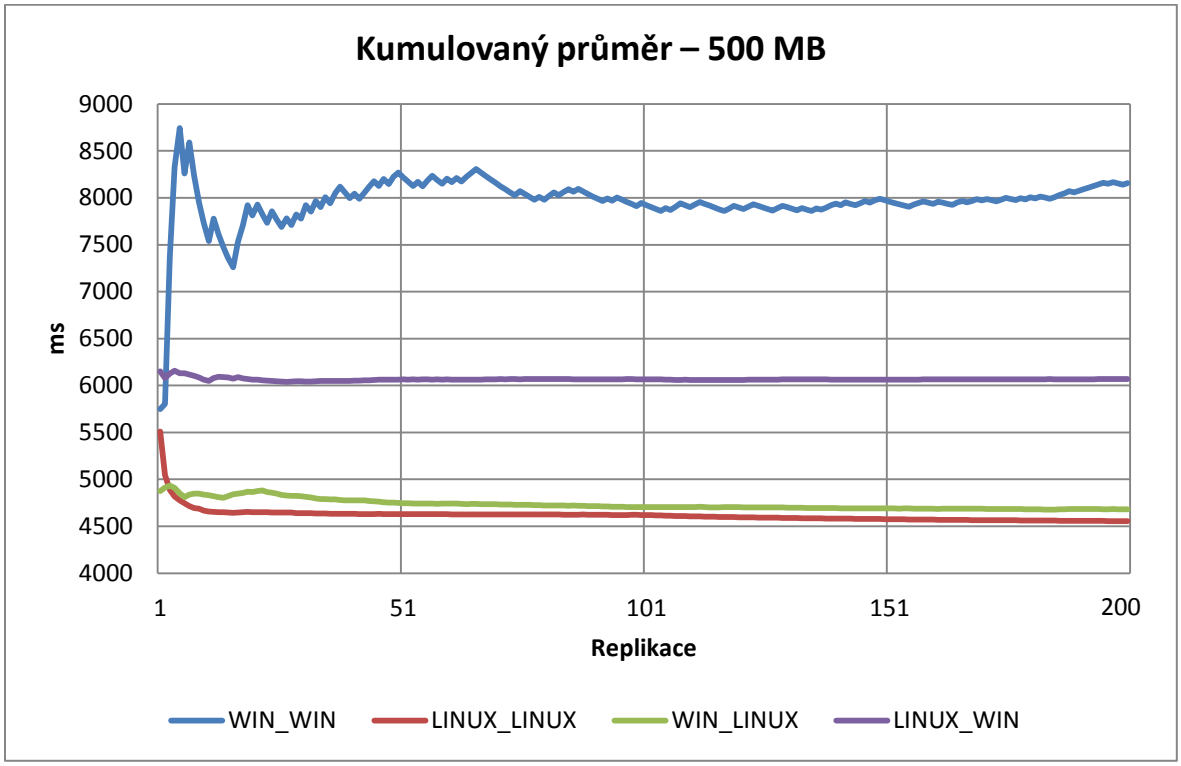
Zdroj: Autor

Výraznější horší výsledky lze pozorovat u měření klient a server na platformě Windows. Druhý nejhorší je server na Windows platformě, což potvrzuje závislost umístění serveru.

7.7 500 MB

Tabulka 8 – Výsledná statistika – 500 MB

Klient_Server	Průměr (ms)	Směrodatná odchylka (ms)	Min (ms)	Max (ms)
WIN_WIN	8158,67	2708,44	5625	12422
LINUX_LINUX	4554,63	104,54	4394	5509
WIN_LINUX	4681,80	145,04	4453	5141
LINUX_WIN	6070,13	91,71	5787	6391



Obrázek 22 – Graf kumulovaného průměru – 500 MB

Zdroj: Autor

Zde je patrný velký rozdíl měření klient a server na Windows platformě a ostatních měření.

8 Závěr

Při samotném testování se naskytlo několik problémů. Na naměřené hodnoty mělo vliv několik nepříznivých faktorů. Často se hodnoty měnily vlivem např. restartováním hostitelského počítače nebo při opakování měření po určitém čase. Avšak hlavní dopad na testování měli samotné aplikace, kde výsledky znehodnocovalo grafické rozhraní aplikací, tzn. jednotlivé výpisy informací pro uživatele v průběhu testování, které nepochopitelně navyšovali měřený čas, i když byly prováděny mimo časově měřenou část. Z tohoto důvodu byla aplikace server vytvořena co nejjednodušší, tedy jako konzolová a aplikace klient má možnost vypnutí veškerých výpisů, aby výsledky byly co nejpřesnější. Celkově jsem i přes problémy během testování s výsledkem práce spokojen.

Podle výsledků při nižší zátěži závisí na jaké platformě, je umístěn klient. Pokud je klient na Windows platformě, je podle směrodatné odchylky znát menší stabilita výsledků, než když je na Linux platformě. Průměrné hodnoty jsou podobné.

U střední zátěže začínají být rozdíly ve výsledných průměrných hodnotách, kde se naopak ukazuje závislost na umístění serveru. Server se s navyšující zátěží lépe vypořádá na Linux platformě.

Při vysoké zátěži jsou směrodatné odchylky podobné, kromě měření klient a server na Windows platformě, kde je odchylka, ale také průměr výrazně vyšší. Výsledky průměrných hodnot jdou ve prospěch serveru na Linux platformě.

Celkově tedy vychází v testování TCP komunikace lépe Linux platforma, která je na straně klienta stabilnější při nižší zátěži a na straně serveru při vyšší zátěži lepší v průměrných výsledných hodnotách. Vysoké zhoršení výsledných hodnot při vysoké zátěži je pozorováno u kombinace klient a server na Windows platformě.

Literatura

1. **Dostálek, L. a A.Kabelová.** *Velký průvodce protokoly TCP/IP a systémem DNS.* Praha : Computer Press, 2000. ISBN 80-7226-323-4.

2. **Wikipedie.** IP fragmentace. [Online] 10. 2 2012. [Citace: 3. 5 2012.] http://cs.wikipedia.org/wiki/IP_fragmentace.

3. **Kristoff, John.** The Transmission Control Protocol. [Online] 24. 4 2000. [Citace: 3. 5 2012.] http://condor.depaul.edu/jkristof/technotes/tcp.html.

4. **Corporate, Nokia.** Qt Whitepaper. [Online] 2. 1 2012. [Citace: 4. 5 2012.] http://qt-project.org/wiki/QtWhitepaper.

5. **Wikipedie.** Framework. [Online] 11. 4 2012. [Citace: 6. 5 2012.] http://cs.wikipedia.org/wiki/Framework.

6. —. Qt (knihovna). [Online] 29. 3 2012. [Citace: 4. 5 2012.] http://cs.wikipedia.org/wiki/Qt_(knihovna).

7. **Corporate, Nokia.** Qt Licensing. [Online] [Citace: 4. 5 2012.] http://qt.nokia.com/products/licensing.

8. —. Signals & Slots. [Online] [Citace: 7. 5 2012.] http://qt-project.org/doc/qt-4.8/signalsandslots.html.

9. —. Qt Creator. [Online] 11. 10 2011. [Citace: 5. 5 2012.] http://qt-project.org/wiki/Category:Tools::QtCreator.

10. **Corporate, Oracle.** Oracle VM VirtualBox - User Manual. [Online] [Citace: 8. 5 2012.] http://www.virtualbox.org/manual/.

11. **Wikipedie.** VirtualBox. [Online] 21. 4 2012. [Citace: 8. 5 2012.] http://cs.wikipedia.org/wiki/VirtualBox.

12. —. Směrodatná odchylka. [Online] 15. 2 2012. [Citace: 9. 5 2012.] http://cs.wikipedia.org/wiki/Sm%C4%9Brodatn%C3%A1_odchylka.

Příloha A – Přiložené CD

Přiložené CD obsahuje zdrojové kódy aplikací, tento text ve formátu pdf a výsledky testování ve formátu MS Excel.