UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Síťový aplikační protokol s nízkou latencí

Miroslav Sobotka

Diplomová práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Miroslav Sobotka**
Osobní číslo:	**I11000**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Síťový aplikační protokol s nízkou latencí**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V teoretické části bude provedena analýza známých transportních protokolů nad IP a budou popsány jejich výhody a nevýhody při použití v aplikacích, kde je vyžadována nízká latence, jako jsou hry pro více hráčů. Následovat bude návrh aplikačního protokolu vhodný pro tuto situaci, případně i se srovnáním s podobnými dostupnými protokoly používanými ve specifikovaných aplikacích.
Praktická část se bude zabývat implementací aplikačního protokolu v jazyce C++. Výsledný protokol bude otestován ve vhodné aplikaci.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

MATTHEW, Neil, STONES, Richard. Linux: Začínáme programovat. 2000. 912 s. ISBN 80-7226-307-2.
PRATA, Stephen. Mistrovství v C++. 2007. 1120 s. ISBN 978-80-251-1749-1.
ECKEL, Bruce. Myslíme v jazyku C++ - knihovna programátora. 2000. 556 s. ISBN 80-247-9009-2.
SDL_net Documentation [online]. 2009-11-03 [cit. 2012-09-30]. URL: http://jcatki.no-ip.org:8080/SDL_net/
MATTHEW, Neil, STONES, Richard. Linux: Programujeme profesionálně. 2000. 912 s. ISBN 80-7226-307-2.

Vedoucí diplomové práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 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 22. srpna 2013

Miroslav Sobotka

Rád bych poděkoval Mgr. Tomáši Hudcovi za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

ANOTACE

Tato práce se zabývá návrhem a implementací aplikačního protokolu s nízkou latencí určeného k synchronizaci herního stavu. Rozebírá různé možnosti synchronizace a technologie, které lze k tomuto účelu využít. Finální návrh synchronizace je následně implementován a zdokumentován v závěrečné části práce.

KLÍČOVÁ SLOVA

synchronizace, hra, latence, aplikační protokol

TITLE

Network Application Protocol with Low Latency

ANNOTATION

This thesis describes the design and implementation of the application protocol with low latency specified to synchronize the game state. It analyzes various synchronization options and technologies which can be used. The final proposal of the synchronization is implemented and documented in the final part.

KEYWORDS

synchronization, game, latency, application protocol

Obsah

SEZNAM ILUSTRACÍ A TABULEK

Seznam obrázků

Seznam tabulek

SEZNAM ZKRATEK A ZNAČEK

ADSL	Asymmetric Digital Subscriber Line
CDMA	Code division multiple access
CPS	Cycle Per Second
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DNS	Domain Name System
FPS	Frame Per Second
FTP	File Transfer Protocol
GPL	General Public Licence
GPRS	General Packet Radio Service
HSPA	High Speed Packet Access
HTTP	Hyper Text Transfer Protocol
IMAP	Internet Message Access Protocol
IP	Internet Protocol
LAN	Local Area Network
RTT	Round-trip time
SDL	Simple DirectMedia Layer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System

1 Úvod

Jednou ze základních potřeb živých organismů na světě je komunikace. Mezi sebou nekomunikují však pouze živé bytosti, ale i neživé věci – například počítače. Ostatně celá počítačová síť internet představuje obrovské komunikační médium, kde si mezi sebou počítače vyměňují zprávy. Již dávno neplatí, že by byly počítače pouze samostatné oddělené jednotky. Dříve se po koupi počítač přinesl domů, spustil a mohlo se začít pracovat. Nyní však stojí před začátkem práce ještě jeden neodmyslitelný krok a to právě připojení k internetu. Existuje mnoho možností, jak se k němu připojit. Nejen z pohledu použité technologie, ale také ve způsobu vzájemné komunikace. I lidé různých národností musí nejprve určit jazyk, kterým budou mezi sebou mluvit a kterým si budou vzájemně rozumět. U počítačů tomu není jinak. Určité pravidlo, řád a zvyklosti v počítačovém světě definuje protokol. A protože na počítačích běží aplikace, které si potřebují vyměňovat zprávy, říká se mu aplikační protokol. V této práci bude rozebrán návrh nového aplikačního protokolu použitého ve hře, kde spolu hraje více hráčů. Cílem tohoto protokolu je udržovat herní stav u všech hráčů stejný.

V druhé kapitole je rozebrána teorie síťových protokolů a možností, jak přenášet zprávy přes počítačovou síť internet. Dále je zde rozebrána problematika latence, která neodmyslitelně patří ke komunikaci mezi čímkoliv v počítačovém světě. Třetí kapitola se zabývá konkrétním technickým řešením komunikace, která slouží jako základ následující, čtvrté, kapitoly. Zde se řeší vlastní návrh synchronizace herního stavu. V poslední kapitole pak probíhá diskuze nad výsledným řešením.

2 Teorie

V této kapitole bude probírána teorie internetových protokolů, bude vysvětlen termín aplikační protokol, možnosti návrhu aplikačního prokolu pro konkrétní účel, výhody a nevýhody různých řešení.

2.1 Internet a protokoly rodiny TCP/IP

Internet, obrovská počítačová síť, je největší informační transportní médium. Aby předáváné informace měly nějaký řád a nebyla to jen bitová anarchie, byly vymyšleny a zavedeny počítačové – internetové protokoly. V dnešním internetu zaujala přední místo rodina protokolů z architektury TCP/IP. Protokol je množina pravidel, které popisují obsah zpráv, a také pravidla, jak si mezi sebou klienti zprávy posílají. [3]

Architektura TCP/IP je členěna do 4 vrstev:

- aplikační vrstva (application layer),
- transportní vrstva (transport layer),
- síťová vrstva (internet layer) a
- vrstva síťového rozhraní (network interface).

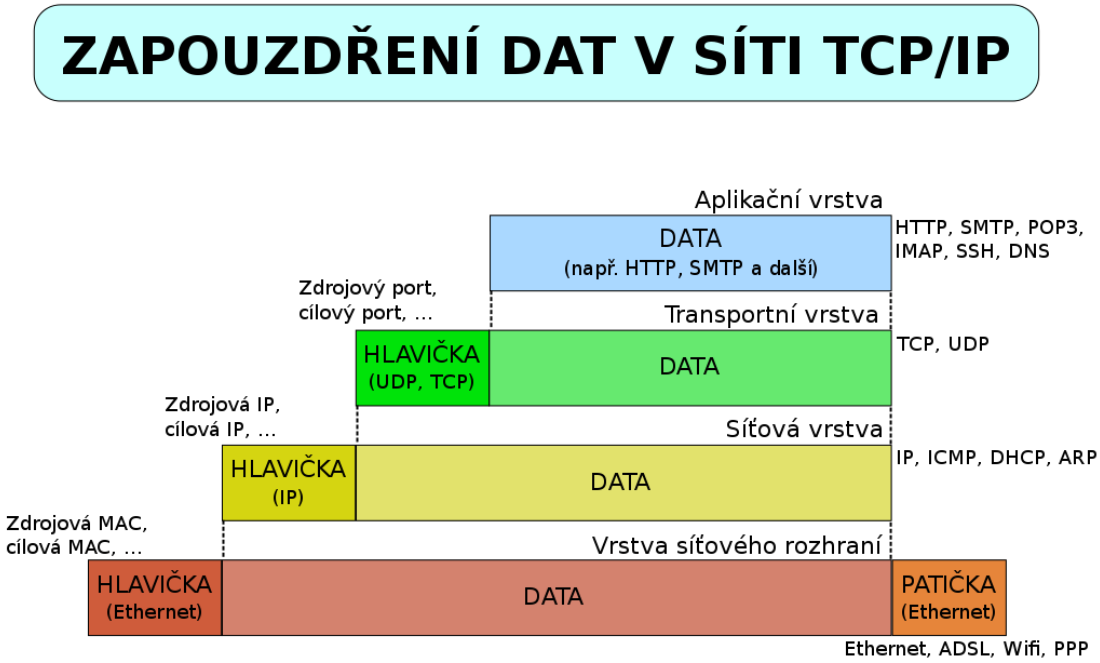
Vrstvy ilustruje obrázek 1, kde je graficky znázorněno, jak se data postupně obalují protokoly nižší vrstvy.

2.1.1 Vrstva síťového rozhraní

Příklady používaných protokolů [4]:

Ethernet je souhrnný název pro v současné době nejrozšířenější technologie pro budování počítačových sítí typu LAN. Běžné síťové protokoly (např. TCP/IP) jsou přenášeny v datové části ethernetových rámců a síťová karta jim sama o sobě nerozumí. Využívá se v něm toho, že je možné současně poslouchat i vysílat. Jako metodu boje proti kolizím se nejčastěji užívá techniky CSMA/CD, která funguje následujícím způsobem:

- Naslouchá, zda je médium (klasicky kroucená dvojlinka) volné. Jestliže není, čeká na jeho uvolnění.



Obrázek 1: Zapouzdření dat v achritektuře TCP/IP.

- Pokud je medium volné, zahájí vysílání. Současně s odesíláním rámce naslouchá, zda nepřichází signál od jiné stanice. Pokud ano, došlo ke kolizi. Stanice ukončí vysílání, odešle signál umožňující rozpoznat kolizi také ostatním stanicím.

- Vybere náhodné číslo z intervalu a podle něj čeká náhodně dlouhou dobu, než se zase pokusí vysílat. V praxi je problémem omezená doba mezi vysíláním a generováním kvalitního náhodného čísla.

Wifi je dnes dominantní protokol pro přenos dat vzduchem pomocí rádiových vln a pro budování malých, lokálních, sítí. Používá bezlicenční pásma, což snižuje její cenu. U rádiových vln není možný současný poslech a vysílání, a proto je třeba použít trochu jiné technologie než u ethernetu. Není možné kolize prostě detekovat, je třeba jim předcházet. K tomu se používá CDMA/CA, kterou lze opět popsat ve třech krocích:

- Je-li médium volné po určenou dobu, může stanice zahájit vysílání. Pokud je vysílání neúspěšné, zahájí exponenciální čekání.

- Pokud je médium obsazeno, počká na jeho uvolnění a následně zahájí exponenciální

čekání, stejně jako při neúspěšném odvysílání.

- Exponenciální čekání znamená odložený pokus o vysílání. Stanice si náhodně vybere dobu z intervalu, jehož velikost se během opakovaných pokusů zdvojnásobuje; to snižuje pravděpodobnost příští kolize.

Bluetooth je poměrně komplexní protokol, který se primárně nestará o připojení počítačů na internet (i když to také umí), ale je určen na propojení počítačů, telefonů a dalších zařízení na krátkou vzdálenost. Klasickými příklady může být komunikace tiskárny a počítače nebo mobilního telefonu a bezdrátového sluchátka. Podobně jako u Wifi jde o bezdrátovou technologii, která je navíc silně orientována na nižší spotřebu energie. Využívá konceptu master-slave: v síti je jeden význačný prvek (téměř náhodně vybrané zařízení) a ten řídí celou síťovou komunikaci. To ji dělá přehlednou a dobře organizovanou, ale také poměrně pomalou, co se přenosové rychlosti týče. Bluetooth se sám stará o komunikaci uvnitř sítě, řeší zabezpečení, přenos souborů atp. Je vhodný například tam, kde chceme rychle a jednoduše ustanovit síť s malou spotřebou – komunikace dvou mobilních telefonů, sluchátka atp. Přenos dat probíhá formou přepojování paketů v asynchronním režimu ACL (Asynchronous Connectionless Link), kde každý paket může mít délku jednoho, tří nebo pěti časových intervalů. Přenos řeči se uskutečňuje v synchronním režimu SCO (Synchronous Connection Oriented). Digitalizovaný hovorový signál je zde rovněž členěn do časových intervalů, v každém intervalu je přenesen jeden paket a práce s delšími pakety zde není možná. V každém z těchto případů je paket přenášen na jediné frekvenci, určené prvním časovým intervalem. V případě kolize na jedné frekvenci se paket pošle znovu v následujícím časovém intervalu, kdy již bude frekvence obou vysílačů s velkou pravděpodobností odlišná.

2.1.2 Síťová vrstva

Na síťové vrstvě se nachází zřejmě nejdůležitějších protokol – IP. V současnosti existuje ve dvou verzích a to IPv4 a IPv6.

Provádí vysílání datagramů na základě síťových IP adres obsažených v jejich záhlaví. Poskytuje vyšším vrstvám síťovou službu bez spojení. Každý datagram je samostatná datová jednotka, která obsahuje všechny potřebné údaje o adresátovi i odesilateli a pořadovém čísle datagramu ve zprávě. Datagramy putují sítí nezávisle na sobě a pořadí jejich doručení nemusí

odpovídat pořadí ve zprávě. Doručení datagramu není zaručeno, spolehlivost musí zajistit vyšší vrstvy (TCP nebo příslušná aplikace).

IP se dále stará o segmentaci a znovusestavení datagramů do resp. z rámců podle protokolu nižší vrstvy (např. ethernet).

Je zajímavé si povšimnout toho, že rodiny protokolů vytvářejí právě na síťové vrstvě střed pomyslných přesýpacích hodin. Ať jdeme ze síťové vrstvy na kteroukoli stranu, počet protokolů postupně narůstá.

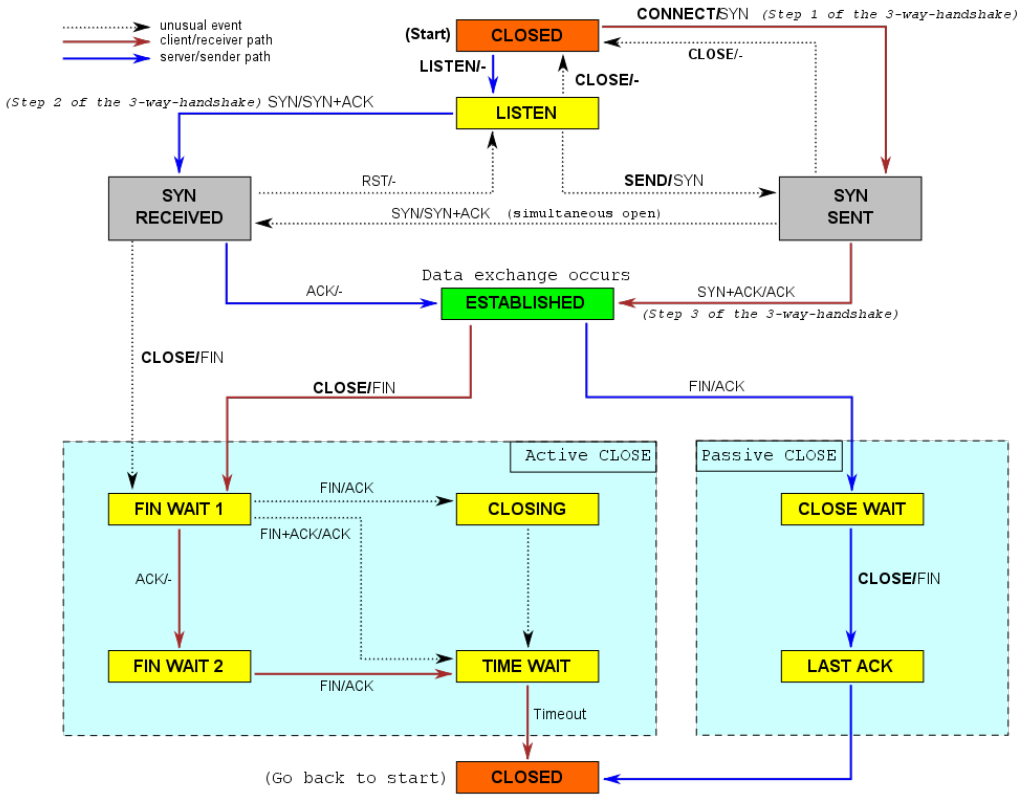
Další protokol, který lze zmínit, je podpůrný protokol ICMP, který má na starosti zasílání chybových zpráv či tvorbu cest mezi směrovači, kterým pomáhá tvořit směrovací tabulky.

2.1.3 Transportní vrstva

Tato vrstva zajišťuje přenos dat mezi koncovými uzly, tedy klienty. Jejím účelem je poskytnout takovou kvalitu přenosu, jakou požadují vyšší vrstvy. Rozhoduje se na ní, zda bude přenos dat spolehlivý či nikoliv, řeší problémy s výpadky sítě či optimalizuje přenosovou rychlost. V zásadě lze hovořit o dvou základních protokolech, které vládnou této vrstvě internetu. [8]

TCP zajišťuje spolehlivý přenos dat, to znamená, že cílem je dodání všech odeslaných paketů ve správném pořadí. Hlavní kritérium je tedy doručení, nikoli rychlost. To je typicky požadováno u přenosu souborů, e-mailů, WWW stránek atd. Mimo jiné se tedy stará nejen o doručování a kontrolu toho, zda vše chodí, jak má, ale snaží se také optimalizovat rychlost přenosu. Jde o dominantní protokol na této vrstvě. Na obrázku 2 je zobrazen statový diagram, na kterém je dobře vidět, jak dokáže TCP protokol zajistit správné doručení.

UDP je zástupcem klasického nespojovaného přístupu. Používá se tam, kde je nejdůležitější rychlost a nemá smysl kontrolovat doručení – příkladem může být IP telefonie (VoIP) nebo video stream. Doručení správných paketů je zbytečné, neboť by došlo k prodlení, což je silně nežádoucí. Proto se používá rychlejší UDP s tím, že případné chyby jsou odstraňovány softwarově (interpolace, prokládání snímků, snížení kvality atp.).



Obrázek 2: Statový diagram komunikace přes protokol TCP.

2.1.4 Aplikační vrstva

Představuje přímou komunikaci programů pomocí protokolů nižší vrstvy. Příkladů této vrstvy existuje nepřeberné množství. Například DNS, DHCP, FTP, HTTP, IMAP atd. Programy si už pak vyměňují zprávy a nemusí řešit komunikaci na nižší úrovni. O to se právě starají protokoly na nižších úrovních

V této diplomové práci bude popsán návrh právě protokolu této vrstvy.

2.2 Latence

Latence je jednoduše řečeno definována jako zpoždění mezi vysláním požadavku a odpovědí na něj, které vzniká při průchodu zprávy prostředím sítě. Mezi běžné klíčové parametry síťového spojení, které uživatelé preferují, patří pouze přenosová rychlost downlinku (data proudící do našeho počítače) a uplinku (data proudící z našeho počítače), nanejvýše pak agregace linky (sdílení více uživateli). Tyto parametry jsou obecně dobře známy. Často opomíjeným pojmem je ale tzv. latence. Ta vyjadřuje zpoždění odezvy vzdáleného počítače (či

síťového zařízení), tedy dobu, za kterou paket (blok dat) vyslaný ze zdroje dorazí k cíli a zpět (např. z našeho počítače na server a zpět). Tato doba se měří obvykle v milisekundách. Výše popsaná latence bývá také označována jako Round Trip Time latency a existuje i varianta nazvaná One Way Latency, což je doba za kterou paket vyslaný ze zdroje dorazí k cíli bez odesílání zpět. Logika by naznačovala, že obousměrná latence se bude rovnat jednosměrné vynásobené dvěma, ale díky nesymetrickému spojení to nemusí být vždy pravda. Paket nemusí jít zpět stejnou cestou jako směrem k serveru ale může projít jiným počtem uzlů v síti. Díky tomu může být pozdržen, a nebo naopak dorazit dříve, protože více uzlů ne nutně znamená větší latenci. Analogií může být cestování autem po silnicích, kdy cestou k serveru jedeme po regionálních silnicích a cestou zpět jedeme po dálnicích. [10]

Latence tedy významně ovlivňuje kvalitu připojení k internetu, u některých služeb pak dokonce natolik, že její vysoké hodnoty jejich využívání zcela znemožňují. Mezi služby citlivé na hodnotu latence patří například hraní her po síti či telefonování po internetu (např. v aplikaci Skype). Zde latence některých mobilních připojení činící až několik stovek milisekund zcela zabraňuje jakémukoliv rozumnému užití těchto služeb. Např. pro telefonování po internetu se nejvyšší snesitelná latence RTT pohybuje kolem 250 ms. Nicméně i v ostatních internetových službách vysokou latenci uživatel negativně pocítí – dokonce i při zobrazení pouhé webové stránky, jíž mnohdy tvoří obrázky a objekty z různých serverů se tak může zpozdit zobrazení stránky na několik sekund bez ohledu na rychlost downloadu či uploadu, kterou máme k dispozici (např. zobrazení jedné webové stránky i přes rychlé 3G mobilní připojení UMTS či HSPA může při vysoké latenci trvat několik sekund namísto zlomku sekundy při latenci nízké). Úroveň latence je dána samotnou technologií připojení, ale i kvalitou sítě. Vliv latence se tedy projeví nejvíce u požadavku s velkým počtem málo objemných prvků, kdy je pro načtení třeba mnohokrát provést potvrzovací cyklus. [1]

Problém s latencí odpadá obecně v případě, kdy přenášíme větší objem dat najednou. Ten se jednoduše počítačovou sítí „protlačí“. Pokud jde o prohlížení webových stránek, je lépe přenášet např. jeden větší obrázek než mnoho malých. U služby wap se problém zkoušel řešit tak, že se kompletní prezentace zabalila do jednoho celistvého elementu, který se přenášel najednou. Problémy s latencí odpadají také u technologie Blackberry, kdy se celý e-mail stáhne a zpracovává se pak lokálně. Zvýšení kvality služeb z hlediska odezvy tudíž nevězí jen ve zvyšování propustnosti sítě např. sdružováním kanálů, ale v komplexním zlepšení a zefektivnění celé přenosové trasy a všech procesů v ní. [5]

Technologie připojení	Hodnota latence
ADSL	53,75 ms
WiFi	36,95 ms
Kabelová TV	34,11 ms
GPRS, EDGE (technologie 2G)	498,03 ms
CDMA, UMTS, HSPA (technologie 3G)	208,78 ms

Tabulka 1: Vybrané naměřené hodnoty latence dle technologie připojení (zdroj: internetprovsechny.cz, rychlost.cz)

Některé webové prohlížeče pak zkoušejí komunikaci komprimovat a sdružovat do větších celků. Nejdále pak došel pravděpodobně internetový prohlížeč Opera pro mobilní telefony, který komprimuje jak webovou stránku, tak obrázky a vše odesílá jako celek. Problémy s latencí se tak výrazně potlačují a ještě se ušetří nemalá část přenesených dat. [7]

Vybrané naměřené (průměrné) hodnoty latence dle technologie připojení ilustruje tabulka 1.

Na obrázku 3 je jasně zobrazen výše popsaný vliv latence na dobu načtení webové stránky (pro příklad byla zvolena stránka Wall Street Journal) Na obrázku 4 je ještě detailněji ukázán vliv různé úrovně latence na načtení této stránky.

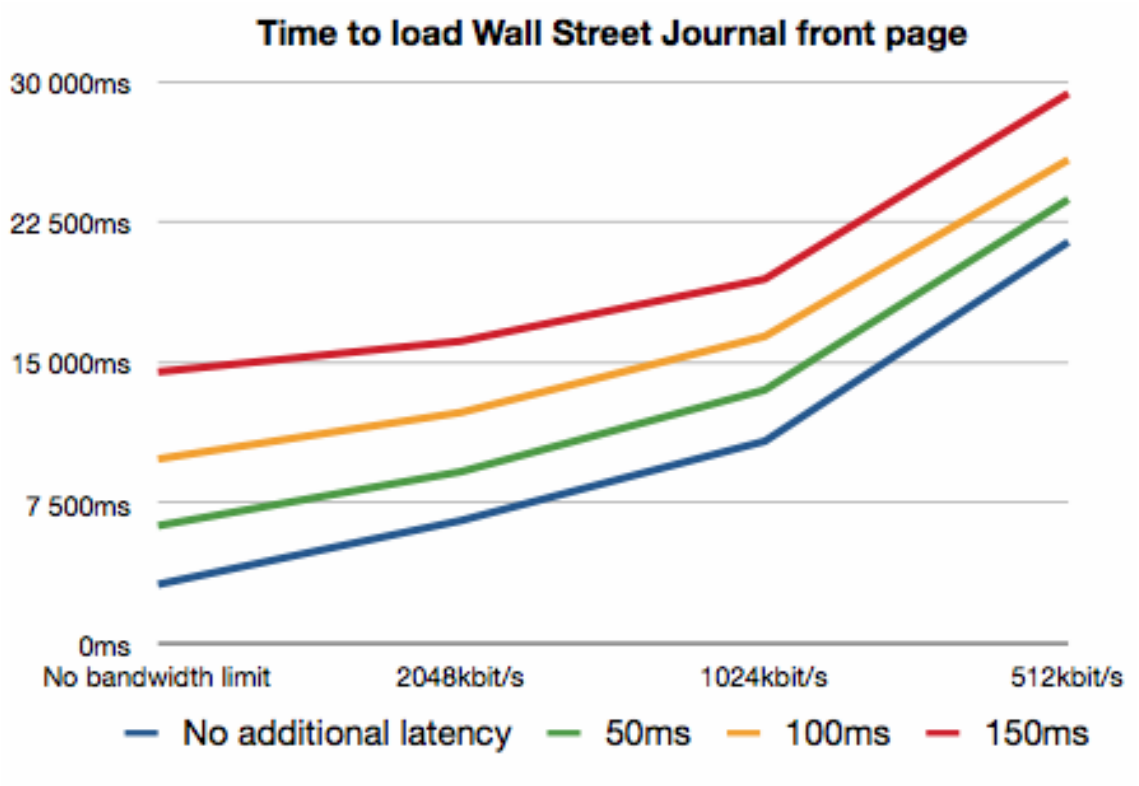
2.3 Aplikační protokol

Protokol patřící do aplikační vrstvy achritektury TCP/IP. Jeho obsahem jsou vlastní data, která si mezi sebou aplikace vyměňují. Obsah zpráv již není žádným standardem definován, záleží tedy čistě na aplikacích. Avšak přesto by se daly protokoly dělit na pár skupin. Prvním příkladem je typ komunikace:

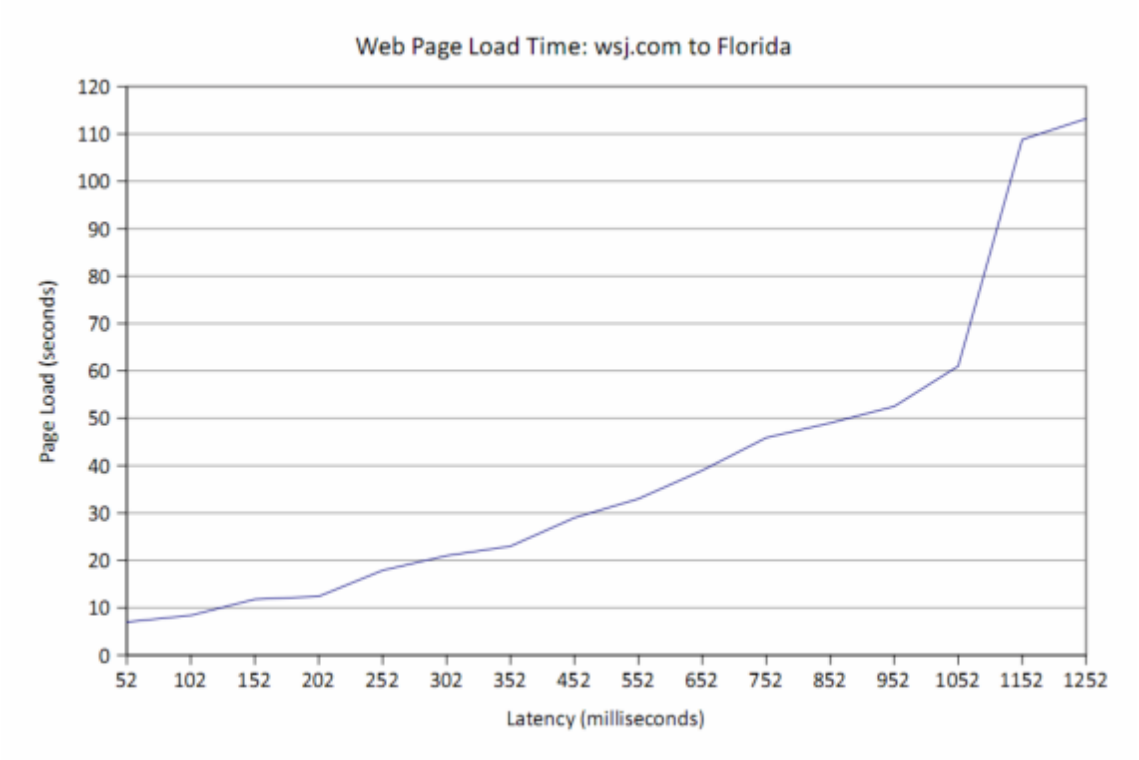
- synchronní (položím dotaz a čekám na odpověď), nebo
- asynchronní (pokládám dotazy a při tom poslouchám odpovědi).

Dále můžeme řešit obsah:

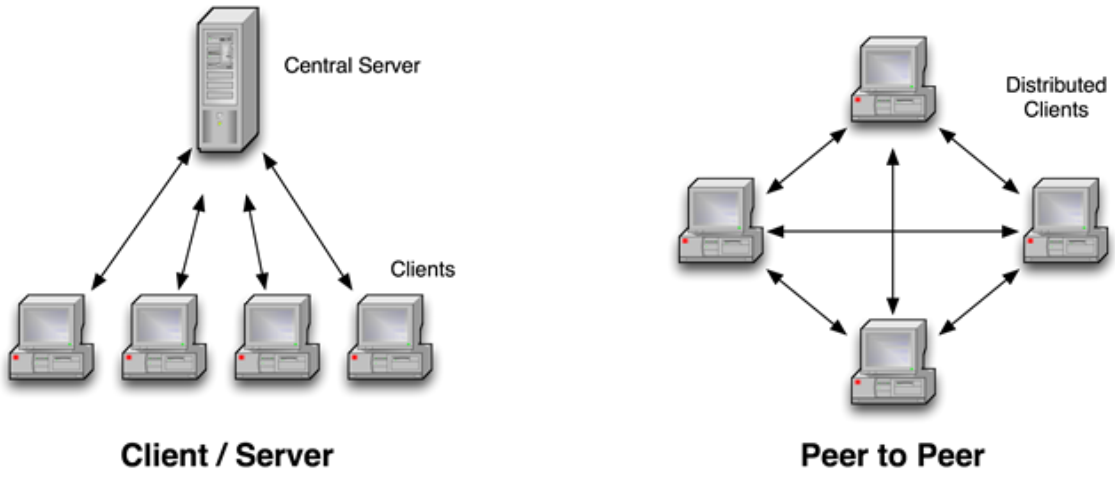
- textový (obsah zpráv je dobře čitelný, obsahuje slova), nebo
- binární (obsahuje nějaká konkrétní data, která však nejsou na první pohled čitelná).



Obrázek 3: Vliv latence na rychlost načtení stránky (zdroj: webkit.org).



Obrázek 4: Vliv latence na rychlost načtení stránky (zdroj: www.o3bnetworks.com).



Obrázek 5: Dvě možné řešení spojení klientů v počítačové síti. Architektura klient-server (Client-Server) a klient-klient (Peer to Peer) [9]

V binární podobě zpráva obsahuje zpravidla číselné hodnoty, podle kterých se klient přímo rozhoduje. U textové zprávy se nejdříve musí načíst obsah, textové příkazy převést na jejich číselné hodnoty a zpracovat. Výhodou textových zpráv je jejich dobrá čitelnost a z toho plynoucí výhoda při ladění programu. Avšak binární zprávy mohou ušetřit dost dat, protože jejich velikost se srovnatelným obsahem je zpravidla zlomková vůči textovým.

2.4 Možnosti propojení klientů

V této kapitole budou popsány dvě možné řešení spojení klientů v počítačové síti. Na obrázku 5 je graficky znázorněno spojení mezi klienty a jejich role v případě architektury klient-server.

2.4.1 Klient-server

Klient-server je síťová architektura, která odděluje klienta a server, kteří spolu komunikují přes počítačovou síť. Klient-server aplikace obsahují jak klienta, tak i server. Alternativou architektury klient-server je klient-klient.

Klient-server architektura popisuje vztah mezi dvěma počítačovými programy, v nichž první program – klient, žádá o služby jiný program, zvaný server. Na tomto modelu je založeno mnoho známých služeb, například E-mail, Web, přístup k databázi apod.

Příkladem takové služby je webový prohlížeč, který je v tomto případě roven klientskému

programu na straně uživatele – webovému prohlížeči, který může přistupovat k informacím na libovolném webovém serveru na světě připojeného do internetu. Modelová situace s prohlédnutím jedné webové stránky pak probíhá například takto: Uživatel zadá do webového prohlížeče adresu. V tomto případě je tedy klientem webový prohlížeč. Prohlížeč se spojí s webovým serverem podle zadané adresy a předá mu požadavek na stránku, kterou chce uživatel vidět. Webový server musí získat z databáze zdroj stránky, který bude odesílat zpět. Proto se webový server přes integrovaného databázového klienta spojí s databázovým serverem a zašle požadavek na stránku. Databázový server najde ve svém uložišti stránku a posílá ji zpět webovému serveru. Ten kompletuje odpověď a odesílá ji zpět webovému prohlížeči. Po obdržení zprávy výsledek zobrazí uživateli.

Model klient-server se stal velice populární a byl pojat jako jedna z hlavních myšlenek síťové technologie. Používá ho například většina obchodních či firemních aplikací (například protokol SOAP), avšak je základem i pro obecné protokoly, například internetové protokoly HTTP, SMTP, Telnet, DNS, apod.

Klient může posílat žádost o data jednomu nebo více serverům. Není nutno se však omezovat pouze na jeden požadavek vůči jednomu serveru v jeden čas. Pokud je například webový server nastaven tak, že má omezenou rychlost přenosu dat v rámci jednoho spojení, lze pak díky navázání dalšího spojení zvýšit propustnost dat. V případě připojení na více různých serverů v jeden čas je pak ovšem potřeba dávat pozor, zda servery čerpají ze stejného zdroje dat. Pokud by zdroj dat nebyl stejný mohlo by se stát, že výsledná data budou neplatný mix dat. Tento nedostatek lze na aplikační úrovni vyřešit elegantně tak, že se data předem dělí na větší logické celky (bloky) a pro ty se spočítá otisk (anglicky hash) pomocí hashovací funkce. Ten se u klienta spočítá též a kontroluje se shoda. Výhodou je, že se nemusí kontrolovat kompletní obsah vůči originálu, ale jen otisky, které jsou mnohonásobně menší.

Klient i server obsahuje pouze dvě části: serverovou a klientskou. Tento typ architektury je někdy označován jako two-tier (dvouvrstvá). Umožňuje zařízením sdílet soubory nebo jiné zdroje.

Nejčastější klienti jsou dnes webové prohlížeče. Servery jsou většinou webové servery, databázové servery a e-mailové servery, mohou se sem počítat i herní servery. V případě MMORPG (anglicky Massive-Multiplayer Online Role-Playing Game) provozuje server výrobce hry, u dalších typů her jako server slouží jeden z hráčů, který spustí hru v serverovém

módu (tzv. Host).

Výhody přístupu klient-server

Ve většině případů architektura klient-server rozdělí jednotlivé úkoly a zodpovědnosti počítačového systému mezi několik počítačů které spolu komunikují pouze prostřednictvím sítě. Tím vzniká další výhoda této sítě a to snadnější údržba. Například je možné nahradit, opravit, modernizovat, přemístit server aniž by to klienti poznali nebo tím byli nějak ovlivněni. Tato nezávislost na klientech se nazývá zapouzdření.

Všechny údaje jsou uloženy na serverech, které jsou mnohem bezpečnější než většina klientů. Servery mohou lépe kontrolovat přístup a zdroje. To zaručuje, že přistupovat a měnit data mohou pouze oprávnění klienti.

Vzhledem k tomu, že se data ukládají centralizovaně, aktualizace údajů je mnohem jednodušší než u klient-klient sítí. V klient-klient sítích je potřeba updatovat data na každé stanici zvlášť, což je pomalé a způsobuje množství chyb, protože mohou být tisíce nebo miliony klientů. Počet chyb roste s počtem klientů.

Mnoho klient-server aplikací, které jsou dnes k dispozici, jsou navrženy s ohledem na vyšší bezpečnost, uživatelskou přívětivost a snadné používání.

Nevýhody přístupu klient-server

Velkým problémem je přetěžování sítě. Vzhledem k tomu, že počet souběžných požadavků klientů na daný server se zvyšuje, server se může snadno přetížit. Naproti tomu u klient-klient sítí se šířka pásma zvětšuje s množstvím klientů, protože každý klient tvoří uzel sítě.

Architektura klient-server není tak robustní jako sítě klient-klient. Pokud dojde k výpadku serveru, žádosti klientů nemohou být splněny. V klient-klient sítích jsou zdroje obvykle distribuovány mezi více uzlů. Dokonce i když více uzlů přeruší sdílení dat, mělo by být možné stáhnou data od zbývajících uzlů.

2.4.2 Klient-klient

Klient-klient, anglicky peer-to-peer (zkráceně P2P) známená doslova rovný s rovným, je označení typu počítačových sítí, ve kterých spolu komunikují přímo jednotliví klienti (uži-

vatelé). Opakem je síť klient-server, ve které jednotliví klienti komunikují vždy s centrálním serverem či servery. Jejich prostřednictvím také komunikují s jinými klienty, pokud je to potřeba. Čistá klient-klient architektura vůbec pojem server nezná, všechny uzly sítě jsou si rovnocenné (a působí současně jako klienti i servery pro jiné klienty). V praxi se však často pro zjednodušení návrhu v protokolu objevují specializované servery, které ovšem slouží pouze pro počáteční navázání komunikace, „seznámení“ klientů navzájem, popř. jako proxy server v případě, že spolu z nějakého důvodu nemohou koncové uzly komunikovat přímo.

Dnes se označení klient-klient vztahuje hlavně na výměnné sítě, prostřednictvím kterých si mnoho uživatelů může vyměňovat data. Příkladem takových sítí jsou např. Gnutella či původní Torrent.

Jednou ze základních výhod klient-klient sítí je fakt, že s rostoucím množstvím uživatelů celková dostupná přenosová kapacita roste, zatímco u modelu klient-server se musí uživatelé dělit o konstantní kapacitu serveru, takže při nárůstu uživatelů klesá průměrná přenosová rychlost. Příkladem tohoto efektu je rozdíl mezi stahováním souboru v architektuře klient-server, kde s každým klientem všem klesne maximální rychlost. Na rozdíl od použití protokolu Torrent, který je směsí klient-server a klient-klient. Server, nazývaný též Torrent Tracker, udržuje informace o souboru a další klienti se na něj připojují. Důležité je, aby v síti byl připojen alespoň jeden uživatel, který má dostupný celý soubor. Od něho pak další klienti stahují. Ty části původního souboru od uživatele, který má kompletní soubor, jež jsou již stažené u dalšího klienta, jsou následně okamžitě dostupné pro ostatní. Rychlost stahování se tak může postupně zvyšovat, protože každý klient může nabídnout další kopii původního souboru.

3 Návrh technického řešení distribuce zpráv mezi klienty

V níže uvedených bodech je popsáno, jaké předpoklady byly vzaty v úvahu před budováním synchronizační strategie. Bližší postup je pak popsán v následující kapitole.

3.1 Transportní protokol

Jako transportní protokol s ohledem na knihovnu SDL_Net se nabízely dvě možnosti: TCP nebo UDP. Bylo rozhodnuto, že bude využit protokol TCP. Na základě dalšího experimentování a vylepšování návrhu synchronizace nakonec bylo usouzeno, že jako další logický krok bude převedení protokolu z TCP na UDP, případně využití obou a tedy těžení výhod z každého protokolu.

3.2 Aplikační protokol

Zprávy budou předávány asynchronně a jejich obsah bude distribuován v binární podobě.

3.3 Typ propojení klientů

Na začátku bylo rozhodnuto, že bude využita architektura klient-klient. Avšak při budování strategie synchronizace se zjistilo, že tento model není vhodným kandidátem a proto bylo rozhodnuto o změně na architekturu klient-server.

3.4 Odesílatel a příjemce

Nabízelo se více možností, jak realizovat architekturu klient-server. Mezi tyto možnosti patří například:

- První fyzický program – server a druhý fyzický program – klient, který lze spustit ve více instancích.
- Jeden fyzický program, který v sobě integruje jak klienta, tak server. Až po spuštění je rozhodnuto, v jakou roli se program promění.

- Jeden fyzický program, který v sobě integruje jak klienta, tak server. Na rozdíl od předchozí možnosti se však server spustí vždy. To znamená, že i když se nekomunikuje přes počítačovou síť s jiným programem, komunikace již probíhá přes síťový protokol.

Jako cílové řešení byl zvolen mix z výše uvedeného. Program v sobě integruje server i klienta. Server se spustí vždy, i když není třeba komunikovat s dalším programem přes počítačovou síť. Pokud komunikuje klient se serverem v rámci jednoho fyzického programu, používá stejné typy zpráv, jako kdyby komunikoval přes počítačovou síť. Jediným rozdílem je pak to, že zprávy se neposílají přes počítačovou síť, ale jsou rovnou předány ke zpracování. Při potřebě synchronizace s více klienty přes počítačovou síť se pak pouze aktivuje v serveru naslouchání na definovaném portu a není třeba příliš upravovat chování komunikace. Tím se také značně zredukovala délka výsledného kódu.

4 Návrhy řešení sychronizace

Aplikační protokol navrhovaný v této diplomové práci bude sloužit primárně jako synchronizační nástroj mezi dvěma či více klienty. Jako testovací aplikaci jsem vytvořil remake online akční hry Atomic Bomberman.

Ve hře Atomic Bomberman se pohybují hráči po herní ploše a mohou pokládat bomby, jejichž cílem je likvidace nepřítele. Na obrázku 6 je zobrazena ukázka ze hry. Vzhledem k tomu, že je celkově hra velmi rychlá a změn, byť malých, je poměrně hodně za krátký čas, vyžaduje co nejrychlejší reakce při síťové hře.

Obrázek 6: Ukázka hry

V první řadě je potřeba vyřešit změnu pozice hráčů, pokládání a exploze bomb.

Standardní herní smyčka se skládá z několika jednoduchých kroků a vypadá následovně:

```
// 0) Nastav příznak stavu programu na Běží
bool game_is_running = true;
// 1) Začátek smyčky
do {
// 2) Načti vstupy (klávesnice, myš, sít', ...)
  check_input();
// 3) Zpracuj vstupy (přesuň hráče, polož bomby, přepni menu, ...)
  update_game();
// 4) Překresli scénu (herní, menu, ...)
  display_game();
// 5) Pokud je příznak stavu programu na Běží,
//    skoč na bod 1), nebo skoč na 6)
} while( game_is_running )
// 6) Konec programu
```

Klasicky tedy v bodě 2) program načte stav stisknutých kláves a podle jejich kombinace zvolí následnou strategii. Příklad: Hráč zmáčkne klávesu Šipka dolů a program ví, že tato událost má vyvolat změnu pozice hráče směrem dolů. To provede úpravou hodnoty proměnné Y, za předpokladu, že proměnné X a Y vyjadřují absolutní pozici hráče na herní ploše.

4.1 Řešení synchronizace: 1. návrh

Byla sestavena pokusná aplikace s herní smyčkou uvedenou výše a byl objeven první problém, který spočívá především v efektivitě. Herní smyčka může stihnout během vteřiny udělat mnoho stovek i více cyklů. Z toho vyplývá, že událostí jen na změnu pozice jednoho hráče bude příliš mnoho. Nehledě na fakt, že čas jedné smyčky se může lišit podle rychlosti počítače. [6]

Problém: Rychlost posunu hráče je přímo úměrná rychlosti vykonávání herní smyčky a ta je závislá na výkonu počítače.

Řešení: Upravení herní smyčky – zavedení konstantního počtu kroků za čas hráče na herní ploše nehledě na rychlost počítače.

Jednotlivé kroky v herní smyčce potřebují různý čas na zpracování. Především pak překreslení scény je podle výsledků profilování nejnáročnější krok z pohledu času. O to víc

se bude projevovat na pomalejších počítačích, například pomalejším pohybem hráče. Jak vyplývá z výše uvedené herní smyčky, počet snímků za vteřinu (ve smyslu náročného překreslení scény) není v obecné herní smyčce nijak omezen a je tedy přímo úměrný rychlosti zpracování počítačem. Jednoduchý trik, jak toto vyřešit, spočívá v tom, že určíme pevný počet cyklů za vteřinu (CPS), aby pohyb byl plynulý a především konstantní. Herní smyčku pak rozdělíme na dvě části – první: načítání a úprava scény a druhou: kreslení. První část bude smyčka a bude se vykonávat maximálně takový čas, který je úměrný času CPS. Až pak může dojít na překreslení scény. Výsledkem tohoto opatření je, že smyčka bude poskytovat konstantní časovou aktualizaci pozic, zatímco počet snímků za vteřinu (FPS) bude přímo úměrný rychlosti počítače. Řešení v jazyce C++ vypadá následovně:

```
// pevný počet CPS
const int TICKS_PER_SECOND = 50;
// kolik času může trvat jeden cyklus
const int SKIP_TICKS = 1000 / TICKS_PER_SECOND;
const int MAX_FRAMESKIP = 10;

// ulož aktuální čas
int next_game_tick = GetTickCount();
// počítadlo cyklů
int loops;

bool game_is_running = true;
while( game_is_running ) {
  loops = 0;
  // dokud ještě nevypršel čas dalšího cyklu a zároveň nebylo
  // vynecháno moc překreslení scény, aby nedocházelo k trhání hry
  while( GetTickCount() > next_game_tick && loops < MAX_FRAMESKIP)
  {
    // načti vstup
    check_input();
    // uprav scénu
```

```
    update_game();
    next_game_tick += SKIP_TICKS;
    loops++;
  }
  // překresli scénu
  display_game();
}
```

Původní řešení zpracování událostí (např. stisknutí/nestisknutí klávesy) spočívalo ve zpracování každé události, bez ohledu na to, zda byla totožná s předchozí událostí. Pokud by se toto množství událostí posílalo jednotlivě přes počítačovou síť, mohlo by dojít k velkým prodlevám.

Problém: Příliš mnoho duplicitních událostí jdoucích za sebou.

Řešení: Redukce počtu událostí a zavedení fronty zpráv.

Fronta zpráv funguje tak, že při načtení a zpracování vstupů se neupravuje scéna přímo, ale nepřímo přes zprávu do fronty. V momentě načtení vstupu v rámci herní smyčky se porovnává nový vstup s předešlým. Až v případě, že nastala změna, se vygeneruje zpráva o změně a přidá se do fronty. V dalším kroku, upravení scény, se postupně vybírají zprávy z fronty a upravuje se scéna. Výhodou tohoto řešení je i nezávislost na tom, kdo bude frontu plnit. Může to být hráč – klávesnice, počítač – algoritmus, nebo síťový hráč – síť.

Příkladem takových zpráv jsou:

- `Jdi(směr)`, kde *směr* určuje směr, kterým se má hráč vydat a
- `Stůj()`, která zastaví hráče.

Tabulka 2 ukazuje původní řešení. V tabulce 3 je pak znázorněno nově navržené řešení. Šipka za zmáčknutím klávesy znázorňuje přímé volání funkce.

Toto řešení vedlo ke snížení počtu kroků a zároveň zkrácení času mezi zpracováním událostí a úpravou herní scény.

Tím je základ aplikace připraven na synchronizaci mezi klienty. Zpráva, která je vygenerována při zpracování události, je zařazena do lokální fronty zpráv. Pokud je připojen další klient přes počítačovou síť, je odeslána do jeho fronty zpráv.

Posloupnost událostí
1. smyčka
(vstupy) klávesa dolů ⟶ hodnota Y (pozice hráče) se změní
2. smyčka
(vstupy) klávesa dolů ⟶ hodnota Y (pozice hráče) se změní
3. smyčka
(vstupy) klávesa dolů ⟶ hodnota Y (pozice hráče) se změní
4. smyčka
(vstupy) klávesa dolů ⟶ hodnota Y (pozice hráče) se změní
5. smyčka
(vstupy) nic

Tabulka 2: Původní způsob přes přímou úpravu proměnných

Posloupnost událostí
1. smyčka
(vstupy) klávesa dolů je zmáčknuta, zpráva do fronty: `Jdi(dolů)`
(úprava scény) výběr zprávy ⟶ hodnota Y (pozice hráče) se změní
2. – 4. smyčka
(vstupy) klávesa dolů je stále zmáčknuta, duplicitní zpráva se neodesílá
(úprava scény) hodnota Y (pozice hráče) se změní
5. smyčka
(vstupy) klávesa dolů již není zmáčknuta, zprávy do fronty – `Stůj()`
(úprava scény) nic

Tabulka 3: Nový způsob přes předáváná zpráv přes frontu

První testování probíhalo tak, že v rámci aplikace byl každý klient obsluhován, jako by byl síťový klient a přidán byla též jedna umělá inteligence. Výsledek testování objevil další problém a to posloupnost zpráv ve frontě.

Problém: Změna pořadí zpráv ve frontě událostí

Řešení: Zavedení prioritní fronty včetně časového razítka zprávy, podle kterého je fronta uspořádána.

Toto řešení dokázalo zprávy uvést do správného pořadí. Toto řešení má jedno zásadní omezení: časové razítko u zpráv udává čas lokálního počítače. V rámci jednoho počítače to nečiní žádný problém. Ten nastává spuštěním aplikací na více počítačích. Každý počítač může mít rozdílně nastavené hodiny, stačí rozdíl jen několika jednotek až desítek milisekund. I kdyby měly všechny počítače však čas přesně seřízen, stále je tu problém s latencí, která navíc může kolísat.

Jako modelový případ, kdy se tento problém projevuje, poslouží pohyb hráče po herní ploše. Dva klienti, mezi jejichž počítači se latence pohybuje v intervalu jedné až dvou vteřin, spustí ve stejný čas hru. Na prvním klientovi hráč zmáčkne klávesu, bude ji 5 vteřin držet a pak ji pustí. To vygeneruje dvě zprávy: o uvedení do pohybu a zastavení hráče na herní ploše. Hráč tedy půjde po herní ploše 5 vteřin. Na klientovi, kde byla klávesa stisknuta, se hráč pohybuje okamžitě a přesně 5 vteřin, avšak na druhém počítači je to s minimálně vteřinovým zpožděním a navíc není zaručeno, že půjde 5 vteřin, ale 5 ± 1 vteřina.

Problém: synchronizace času

Řešení: odeslání synchronizační zprávy s časem 0.

Server vezme svůj aktuální čas jako referenční a odešle ostatním klientům zprávu s časem 0, tedy svůj čas, který považuje za počáteční. Klienti si jej nastaví jako počáteční čas. To znamená, že při přijetí takovéto zprávy provedou diferenci se svým aktuálním časem a tu budou přičítat ke každé zprávě. Toto řešení má však jednu zásadní podmínku: latence mezi klienty musí být rovna 0. Pokud bude větší než 0, všechny zprávy budou zatíženy chybou o hodnotě latence mezi klientem a serverem.

Problém: kolísající latence

Řešení: Jedním z řešení je, že od časového razítka (času zprávy) klient odečte svou latenci vůči serveru. Avšak i toto řešení má problém, protože latence se může a hlavně bude měnit.

Závěr: Tento typ synchronizace se ukázal jak nevhodný. Je zde problém s výběrem výkonného klienta a s latencí. Místo toho, aby se jednotlivým klientům distribuovala zpráva co se má udělat, je vhodné určit jedno místo, kde se o tom co se má udělat rozhodne. Jedno místo se určuje proto, že každý klient se může rozhodnout jinak. Místo toho aby se distribuovala zpráva co se má stát (např. požadavek jdi), distribuuje se zpráva s konkrétním obsahem (přesuň se na pozici X,Y). Následně se tato nová zpráva distribuuje do ostatních front. Klienti se mohou rozhodnout jinak vinou latence, protože zprávy které modifikují stav

hry dorazí později. Tím pádem se klient rozhoduje na základě neaktuální informace o stavu hry, tudíž špatně. Další problém se synchronizací v architektuře klient-klient je při události, která nastane ve více klientech zároveň (např. výbuch bomby). Každý klient odešle zprávu o výbuchu té samé bomby. Zpráva o výbuchu by měla existovat pouze jednou. Řešením je opět určení jednoho místa, kde se o výbuchu bomby rozhodne a odkud bude rozeslána zpráva.

Bylo nutno tedy vymyslet zcela jiný typ synchronizace.

4.2 Řešení synchronizace: 2. návrh

Druhý návrh synchronizace vychází částečně z prvního. Byl ponechán návrh úpravy herní smyčky, který udrží konstantní rychlost hry nezávisle na rychlosti počítače a byla ponechána fronta zpráv.

Prvním rozdílem je, že pokud klient z jiného počítače bude chtít pohnout hráčem, nebude odesílat dvojici zpráv `Jdi()` a `Stůj()`, ale bude posílat pouze jednu zprávu `Jdi(A,B)`, kde A a B jsou parametry, o kolik se má hráč posunout. Nevyjadřují absolutní pozici hráče, ale jen relativní posun oproti stávajícímu stavu.

Dalším rozdílem oproti prvnímu řešení je, že se místo rovnocenných klientů architektury typu klient-klient, použije architektura typu server-klient. Změna architektury se projevuje především tím, jak aplikace naloží se zprávami z fronty zpráv. Místo přímé interpretace zpráv (změny scény) je přetvoří na síťové zprávy s příznakem *Přeposlat* a aplikace je zašle serveru. Server přijme zprávu a pokud má příznak *Přeposlat*, pošle zprávu všem připojeným klientům. Což je velký rozdíl od architektury klient-klient. V ní musel klient svou zprávu všem ostatním klientům rozesílat sám. Po přijetí síťové zprávy od serveru se vykoná zpráva jako příkaz na změnu scény.

Na níže uvedené tabulce je uveden rozdíl mezi řešeními. Podstatný rozdíl spočívá v tom, že v původním řešení se například při zprávách o změně pohybu hráče řešila jen změna směru pohybu a stav pohybu jako takový. Vlastní posun pak jen cyklicky opakovalo herní jádro, dokud nepřišla zpráva o změně. Díky tomu nastával chaos v konkrétních pozicích – hodnotách X,Y jednotlivých hráčů. Nesoulad nastával díky měnící se latenci mezi klienty. Distribuce zpráv navíc probíhala paralelně s již vykonávaným/vykonaným pohybem hráče.

Obecně platí, že pokud přijatá zpráva má příznak *Přeposlat*, přeposílá se. Pokud je pří-

jemcem takové zprávy server, přeposílá ji všem klientům. Pokud je příjemce klient, posílá ji serveru.

V novém řešení se nejprve ke zprávě přidá příznak *Přeposlat*. Tím se z ní formálně stane požadavek. Požadavek se přeposílá na server. Ten musí tedy především odstranit příznak *Přeposlat*, jinak by zpráva zbytečně kolovala znovu sítí. Následně ji přeposílá všem klientům. Ti tuto zprávu již zpracovávají jako příkaz a provádí změnu scény.

Fronta zpráv se tedy nemusí složitě přepočítávat podle aktuální latence všech klientů. Druhým příjemným efektem je, že problémy způsobené vysokou latencí se projevují jen u klienta, který má vysokou latenci a neovlivňuje tak všechny ostatní. Příklad: Zpráva, kterou klient odešle směrem na server, dorazí s velkou latencí na server. Ten ji přeposílá ostatním klientům s malou latencí. Ti mají paradoxně scénu změněnou dříve, než klient, který o změnu scény žádal.

V tabulkách 4 a 5 jsou znázorněny rozdíly mezi architekturami klient-klient a klient-server na příkladu komunikace. V závorce jsou uváděny ilustrační časy v milisekundách. V tabulce 6 je znázorněna hlavní výhoda nového řešení – přesun řešení problému s latencí na toho klienta, který má hodnotu latence největší.

Posloupnost událostí
Klient 1
(0) zmáčknutí klávesy
(0) generování zprávy: pohyb hráče
(0) distribuce do lokální fronty zpráv a ostatním klientům
(0) zpracování lokální fronty; pohyb hráče
Klient 2 (hodnota latence vůči druhému klientovi je 10 ms)
(10) přijetí zprávy
(10) zpracování lokální fronty; pohyb hráče

Tabulka 4: Řešení sychronizace pomocí achritektury klient-klient

Posloupnost událostí
Klient 1 (hodnota latence vůči serveru je 1 ms)
(0) zmáčknutí klávesy
(0) generování zprávy: `Jdi(A,B)`
(0) zpracování herním klientem: převedení na požadavek a odeslání hernímu serveru
(2) přijetí příkazu; `Jdi(A,B)`
Server
(1) herní server převádí požadavek na příkaz a odesílá všem připojeným klientům
Klient 2 (hodnota latence vůči serveru je 10 ms)
(11) přijetí příkazu; `Jdi(A,B)`

Tabulka 5: Řešení sychronizace pomocí achritektury klient-server

Posloupnost událostí
Klient 1 (hodnota latence vůči serveru je cca 100 ms)
(0) zmáčknutí klávesy
(0) generování zprávy: `Jdi(A,B)`
(0) zpracování herním klientem: převedení na požadavek a odeslání hernímu serveru
(200) přijetí příkazu; `Jdi(A,B)`
Server
(100) herní server převádí požadavek na příkaz a odesílá všem připojeným klientům
Klient 2 (hodnota latence vůči serveru je cca 1 ms)
(101) přijetí příkazu; `Jdi(A,B)`

Tabulka 6: Ukázka paradoxu synchronizace

Při testování však byl zjištěn potenciální problém při distribuci zpráv s příkazem, tedy již na cestě od serveru ke klientům. Pokud se zpráva ztratí mohou nastat dva problémy.

Situace 1

První možností je, že se ztratí zpráva cestou od serveru ke klientovi, který vyslal požadavek. Situace probíhá následovně:

1. Klient 1 odeslal serveru požadavek: `Jdi(A1,B1)`
2. Zpráva na server nedorazila.
3. Klient 1 odesílá serveru požadavek: `Jdi(A1,B1)`
4. Zpráva na server dorazila,
5. Server odesílá klientovi 1 i klientovi 2 příkaz: `Jdi(A1,B1)`.
6. Oba klienti přijmou zprávu.
7. Oba klienti se posouvají o A1,B1

Situace 2

Druhou možností je, že se ztratí zpráva cestou od serveru ke klientovi, který pouze provádí synchronizaci s ostatními klienty. Pak se průběh situace změní na toto:

1. Klient 1 odeslal serveru požadavek: `Jdi(A1,B1)`
2. Zpráva na server dorazila.
3. Server odesílá klientovi 1 i klientovi 2 příkaz: `Jdi(A1,B1)`.
4. Klient 1 přijímá zprávu.
5. Klientovi 2 zpráva nikdy nedorazí.
6. Klient 1 se posouvá o A1,B1.
7. Klient 2 nedělá nic.

V prvním případě je pohyb pouze o jeden herní cyklus pozastaven. V druhém případě jsou však fatálnější dopady, protože hráč je na každém klientovi na různých pozicích.

Problém: Synchronizace pozic při výpadku zpráv

Řešení: Neposílat relativní změny pozic, ale absolutní hodnoty.

Místo posílání zpráv `Jdi(A,B)` se formát zpráv změnil na `Jdi(X,Y)`, kde X a Y jsou absolutní pozice hráče, takže zpráva říká, ne o kolik, ale kam se má hráč posunout. Pozitivem je tak to, že při výpadku zpráv nejsou předchozí zprávy o změně pohybu potřeba, protože je známo, kam přesně se hráč má přesunout. Příklad: Pokud používáme zprávy `Jdi(A,B)` a budeme chtít hráče desetkrát posunout, vznikne deset zpráv o posunu. Aby bylo vše v pořádku, tak musí všem klientům dorazit všech deset zpráv. Avšak stačí, aby jedna nedorazila, a hned bude hráč stát špatně. Díky novému řešení se může předchozích devět zpráv ztratit a přes to bude pozice u všech v pořádku.

Negativem výše uvedeného řešení je však ztráta informace o směru pohybu hráče. Předchozí řešení buď přímo určovaly směr (např. jdi doleva), nebo se to dalo spočítat z číselné pozice (posuň se o A,B, kde pokud například bude A kladné, hráč se pohybuje doprava,...). Nynější typ zprávy nám však nic o směru neříká, a proto byla zavedena do zprávy druhá dvojice X,Y. Nový typ zprávy má tedy tvar `Jdi(X1,Y1,X2,Y2)`. To efektivně vyřešilo problém s určením směru pohybu a nemusí se tak zvlášť posílat zpráva o změně natočení hráče.

Odečtením souřadnic X2-X1 a Y2-Y1 získáme hodnoty, ze kterých určíme směr pohybu stejným způsobem, jako zprávy typu `Jdi(A,B)`.

4.3 Porovnání s existujícími řešeními

Při studování a návrhu řešení synchronizace bylo využito též již hotových řešení. Výběr řešení proběhl na základě dostupnosti zdrojového kódu aplikací – vybrány byly aplikace veřejně dostupné pod svobodnou licencí GPL (anglicky GNU General Public Licence[1]). První vybraný program je budovatelská hra OpenTTD (anglicky Open Transport Tycoon Deluxe[2]. Druhý vybraný program je OpenArena[3], rychlá akční hra.

Obě hry umožňují se spustit bez uživatelského rozhraní pouze jako servery. Tomuto chování se říká dedikovaný server.

4.3.1 OpenTTD

Tato hra vznikla jako remake původní hry s názvem Transport Tycoon. Je naprogramována v jazyce C++ a až do dnešního dne probíhá její aktivní vývoj.

Analýzou zdrojového kódu bylo zjištěno, že používá jako transportní protokol při síťové hře oba majorivní protokoly, TCP a UDP. Protokol TCP používá na komunikaci řízení a vlastní zprávy pak posílá přes rychlý protokol UDP.

Vlastní synchronizace probíhá tak, že uživatel provádí akce, které generují události. Zpracování události se vykoná lokálně u klienta a souběžně ve stejnou chvíli se přeposílají jako příkaz na server. Ten jen distribuuje na ostatní klienty a udržuje je tak ve stejném stavu.

4.3.2 OpenArena

Ve světě velice populární hru Quake 3 Arena se rozhodl její původní autor po mnoho letech uvolnit. K původně komerčnímu programu tak najednou existovaly originální zdrojové kódy a je to tedy skvělý zdroj kde čerpat inspiraci.

Tento program je naprogramován v jazyce C a upravením a vylepšením originální hry tak vznikla nová hra OpenArena.

[1] http://www.gnu.org/licenses/licenses.html
[2] http://www.openttd.org/
[3] http://www.openarena.ws/

V základu je hra dělena na server a klient. Striktně je oddělen i zdrojový kód: klient, server a herní engine. Vlastní komunikace staví na UDP protokolu a podobně jako v předchozí hře, se posílají příkazy a k nim definované souřadnice o změně. Tyto zprávy server zpracovává a přeposílá klientům. Existuje zde mechanizmus, který trvale hlídá hodnotu latence mezi serverem a klienty a pokud je vyšší, než 999 milisekund, či je nedostupný, je označen jako potenciálně vyřazený hráč.

5 Testovací aplikace

Aplikace bude implementována v jazyce C++ a bude stavět na knihovně SDL, o síť se postará podpůrná knihovna SDL_Net. Jako primární cílová platforma je 64 bitový Debian s jádrem Linux.

5.1 SDL

Simple DirectMedia Layer (SDL) je multiplatformní multimediální knihovna poskytující nízkoúrovňový přístup na audio, klávesnici, joystick, 2D a 3D počítačovou grafiku přes OpenGL. Napsaná je v jazyce C, nicméně díky tzv. language bindings je možné knihovnu použít také v Javě, Delphi, Pythonu, Perlu a dalších. [2]

SDL samotné je velmi jednoduché; funguje jako tenký multiplatformní wrapper poskytující podporu pro 2D operace s pixely, zvuk, přístup k souborům, zpracování událostí, časování, vlákna atd. Je často použito jako doplněk OpenGL k nastavení grafického výstupu a poskytnutí vstupu z klávesnice a myši, což je za předmětem zájmu OpenGL.

Některé rozšiřující knihovny, které obohacují základní funkcionalitu SDL knihovny

- SDL_image – podpora pro více formátů obrázků,
- SDL_mixer – komplexní funkce pro práci s audiem, zejména pro mixování zvuku,
- SDL_net – podpora síťování,
- SDL_ttf – podpora vykreslování fontů TrueType,
- SDL_rtf – jednoduché renderování RTF.

Současná verze (1.2.14) pro Microsoft Windows využívá knihovnu DirectX verze 7, avšak linuxová verze má s akcelerací problémy a tak se využívá softwarového renderování, které je výrazně výpočetně a hlavně časově náročnější. Existují sice varianty, jak dodat akceleraci, avšak to znamená dodatečné, ne vždy banální zásahy do kódu programu.

Budocí verze 2.0, která je nyní již ve fázi testování, je v tomto směru mnohem lepší. Především proto, že veškeré práce ve 2D prostoru provádí transparentně pomocí již dobře podporované 3D akcelerace.

5.2 SDL_Net

SDL_Net[4] knihovna je též multiplatformní multimediální knihovna poskytující nízkoúrovňový přístup na síť. Zjednodušuje tak práci se sítí pomocí připravených funkcí a struktur. Současná verze bohužel podporuje pouze IP verze 4, což mírně degraduje do budoucna podporu celé aplikace.

5.3 Network stack

5.3.1 CNetwork

Základní třída pro práci se sítí. Ve svém kontruktoru mj. obsahuje parametr, kterým určujeme, zda se bude chovat jako server nebo klient. Podle toho inicializuje proměnné a vytvoří vlákna. V případě, že instanci vytvoříme jako server, vytváří a spouští instance tříd `CAcceptThread` (na vytvoření příchozích spojení od klientů) a `CSendThread` (na odeslání zpráv).

Pokud neučiníme předanými parametry jinak, ve výchozím stavu je nastavena, aby naslouchala na rozhraní `localhost` s portem 40666.

Dále obsahuje dvě důležité funkce:

- `send()` – odesílá zprávu,
- `get()` – přijímá zprávu.

Obě funkce volají příslušné funkce z instance třídy `CNetworkMonitor`, která je detailně popsána níže.

5.3.2 CNetworkThread, CReceiveThread, CSendThread, CAcceptThread

Tyto třídy obsluhují vlastní komunikaci po síti. Základem je třída `CNetworkThread`, která definuje především funkci funkci `run()`. Tato virtuální funkce slouží jako primární výkonná funkce, která se volá v jednotlivých instancích a potomcích této třídy.

Mezi třídy, které od třídy `CNetworkThread` dědí, patří třídy `CReceiveThread`, `CSendThread` a `CAcceptThread`.

[4] http://www.libsdl.org/projects/SDL_net/

- CReceiveThread – naslouchá pomocí funkce `SDLNet_TCP_Recv()` (blokující volání) na určeném socketu a při přijetí zprávy ji přidá do fronty zpráv a čeká na další.
- CSendThread – odesílá zprávu pomocí funkce `SDLNet_TCP_Send()` na všechny připojené klienty. Této funkcionality se využívá při synchronizaci, které je popsání podrobněji v další kapitole.
- CAcceptThread – naslouchá na zvoleném portu na příchozí nově vznikající spojení od klientů. Využívá funkci `SDLNet_TCP_Accept()`, která je však neblokující. Proto je po každém testu spojení vložena pauza 10 ms. V tomto čase se tedy žádný klient nepřipojí, avšak jeho požadavek není ztracen, a jeho požadavek bude přijat.

5.3.3 CNetworkMonitor

Network monitor slouží k práci s frontou zpráv. Je to tedy datová struktůra typu FIFO. Tato třída je generická a vyžaduje tedy parametr, který určuje, s jakým typem zpráv se pracuje.

Třída obsahuje 4 základní metody, které lze však ještě dělit podle směru na příchozí a odchozí: `insertIncoming` a `insertOutcoming` vkládající zprávy na vstup a výstup, resp. `getIncoming` a `getOutcoming`, čekající na zprávy na vstupu a výstupu.

- `insertOutcoming()` – uloží zprávu do fronty zpráv k odeslání
- `getOutcoming()` – odesílá zprávy ze fronty zpráv k odeslání
- `insertIncoming()` – uloží nově příchozí zprávy do fronty zpráv
- `getIncoming()` – vyzvedává zprávy z fronty příchozích zpráv. Volání je blokující, takže pokud ve frontě není žádná zpráva, čeká se do doby, dokud nebude alespoň jedna ve frontě.

5.3.4 CNetworkConnection

Základní třída udržující spojení mezi serverem a klienty. Obsahuje seznam čísel otevřeným socketů. Třída pracuje pouze s TCP sockety.

5.4 IMessages

Třída IMessages je rozhraní definující funkce, které klienti volají a jsou buď přímo (v případě lokální hry) nebo nepřímo (v případě klienta připojeného přes síť) následně zpracovány serverem.

Mezi funkce, které se klientům nabízí, patří například:

- `player(player, x1, y1, x2, y2, state, life)` – změna pozice, stavu (běžící, stojící, apod.) a života (živý, mrtvý, blokovaný)
- `bomb(player, x, y, index, state)` – operace s bombou na určité pozici, parametr state definuje, zda-li pokládáme bombu, či exploduje apod.
- `inventory(player, powerup, value)` – nastavení horních hranic inventáře u hráče
- `updateField(x, y, occupancy, powerup)` – úprava mapového podkladu
- `setDefPlPos(player, x, y)` – nastavení výchozí pozice hráče
- `changeMenu(menu)` – přepíná herních meny
- `startGame()` – odstartování hry
- `showResults()` – zobrazení výsledkové listiny
- `exitGame()` – ukončení hry
- `registerPlayer(player, node, control)` – registrování hráče u serveru do startovací listiny a jeho ovládání
- `unregisterPlayer(player)` – smazání hráče ze startovací listiny

5.5 IClient, CLocalClient, CNetworkClient

Třída IClient dědí od třídy `IMessages` a přidává důležitou metodu `Think()`, která je pravidelně volána z herní smyčky. Metoda je určena na zpracování zpráv. Vyzvedne zprávy z fronty zpráv, provede zpracování a případně plní frontu odchozími zprávami. Třídy `CLocalClient` a `CNetworkClient` dědí od rozhraní `IClient` a nepřidávají žádnou další důležitou funkcionalitu. Důležitá změna se děje až při zpracování zpráv. Lokální klient

zprávy tranformuje na přímé volání ekvivalentních funkcí ve hře. Síťový klient musí nejprve veškerá volání zabalit do zprávy, kterou odešle přes síť.

5.6 IServer, CLocalServer

U serveru je dědičnost stejná, jako u klienta: třída `CLocalServer` dědí od rozhraní `IServer`. Kromě metody s totožným názvem a funkcionalitou jako u klienta, `Think()`, definuje ještě dvojici metod `startNetwork()` a `stopNetwork()`. Tyto metody spouští či zastavují síťovou podporu herního serveru. Když je síť spuštěna, server naslouchá na předem určeném portu, přes který navazuje spojení a komunikuje s klienty.

6 Testování aplikace

V této kapitole je krátce popsáno vytvoření síťové hry a její ovládání.

6.1 Spuštění síťové hry

Otestování synchronizace mezi dvěma aplikacemi se provádí takto:

- Na obou počítačích se spustí aplikace.
- Z první aplikace se udělá server – zvolí se položka *Start network game*.
- Druhá aplikace bude tedy klient – zvolí se položka *Joint network game*. Zadá se IP adresu nebo hostname serveru. Pokud se spouští obě aplikace na jednom počítači, lze ponechat výchozí volbu. Pokračuje se kliknutím na tlačítko *Connect*.
- Na obou aplikacích zvolíme hráče.
- V serverové aplikaci se kliká na tlačítko *Next* a v následující obrazovce na *START!*.

Podmínkou spuštění hry jsou zvolení alespoň dva hráči. Není nutno, aby byly dva hráči v každé aplikaci.

6.2 Ovládání hry

Zde je uveden výčet kláves, které lze používat ve hře.

6.2.1 Ovládání hráčů

Hráč 1 (Player 1)

- Doleva: šipka doleva
- Doprava: šipka doprava
- Nahoru: šipka nahoru
- Dolů: šipka dolů
- Položení bomby: Mezerník

Hráč 2 (Player 2)

- Doleva: klávesa s
- Doprava: klávesa f
- Nahoru: klávesa e
- Dolů: klávesa d
- Položení bomby: klávesa a

6.2.2 Ovládání hry

- Přerušení běžící hry: klávesa F11
- Skok do základní nabídky v menu nebo z výsledkové listiny: klávesa ESC

7 Závěr

Cílem této diplomové práce byl návrh síťového aplikačního protokolu sloužícího k synchronizaci herního stavu. Pro dobrý požitek ze hry bylo zapotřebí sofistikovaně redukovat problémy vzniklé latencí mezi jednotlivými hráči na minimum. Toho bylo dosaženo především zvolením architektury klient-server, která do značné míry zredukovala počet zpráv nutných k synchronizaci stavu jako takového a přidala nové možnosti k redukci negativních projevů latence.

Pro přenos zpráv jsem zvolil přenosový protokol TCP především proto, že synchronizace samotná představovala dostatečně složitý problém. Původním záměrem bylo přenášet synchronizační zprávy se stoprocentní úspěšností. Testováním a postupnou evolucí způsobu předávání zpráv a jejich obsahu, například upuštění od použití zpráv synchronizujících čas u jednotlivých klientů, se však podařilo dosáhnout stavu, kdy je možné uvažovat o změně na protokol UDP, na který se doposud spoléhalo. Změna by to byla buď kompletní, nebo částečná. To znamená využít kombinaci obou zmíněných protokolů, jak je to řešeno u zkoumaných existujících řešení. U zpráv, kde není vyžadována nízká latence, by bylo možné použít protokol TCP a u zpráv, kde může dojít ke ztrátě, avšak je vyžadována nízká hodnota latence, použít protokol UDP.

Kombinací protokolů TCP a UDP se jeví jako ideální pokračování vylepšování aplikačního protokolu. Urychlením přes protokol UDP by mohlo odpadnout poslední velké břemeno, které ještě může mít značný vliv na hodnotu latence a tím by se mohl zlepšit celkový dojem ze hry s dalšími hráči. Avšak již v tomto momentě je mnou navržený protokol schopen dobře synchronizovat herní stav.

Ačkoliv jsem navrhoval protokol nejvyšší vrstvy architektury TCP/IP, musel jsem se hned ze začátku věnovat vrstvám nižším. Testováním na několika odlišných datových linkách různé kvality, včetně mobilního velmi pomalého GPRS, jsem tak nabyl zkušeností, jak málo stačí, aby se komunikace změnila v nepoužitelnou. Mnoho nápadů se tak často rázem stalo bezcennými.

Hra byla původně vyvíjena jen na jednom počítači a pro jednoho hráče. Po spuštění na jiném, výkonnějším počítači hra vykazovala vyšší rychlost pohybu hráče. První úpravy programu tedy započaly ještě před vlastní implementací síťového aplikačního protokolu, především pak umožnění ovládání hry i z jiného místa přes počítačovou síť. Před vlastní im-

plementací aplikačního protokolu do hry to prakticky znamenalo, že příprava hry na síťovou komunikaci zabrala více času, než jsem původně předpokládal.

Literatura

[1] Neznámý autor. Latence. [online] 28. 6. 2013, [Citace: 11. 5. 2013] http://cs.wikipedia.org/wiki/Latence.

[2] Neznámý autor. SDL. [online] 13. 8. 2013, [Citace: 24. 5. 2013] http://cs.wikipedia.org/wiki/Simple_DirectMedia_Layer.

[3] Neznámý autor. TCP/IP. [online] 26. 5. 2013, [Citace: 12. 5. 2013] http://cs.wikipedia.org/wiki/TCP/IP.

[4] Věra Gošová. Internetové protokoly. [online] 3. 11. 2011, [Citace: 24. 5. 2013] http://wiki.rvp.cz/Knihovna/1.Pedagogický_lexikon/I/Internetové_protokoly.

[5] Luboš Klaška. Základní kvalitativní parametry sítě (1) - latency (zpoždění). [online] 6. 12. 2006, [Citace: 11. 5. 2013] http://www.svetsiti.cz/clanek.asp?cid=Zakladni-kvalitativni-parametry-site-1-latency-zpozdeni-6122006.

[6] Koonsolo. deWiTTERS Game Loop. [online] 13. 7. 2009, [Citace: 24. 5. 2013] http://www.koonsolo.com/news/dewitters-gameloop/.

[7] Jiří Kysela. Parametry sítí a nástroje pro jejich diagnostiku. [online] 24. 1. 2012, [Citace: 11. 5. 2013] http://www.internetprovsechny.cz/parametry-site-a-nastroje-pro-jejich-diagnostiku/.

[8] Autor neznámý. Protokol UDP 1.část. [online] 3. 2. 2003, [Citace: 11. 5. 2013] http://www.builder.cz/rubriky/c/c–/protokol-udp-1-cast-156226cz.

[9] Drew Prindle. How to Use BitTorrent. [online] 13. 8. 2013, [Citace: 10. 5. 2013] http://www.digitaltrends.com/computing/how-to-use-bittorrent/.

[10] Jaroslav Snášel. Jak vzniká zpoždění při přenosu dat v mobilních sítích. [online] 12. 5. 2005, [Citace: 11. 5. 2013] http://www.mobilmania.cz/clanky/jak-vznika-zpozdeni-pri-prenosu-dat-v-mobilnich-sitich/sc-3-a-1110027/default.aspx.

Příloha A – Obsah CD

Na CD, které je přiloženo v tištěné verzi diplomové práce, se nachází

- elektronická verze této diplomové práce,
- kompletní zdrojové kódy aplikace (adresář src),
- návod na přeložení aplikace (soubor README),
- zkompilovaný binární soubor aplikace, formát ELF 64-bit (soubor openatomic) a
- UML diagram síťové části hry (soubor network.png).