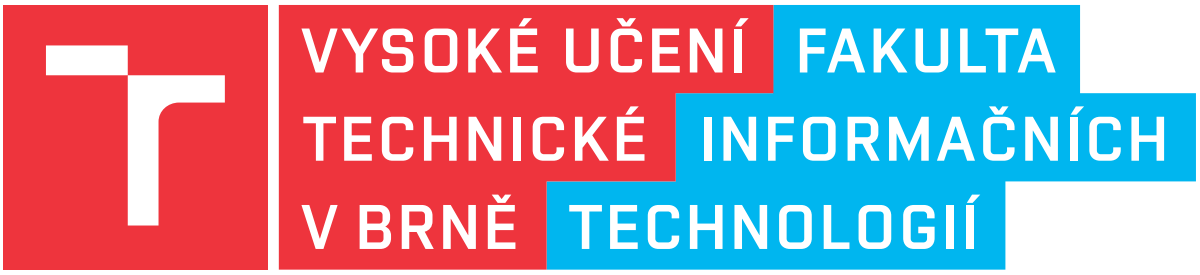

Přenos dat, počítačové sítě a protokoly (PDS)

Hromadný projekt - Hybridní chatovací P2P síť

v Brně
28. dubna 2019

Václav Martinka
xmarti76

Obsah

1 Zadání

Cílem projektu je vytvořit P2P chatovací síť. Tedy naprogramovat chatovacího peera, registrační uzel a jejich RPC[1] ovládání. Dále je třeba výsledek řádně otestovat a to včetně kompatibility s jinými implementacemi.

2 Struktura P2P sítě a komunikační protokol

Síť je tvořena dvěma typy uzlů. Tzv. *peery*, což jsou samotní chatovací klienti ovládaní uživateli a dále *nody*, které zastupují funkci adresářů a poskytují peerům potřebné informace. Nody jsou dále schopny mezi sebou navázat sousedství a vzájemně si vyměnit údaje o peerech i svých sousedech, čímž vzniká full mesh síť[2], tudíž případný výpadek nodu má přímý vliv pouze na peery k němu připojené.

Zadání dále obsahuje přesnou specifikaci komunikačního protokolu, což zaručuje (nebo by alespoň mělo) vzájemnou kompatibilitu. Základem komunikace jsou zprávy strukturované jako *JSON*, které jsou před odesláním kódované pomocí *bencode*. Všechny zprávy obsahují povinný unikátní číselný identifikátor *txid* a definici typu zprávy *type*. Veškerá komunikace probíhá pomocí protokolu `UDP`, který je v aplikační vrstvě rozšířen o určité zabezpečení spolehlivosti přenosu, viz podkapitola 2.3

2.1 Peer

Úkolem peera je odesílat zprávy jiným uživatelům a zároveň přijímat zprávy od ostatních. Aby bylo možné jej v síti identifikovat, je mu uživatelem přidělena přezdívka. Mimo to je nutné, aby se zaregistroval k některému z nodů, čímž získá přístup k adresám jiných uživatelů a sám tak ostatní informoval, na které adrese je dostupný. V rámci své komunikace využívá následující typ zpráv:

`hello` slouží k registraci, prodloužení registrace a odhlášení peera u noda

`getlist` si vyžádá seznam uživatelů a jejich IPv4 adres a portů

`message` zašle zprávu, přičemž zpráva `message` je poslána přímo příjemci bez zapojení noda (v opačném případě by se nejednalo o P2P síť, ale klasický klient-server přístup).

2.2 Node

Node poskytuje peerům služby adresáře, to znamená, že přijímá registrace peerů, jejichž IPv4 adresy a příslušné porty zaznamenává ve své databázi. Kromě toho může navazovat sousedství s jinými nody. Se svými sousedy si posléze vyměňuje nejen svoji databázi peerů, ale taktéž databázi svých sousedů. Tato informace je využita k navázání dalších sousedství a vytvoření *full mesh sítě*. Komunikace je tvořena následujícími zprávami:

`list` reprezentuje obsah databáze registrovaných peerů, je odpovědí na žádost `getlist`

`update` slouží k navázaní a udržení sousedství, přičemž obsahuje nejen databázi registrovaných peerů, ale také seznam všech svých sousedů a u nich registrovaných peerů

`disconnect` ukončí sousedství

2.3 Potvrzování zpráv

Protože všechny zprávy jsou zasílány pomocí `UDP` protokolu, který je ze své podstaty negarantovaný, byl protokol doplněn o potvrzovaní zpráv speciálním typem zprávy *ack*, jejíž *txid* není jedinečné,

[1] Remote procedure call
[2] Všechny uzly jsou navzájem propojeny

ale odpovídá *txid* potvrzované zprávy. Pokud do 2 sekund[3] nedorazí *ack*, je zpráva považována za nedoručenou. Zprávy typu *hello*, *update*, *error* a samozřejmě také *ack* nejsou potvrzovány.

Protokol obsahuje ještě jeden speciální typ zprávy, a tou je zpráva *error*. Ta značí libovolnou chybu, její `txid` taktéž není unikátní, nýbrž odpovídá `txid` chybné zprávy. Kromě toho obsahuje ještě slovní popis chyby.

2.4 Možná rozšíření protokolu

Protokol je velmi jednoduchý (obsahuje pouhých 8 typů zpráv), přesto je dostatečně robustní a funkční. Během implementace jsem ovšem narazil na několik drobností, které by mohli být řešeny jinak:

- Jelikož *hello* zpráva není potvrzována, nemůže peer zjistit, zda byla registrace úspěšná, popř. že je node nedostupný. Řešením by mohl být potvrzování *hello* zpráv.
- Před odesláním zprávy je nutné si vždy vyžádat seznam VŠECH registrovaných uživatelů, to může být v případě rozsáhlejší sítě poměrně velká zpráva, která zbytečně zatěžuje noda, který ji musí pokaždé vygenerovat, peera, který ji musí dékódovat a konečně i samotnou síť, která ji musí přenést. Částečným řešením může být cachování výsledků. Ovšem ideální by byla přítomnost zprávy typu *getuser*, pomocí které by si peer vyžádal adresu konkrétního uživatele.
- U zprávy *update* je situace obdobná. Tato zpráva totiž obsahuje nejen vlastní databázi, ale také databázi všech sousedů. Ovšem zadání jasně zakazuje tyto databáze zpracovat a místo toho je nutné si nejdříve s jednotlivými uzly vytvořit sousedství. Opět to generuje zbytečně velké zprávy, proto si myslím, že by bylo lepší zasílat seznam peerů registrovaných pouze k danému nodu a dále seznam jeho sousedů (ovšem už bez peerů registrovaných k nim).
- Protokol by mohl také obsahovat zprávu typu *void*, která by nedělal nic, pouze by vyžádala vygenerování *ack*. To by mohlo sloužit k ověření dostupnosti dané adresy.
- Zprávy *error* by mohli kromě slovního popisu chyby obsahovat i její identifikátor pro jednodušší strojové zpracování.

3 Implementace

Projekt byl implementován pomocí jazyka `C++`, standard z roku 2017. Zadání nutně vede na vícevláknové zpracování, to je obstaráno pomocí direktiv *OpenMP*. Základem programu jsou tzv. *sekce*, což jsou bloky kódu prováděné paralelně. Každá sekce má na starosti jinou část programu, jsou jimi:

- Příjem a zpracování zpráv
- Pravidelné odesílání *hello* popř. *update* zpráv a mazání starých údajů v databázích
- Obstaraní vstupů včetně RPC

V rámci sekce je u náročnějších úloh (např. zpracování zprávy) vytvořen *task*, což je pod-úloha, taktéž vykonávaná paralelně.

Mimo direktiv *OpemMP* a standardních `C++` knihoven byl dále použit hlavičkový soubor `bencode.hpp` od autora *Jim Porter* šířený pod licencí *BSD*[4].

[3] Hodnota ze zadání

[4] https://github.com/jimporter/bencode.hpp

3.1 Struktura kódu

Program je tvořen několika objekty. Základem je objekt `Communicator`, který je tvořen ze dvou objektů `Sender` a `Receiver`. Ty, jak naznačuje jejich název, slouží k samotnému odesílání a přijímání dat. Díky tomu lze velmi snadno přejít na libovolný jiný komunikační protokol, bez nutnosti většího zásahu. `UDP` komunikace byla zprovozněna pomocí návodu ze stránky http://www.builder.cz/rubriky/c/c--/protokol-udp-1-cast-156226cz a http://www.builder.cz/rubriky/c/c--/protokol-udp-2-cast-156227cz. Kromě toho tento objekt zaštiťuje odesílání, příjem a zpracování obecných typů zpráv jako jsou *ack* a *error*.

Neméně důležité jsou objekty `Peer` a `Node`, které jsou potomky objektu `Communicator` a doplňují komunikaci o další zprávy. Kromě toho jsou dále využity např. objekty `Options` pro zpracování a uložení argumentů a `Destination` popisující jednu adresu (kombinace IPv4 adresy a portu) včetně kontrol validity.

Všechny zdrojové kódy jsou dokumentovány pomocí dokumentačních komentářů, což zaručuje vytvoření validní doxygen dokumentace.

3.2 RPC

RPC je obstaráno pomocí pojmenované roury. Každý peer i node si vytvoří vlastní rouru umístěnou v adresáři `/tmp`, přičemž její název obsahuje také ID instance programu, tudíž nedochází ke kolizím. Program `pds18-rpc` tak po zpracování argumentů otevře danou rouru, odešle do ní data a ukončí se. Peer či node neustále naslouchají a rovnou zpracují případná příchozí data. Pokud akce vede k vypsání zpráv, jsou vypisovány na `stdout` peera (či noda).

Kromě toho je programy možné ovládat také pomocí standardního vstupu. Příkazy odpovídají RPC voláním ze zadání. Vždy musí začínat znakem `\` a jsou ukončeny zakončením řádku. Seznam příkazů je možné vypsat pomocí `\help`. Pro zadaní příkazů lze využít i jejich jednopísmenné zkratky povětšinou vycházející z prvního písmene (např. `\p` je interpretováno stejně jako `\peers` a slouží k vypsání všech peerů registrovaných na daném nodovi, tedy v podstatě se jedná o odeslání *getlist* zprávy a vizualizace odpovědi). Pokud příkaz přijímá další parametry (např. příkaz *message* pro odeslání zprávy), jsou uvedeny za příkazem odděleny jednou mezerou. Tedy poslání zprávy od Pepy Honzovi bude vypadat nějak takto: `"\m Pepa Honza Ahoj, jak se vede?"`.

Pro vlastní potřebu jsem RPC rozšířil o možnost danou instanci peera nebo noda ukončit. Slouží k tomu příkaz `quit`.

4 Testování

V rané fázi byl peer a node testovány vůči sobě. Ve finální fázi jsem pokročil k testování s ostatními spolužáky. Hlavní test jsem provedl vzájemně s Lucií Pelantovou (xpelan04) a Jiřím Matějkou (xmatej52), kteří projekt implementovali v `pythonu`. Během testování jsme se zaměřili především na nestandardní situace (např. velmi dlouhá či úplně prázdná zpráva) a objevili především drobnější chybky. Testování probíhalo na merlinovi, přičemž během testování se na naše nody připojili boti a skrze ně i další nody. Tím se krásně ověřilo vytváření full mesh sítě.

Mimo to jsem zkoušel spojení s dalšími nody běžícími na merlinovy (vetšinou `python`, ale i `C/C++`) a nenarazil na chyby.

5 Překlad a spuštění

Program lze přeložit pomocí přiloženého `Makefile`, která obsahuje následující příkazy:

`all` popř. pouze `make` přeloží všechny programy.

`pds18-peer` přeloží pouze peer.

`pds18-node` přeloží pouze node.

`pds18-rpc` přeloží pouze rpc.

`doxygen` vygeneruje *doxygen* dokumentaci.

`pack` vytvoří archiv se zdrojovými kódy.

`clean` smaže soubory vzniklé při překladu.

Chování peera lze ovlivnit pomocí následujících parametrů:

`--help (-h)` Zobrazí nápovědu a ukončí program

`--id` Nastaví identifikátor instance (pro potřeby RPC ovládání), výchozí hodnota je 0.

`--username` Nastaví přezdívku uživatele, výchozí hodnota je *user*.

`--chat-ipv4` Nastaví `IPv4` adresu, na které peer očekává odpovědi, výchozí je `127.0.0.1`.

`--chat-port` Nastaví port, na kterém peer komunikuje, výchozí je 6677.

`--reg-ipv4` Nastaví `IPv4` adresu, na noda, výchozí je `127.0.0.1`.

`--reg-port` Nastaví port noda, výchozí je 6677.

`--verbous (-v)` Povolí vypisování informací o průběhu.

Node má velmi podobné parametry:

`--help (-h)` Zobrazí nápovědu a ukončí program

`--id` Nastaví identifikátor instance (pro potřeby RPC ovládání), výchozí hodnota je 0.

`--reg-ipv4` Nastaví `IPv4` adresu, na které node očekává komunikaci, výchozí je `127.0.0.1`.

`--reg-port` Nastaví port, na kterém node komunikuje, výchozí je 6677.

`--verbous (-v)` Povolí vypisování informací o průběhu.

RPC se používá následovně:

`--help (-h)` Zobrazí nápovědu a ukončí program

`--peer (-p)` Specifikuje, že příkaz bude předán peeru

`--node (-n)` Specifikuje, že příkaz bude předán nodu

`--id (-i)` určuje instanci programu, výchozí hodnota je 0

`--command (-c) <command params>` definuje konkrétní příkaz a jeho případné parametry

Jednotlivé příkazy odpovídají zadání, pouze jsou doplněny o příkaz *quit* a jejich jednopísmenné varianty. Pokud příkaz něco vypisuje, je to vypsáno přímo na `stdout` dané instance programu.

Peer podporuje následující příkazy:

`message (m)` Odešle zprávu, je nutné doplnit o
`--from <from_username> --to <to_username> --message <message>`

`getlist (g)` Zašle *getlist*

`peers (p)` Odešle *getlist* a zobrazí formátovanou odpověď

`reconnect (r)` Odpojí se od současného noda a připojí se na nový, třeba doplnit o
`--reg-ipv4 <node_ipv4> --reg-port <node_port>`

`quit (q)` Ukončí peer

Node rozumí následujícím příkazům:

`database (d)` Zobrazí aktuální databázi registrovaných peerů

`neighbors (n)` Zobrazí databázi sousedních uzlů

`connect (c)` Pošle *update* zprávu novému uzlu, třeba doplnit o
`--reg-ipv4 <node_ipv4> --reg-port <node_port>`

`disconnect (x)` Zruší sousedství se všemi sousedy

`sync (s)` Pošle *update* zprávu všem svým sousedům

`quit (q)` Ukončí node

6 Závěr

Výsledkem projektu je trojice aplikací, pomocí které je možné vytvořit full mesh P2P chatovací síť. Tato síť se umí automaticky rozrůstat (pomocí navazování sousedství) a je odolná vůči výpadkům. Díky dodržení komunikačního protokolu je zajištěna vzájemná kompatibilita s implementacemi ostatních spolužáků, což bylo zároveň i ověřeno. Byť je síť provozována pomocí protokolu `UDP`, tak díky potvrzování zpráv je zaručeno jejich doručení.