Hybridná chatovacia P2P sieť

Přenos dat, počítačové sítě a protokoly (PDS)

Vysoké učení technické v Brně

Bc. Michal Ormoš

28. apríla 2019

Obsah

1 Zadanie

Cieľom tohto projektu bolo vytvoriť hybridnú klient-klient [2] chatovaciu aplikáciu. Táto aplikácia mala pozostávať z chatovacieho klienta (rovnako ďalej označovaného ako peer), registračného uzlu (rovnako ďalej označovaného ako node) a RPC modulu, pre testovacie účely. Úlohou registračného uzlu je vytvárať priestor na ktorý sa môžu chatovací klienti pripájať. Týmto spôsobom si uzol zberá informácie o dostupných klientoch a na požiadavok klienta im túto databázu klientov môže poskytnúť. Vďaka tejto informácií samotný klient zistí mená a adresy klientov vo svojom okolí a môže sa na nich priamo bez pomoci registračného uzlu pripájať a odosielať im správy. Ako je už možné už vidieť, celý zmysel klient-klient chatovacej aplikácie je vynechať z procesu samotného posielania správ medzi dvoma klientami server a tým zabezpečiť (aj ked táto aplikácia nezabezpečuje šifrovanie správ) komunikáciu medzi dvoma osobami, bez toho aby ju spracovala tretia strana.

2 Implementácia

Ako implementačný jazyk bol zvolený jazyk Python z dôvodu jednoduchej práce s formátom JSON, ktorý si bude tento projekt vo veľkom množstve vyžadovať. Napriek snahe implementovať full-mesh sieť, sa mi vytvorenie spolupráce uzlov (full-mesh sieť) nepodarilo realizovať, preto registračné uzly v tomto projekte medzi sebou nevedia komunikovať, ani si vymienať informácie.

2.1 Implementácia peera

Úlohu peera v projekte sú tri základne veci.

- Pripojenie sa na registračný uzol
- Pripojenie sa na RPC modul
- Načúvanie na porte pre správy

Klient je implementovaný pomocou jedného zdrojového súboru, ktorý obsahuje triedu Client. Tá si pri inicializácii vytvorí UDP socket [1] na príjem chatovacích správ podľa vstupných argumentov a daný socket si rezervuje, aby mu ho operačný systém nezmenil. Na tomto sockete čaká na správy `MESSAGE` od iných klientov a v prípade ich prijatia im rozošle správu `ACK`, na adresu, ktorá správu poslala.

Ďalej si klient podľa vstupných argumentov si vytvorí vlákno na ktorom začne periodicky každých 10 sekúnd posielať správy `HELLO` na svoj registračný uzol, ktorý mu bol zadaný. Kedže sa pri správach `HELLO` nekontroluje `ACK`, klient chlucho vysiela správy pre registračný uzol. Klient si pamätá v akom čase poslal poslednú správu `HELLO` a pomocou periodického zisťovania času vo formáte `timestamp` si zistí kedy mu ubehlo 10 sekúnd od poslanie poslednej správy a teda musí poslať novú a aktualizovať svoj čas. V spojení so

svojim registračným uzlom vie klient na dotaz od RPC požiadať uzol o zoznam ostatných klientov v tejto sieti a následne potvrdiť, že ho prijal. Vďaka čomu môže posielať správy iným klientom v sieti bez učasti uzlu na v rámci tejto komunikácie, teda poslať správu napriamo.

V druhom vlákne si klient vytvorí další UDP socket pre príjem správ od RPC. A to tak, že na svoju chatovaciu `IPv4` adresu priradí port nasledovne, zoberie svoje `id`, ktoré mu bolo na vstupe priradené a pripojí k nemu deviätky až jeho dĺžka portu je o veľkosti štyri. Teda ak ma ako `id` čislo 7, je ho RPC port bude 7999. Na tomto RPC načúva na klasickom UDP socketu, na správy od svojej RPC autority, ktorá ho môže požiadať aby získal zoznam ostatných klientov od svojho uzlu, kontaktoval iného klienta, alebo sa odpojil od aktuálneho uzlu a pripojil na uzol iný.

2.2 Implementácia node

Implementácia registračného uzlu je obdobná implementácii klienta. Uzol si pri spustení vytvorí UDP socket na ktorom, podľa vstupných argumentov čaká na správy od klientov. Správy môžu prísť trojakého druhu.

- HELLO - Klient sa hlási svojemu uzlu, kde mu posiela informácie o sebe samom, ktoré si registračný uzol ukladá do svojej vnútornej databazy, ktorú následne poskytuje ak je o to požiadaný. Uzol si kontroluje ci už daný klient v jeho databáze existuje, ak áno tak mu len aktualizuje časovú známku, kedy sa naposledy hlásil. Pretože ak sa klient nehlási pomocou správy `HELLO` dlhšie ako 30 sekúnd, je z registračného uzlu vyhodený.
- GETLIST - Klient môže požiadat registračný uzol o datbázu klientov, ktoré su k nemu pripojené. Uzol vyhovie zaslaním správy `LIST` a počkaním si na potvrdenie, že ju klient prijal `ACK`.
- ACK - Klient oznamuje, že úspešne prijal poslednú tranzakciu.

V osobitnom vlákne si vytvorí RPC socket, na ktorom načúva na pr9kazy od RPC, a RPC port si určí obdobne ako to spravil aj klient. Ten mu môže prikázať aby zobrazil aktuálnu databázu svojich uzlov.

2.3 Implementácia RPC

RPC je implementovaný ako další klient, ktorého úlohou je posielať špeciálne správy klientovi či uzlu. Implementácia bola vykonaná pomocou rovnákeho UDP socketu, avšak nastal problém ako by RPC modul zistil, kde ma kontaktovať uzol alebo klienta. Pri každom spustení či už klienta, alebo uzlu, sa daný klient/uzol zapíše do špecialného súboru `network.dat`, kde si vytvorí záznam sám o sebe, a to `type` (peer/node), `id`, `ip_adresa` a špeciálny `RPC_port`. Vďaka tomuto ma RPC prehľad kde a ako bežia jednotlivý klieti či

uzly. Ak sa klient či uzol ukonči (užívateľom, či systémovo), svoj záznam z daného súboru odstráni. Ak je tento odstránený záznam posledný, celý súbor sa zmaže. Výsledky RPC príkazov sa vypisujú na príslušný node/peer a nie na samotný výstup RPC.

3 Komunikácia

Komunikácia prebieha pomocou zasielania JSON správ, ktoré su `benkodované`. Na ich benkódovanie bola využitá knižnica yabencode [1]. Implementácia modulu na benkódovanie správ mi prišla zbytočná. Keďže som narazil na pár knižníc, ktoré to vedia dokonalo a ich obsah je voľný k šíreniu. Preto táto knižnica bola pridaná k programu ako modulu, kedže testovacia virtuálka ju neobsahovala. Rovnako sa jedná o jediný externý kód, ktorý bol v programe prevzatý.
Okrem správ HELLO a ERROR, sa každá správa komunikácie potvrdzuje pomocou ACK.

4 Testovanie projektu

Kedže sa mi full-mesh sieť uzlov nepodarilo za žiadných okolností stabilne implementovať, testovanie aplikácie som si nechal na samotný koniec, čo bolo chyba a všetci znamý kolegovia už mali po štyroch ľudom s ktorými testovali, preto prikladám moje trošku skromnejšie testovanie s dvoma ľudmi. Aktuálne implementované RPC príkazy:

- `-peer -command message -from <username1> -to <username2> -message <zpráva>`
- `-peer -command getlist`
- `-peer -command peers`
- `-peer -command reconnect -reg-ipv4 <IP> -reg-port <port>`
- `-node -command database`

4.1 Testovanie s Bc. Vojtěch Matějíček (xmatej51)

Jazyk implementácie: Python

Testované prvky

- Testovanie prebiehalo na verejnej adrese server Merlin a lokálnom PC
- RPC volanie pre kombinácie peer/node
- Odpájanie peerov z uzlov

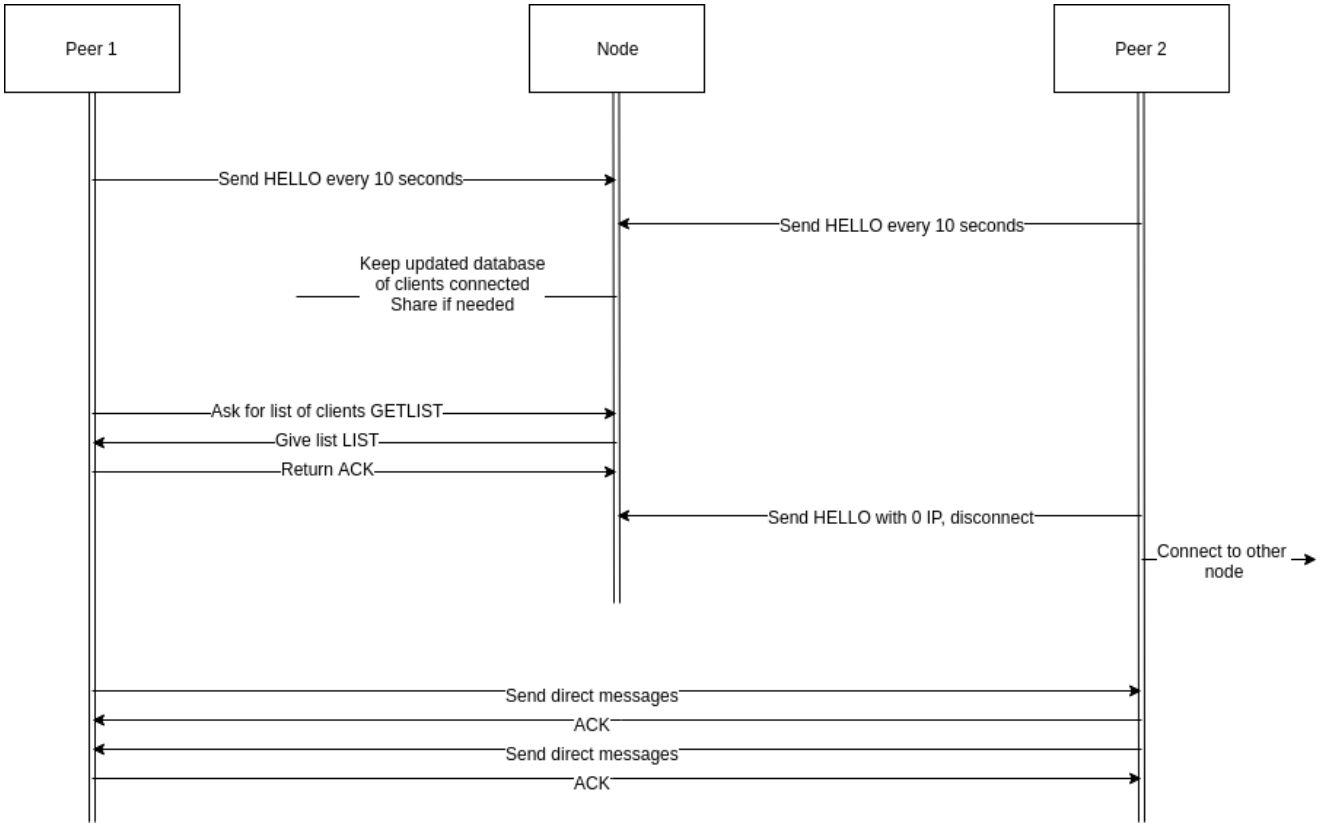
[1] https://pypi.org/project/yabencode/

4.2 Testovanie s Bc. Ľuboš Mjachky (xmjach00)

- Jazyk implementácie: Python
- Testované prvky
  - Pripojenie/odpojenie peera z vzájomných ulozv
  - Posielanie správy medzi peerami navzájom cez rôzne uzly
- Detaily implementácie:
  - Použité jedno vlákno ja jeden request (správu). N vlákien pre n požadaviek.
  - Testovanie prebiehalo na verejnej adrese servera Merlin
  - Implementovaný bol celý projekt zo strany xmjach00
  - RPC načúva v samostnatnom vlákne a odbavuje požiadavky, výsledok/chybovú hlášku vypíše na výstup RPC.
  - RPC komunikácia prebieha na základe AF_UNIx socketu, ktorý je spoľahlivý z hľadiska doručovania
  - na stopovanie času uzlov či klientov, je použitý jednoduchý timer

1 Sekvenčný diagram

Obr. 1: Sekvenčný diagram programu



Literatúra

[1] Python: Wiki,
`https://wiki.python.org/moin/UdpCommunication`

[2] KOEGEL BUFORD, John F, Hong Heather YU a Eng Keong LUA. P2P networking and applications. Burlington: Morgan Kaufmann, 2009, xxi, 415 s. : il. ISBN 978-0-12-374214-8.