VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

HYBRIDNÁ CHATOVACIA P2P SIEŤ

HYBRID P2P NETWORK FOR CHATTING

PŘENOS DAT, POČÍTAČOVÉ SÍTĚ A PROTOKOLY
DATA COMMUNICATIONS, COMPUTER NETWORKS AND PROTOCOLS

AUTOR **DÁVID BOLVANSKÝ**
AUTHOR

BRNO 2019

Obsah

Kapitola 1

Úvod

Úlohou projektu bolo navrhnúť, implementovať a otestovať hybridnú chatovaciu P2P sieť obsahujúcu chatovacích peerov a registračné uzly. V nasledujúcich kapitolách sú opísané dôležité časti projektu.

V kapitole 2 prebieha úvodom do problematiky, ozrejmením relevantných pojmov. Kapitola 3 sa zaoberá komunikačným protokolom. V kapitole 4 je opísaná jeho samotná implementácia. Kapitola 5 obsahuje informácie o priebehu ladenia a testovania. Kapitola 6 poskytuje prehľad nad používaním programu. Posledná kapitola zhrňuje získané vedomosti a skúsenosti z projektu.

Kapitola 2

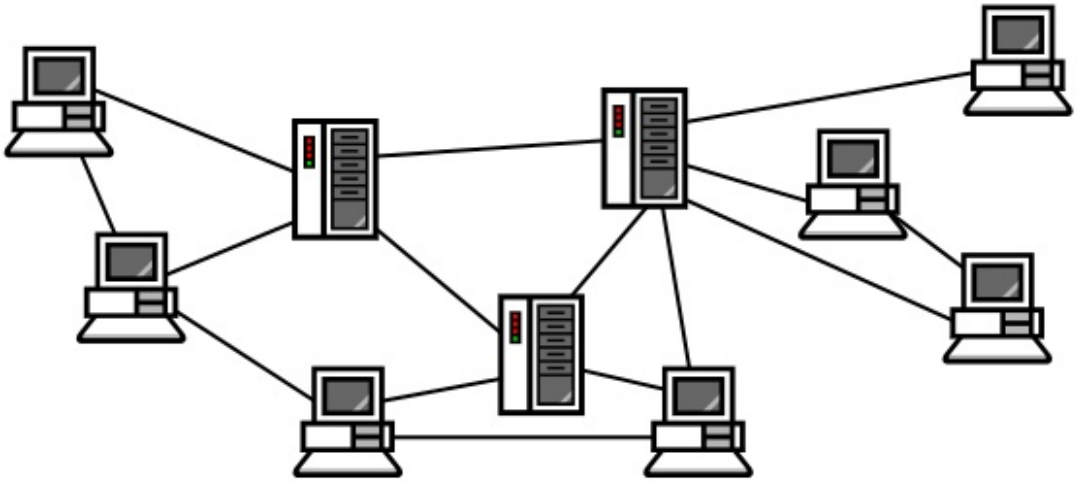
Súhrn pojmov

Táto kapitola obsahuje prehľad a ozrejmenie základných pojmov súvisiacich s problematikou tohto projektu. Informácie ohľadom P2P sietí boli čerpané hlavne z prednášky v predmete PDS [4] a súvisiaceho RFC dokumentu.

P2P sieť

P2P sieť je možné definovať ako súbor nezávislých uzlov (označovaných ako „peers“), ktoré sú prepojené a ich zdroje sú k dispozícii iným uzlom v tejto sieti. P2P siete sa nezaobídu bez funkčnej IP infraštruktúry a taktiež je nutné v týchto sieťach riešiť problematiku adresovanie, smerovania, zabezpečenia a pod. Základom každej P2P siete je logická sieť, ktorá je postavená nad existujúcou sieťovou infraštruktúrou. Logická sieť definuje spôsob prepojenia uzlov, smerovanie, vyhľadávanie informácie a pod. Medzi význačné vlastnosti P2P sietí patria: samorganizovateľnosť, autonómne chovanie, spoľahlivosť a životnosť uzlu. Všeobecne rozlišujeme dva typy P2P sietí: pravé, kde odobranie ľubovolného uzlu zo siete nemá vplyv na schopnosť poskytovať službu a hybridné, ktoré pre svoju činnosť využívajú centrálny uzol pre poskytovanie časti ponúkaných sieťových služieb (ako napr. autentizácia, indexovanie či inicializácia uzla). Na obrázku 2.1 [1] je zobrazená schéma hybridnej P2P siete. Viac informácií ohľadom P2P sietí je možné nájsť v dokumente RFC 5694 [2].

[1] https://www.slideshare.net/jamezsa/lecture-network-technologies-peertopeer-networks



Obr. 2.1: Schéma hybridnej P2P siete

UDP

UDP je základným protokolom transportnej vrstvy v architektúre TCP/IP. Je tiež označovaný ako tzv. nespojovaný („connectionless“) protokol. Protokol prináša malé zaťaženie siete, no jeho nevýhodou je to, že sa neriešia prípady straty, poškodenia paketov, prípadne zmena poradia ich doručenia, a tiež môže dochádzať k duplicite dát. Protokol UDP je podrobne opísaný v RFC 768 [5].

Na obrázku 2.2 je znázornený model RPC [2]. Klient odosiela požiadavku na server. Požiadavka sa na serveri vykoná a možná odpoveď sa posiela späť žiadateľovi, tj. klientovi. Server následne čaká na ďalších klientov a ich požiadavky.

Bencode

Bencode je kódovanie používané peer-to-peer systémom na zdieľanie súborov *BitTorrent* na ukladanie a prenos dát. Podporuje štyri rôzne typy hodnôt: reťazce, celé čísla, zoznamy a slovníky. Najčastejšie sa používa v torrent súboroch, kde sú tieto metadátové súbory kódované slovníky práve pomocou Bencode. Viac podrobností ohľadom bencode a samotný algoritmus spôsob kódovania je možné nájsť na (neoficiálnej) Wiki stránke [3]. Dokument obsahujúci špecifikáciu Bencode je BEP 3 [3].

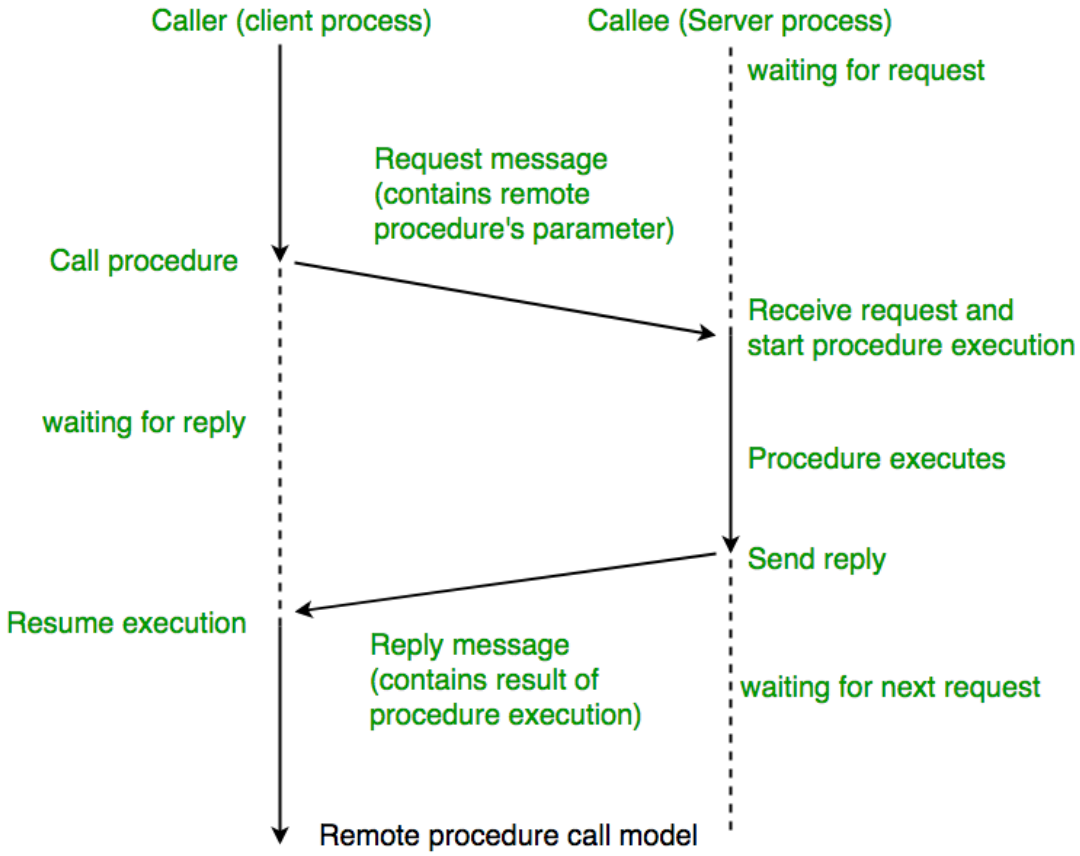
JSON

JSON („JavaScript Object Notation“) je spôsob zápisu dát (dátový formát) nezávislý na počítačovej platforme, určený pre prenos dát, ktoré môžu byť organizovaná v poliach alebo zoskupená v objektoch. Vstupom je ľubovoľná dátová štruktúra (číslo, reťazec, boolean, objekt alebo z nich zložené poľa), výstupom je vždy reťazec. Špecifikáciu JSON formátu je možné nájst v dokumente RFC 8259 [1].

[2] https://www.geeksforgeeks.org/operating-system-remote-procedure-call-rpc/
[3] https://wiki.theory.org/index.php/BitTorrentSpecification#Bencoding

RPC

„Remote Procedure Call“ (RPC) je technika vzdialeného volania procedúr. Umožňuje programu vykonať kód na inom mieste, než je umiestnený volajúci program.



Obr. 2.2: Model Remote Procedure Call

Kapitola 3

Komunikačný protokol

Peeri a registračné uzly používajú jednoduchý komunikačný protokol pozostávajúci z nižšie uvedených správ. Všetky správy medzi dvoma peerami, dvoma registračnými uzlami, či medzi peerom a registračným uzlom sú prenášané cez protokol UDP. Všetky správy majú JSON syntax, kde povinným atribútom je *type*, ktorý špecifikuje typ správy. Pred prenosom cez UDP je obsah správy zakódovaný pomocou Bencode. Keďže protokol UDP negarantuje doručenie, tak sa na potvrdenie používa správa typu `ack`, ktorá v sebe nesie odkaz na jedinečný transakčný identifikátor správy, ktorú potvrdzuje. Ak dôjde pri spracovaní ľubovoľnej správy k chybe, príjemca správy posiela odosielateľovi správu typu `error`, ktorá okrem identifikátora transakcie obsahuje aj slovný popis problému, ku ktorému došlo. Správy typu `hello`, `update` a `error` nie je potrebné pomocou správy `ack` potvrdzovať. Na potvrdenie `ack` sa čaká maximálne 2 sekundy, potom najprv zresetovat stav spracovania súvisiaceho s nepotvrdenú správou a následne ohlásiť chybu na štandardný chybový výstup (program sa neukončí, iba upozorní používateľa o chybe). Všeobecne je možné správy mimo očakávaný stav protokolu zahadzovať.

Protokol podporuje nasledujúce správy:

```
HELLO := {"type":"hello", "txid":<ushort>, "username":"<string>",
  "ipv4":"<dotted_decimal_IP>", "port": <ushort>}

GETLIST := {"type":"getlist", "txid":<ushort>}

LIST := {"type":"list", "txid":<ushort>, "peers": {<PEER_RECORD*>}}

PEER_RECORD := {"<ushort>":{"username":"<string>",
  "ipv4":"<dotted_decimal_IP>", "port": <ushort>}}

MESSAGE := {"type":"message", "txid":<ushort>, "from":"<string>",
"to":"<string>", "message":"<string>"}

UPDATE := {"type":"update", "txid":<ushort>, "db": {<DB_RECORD*>}}

DB_RECORD := {"<dotted_decimal_IP>,<ushort_port>":{<PEER_RECORD*>}}

DISCONNECT := {"type":"disconnect", "txid":<ushort>}
```

Kapitola 4

Popis implementácie

Kapitola popisuje návrhové rozhodnutia, použité mechanizmy a zaujímavejšie časti implementácie. Programy sú napísané v Pythone 3, vývoj prebiehal nad Pythonom 3.6 na Ubuntu 18.04 LTS.

4.1 Chatovací peer

Chatovací peer je implementovaný v súbore `pds18-peer.py`. Peer po spracovaní argumentov vytvorí nové vlákno, ktoré obsluhuje príkazy z RPC. V tomto vlákne spustí TCP server na voľnom porte a informáciu o IPv4 adrese a porte tohto servera uloží do súboru pre potreby RPC (viď podkapitola venujúca sa implementácii RPC). Medzi implementačne zaujímave príkazy patria `peers`, `message` a `reconnect`.

V prípade príkazu `peers` je nutné poslať správu `getlist` registračnému uzlu a čakať na odpoveď, ktorou je správa `list`. RPC vlákno spustí slučku, v ktorej po každej desatine sekundy kontroluje, či sa príznak prijatia správy `list` nenastavil. Následne pošle `ack` registračnému uzlu na túto prijatú správu. Prijatú správu `list` pošle späť na RPC klienta, ktorý ju vypíše na svoj výstup. Každá prijatá správa `list` zároveň aktualizuje internú „cache“ mapovaní peerov, ktorú si každý peer udržiava.

Táto cache sa využíva pri správe `message`, kde sa najskôr hľadajú informácie o príjemcovi správy najskôr v tejto cache. V prípade, že sa záznam o príjemcovi nenájde, pošle sa správa `getlist` pre získanie najaktuálnejšieho mapovania peerov. Ak sa ani v tomto zozname nenájde záznam o príjemcovi, chybové hlásenie o tomto probléme sa vypíše na štandardný chybový výstup a proces odosielania správy končí. Ak záznam existuje, je správa odoslaná príjemcovi na IPv4 adresu a port z nájdeného záznamu. V prípade, že odosielateľ špecifikovaný v RPC príkaze nesúhlasí s menom peera, je vypísané chybové hlásenie a ďalej sa proces posielania správy pracuje s menom odosielateľa uvedeného v parametroch RPC príkazu `message`.

Po prijatí príkazu `reconnect` z RPC peer odošle správu `hello` svojmu registračnému uzlu s nulovými položkami na mieste IPv4 adresy a portu, čím oznamuje registračnému uzlu, že sa odpája od neho. Následne posiela klasickú správu `hello` novému registračnému uzlu, ktorého IPv4 adresa a port sú definované v parametroch RPC príkazu.

Po spustení RPC vlákna sa v „hlavnom“ vlákne taktiež spúšťa periodický časovač, ktorý každých 10 sekúnd odosiela správu `hello` svojmu registračnému uzlu. Ďalej prichádza na rad časť spracovávania správ, ktoré môže peer prijať. Ak dôjde k chybe pri benkódovaní, prípadne správa neobsahuje nutné položky, je vypísané chybové hlásenie s popisom problému

a odoslaná správa typu `error` odosielateľovi poškodenej správy, ktorá ako sekvenčné číslo použije číslo z prijatej správy. V prípade, že v prijatej správe chýbalo sekvenčné číslo, alebo toto číslo bolo mimo rozsah typu *unsigned short*, je v správe typu `error` použité číslo 0 ako sekvenčné číslo. Po odoslaní správy od peera sa v „ACK“ slovníku pre dané sekvenčné číslo nastaví hodnota False.

Po prijatí potvrdenia sa hodnota pre sekvenčné číslo z prijatého potvrdenia v tomto slovníku zmení na True. Po uplynutí časovača nastaveného na dve sekundy sa znova skontroluje záznam v slovníku - ak je stále False, potvrdenie nedorazilo a peer vypíše chybové hlásenie a pokračuje v činnosti ďalej. Peer po prijatí správy `list` si aktualizuje svoju „cache“ mapovaní peerov a zmenou príznaku upozorní potenciálne čakajúce RPC vlákno na prijatie správy typu `list`. Po prijatí správ `message` a `list` peer odosiela potvrdenie o doručení registračnému uzlu.

Po ukončení peera pomocou Ctrl-C (SIGINT) sa peer odpojí od svojho registračného uzla tak, že zašle správu `hello` s nulovými položkami na mieste IPv4 adresy a portu. Taktiež vypína aktuálne bežiace časovače a uzatvára sockety.

Rôzne premenné je nutné používať vo vláknach aj časovačoch. Z tohto dôvodu sú v implementácii na globálnej úrovni a pre jednoduchšiu orientáciu medzi premennými vo funkciách ich názov obsahuje prefix „g_“.

4.2 Registračný uzol

V implementácii registračného uzla (`pds18-node.py`) sa používajú rovnaké návrhové rozhodnutia a princípy ako u peera, a preto je pre čitateľa odporúčané sa najskôr oboznámiť s predchádzajúcou podkapitolou. Špecificky však registračný uzol pridáva periodické časovače, ktoré posielajú správy `message` susedom registračného uzla. Taktiež pridáva aj časovače na detekciu odpojenia svojho peera či suseda zo siete.

Po detegovaní odpojenia si registračný uzol upravuje svoje interné zoznamy a slovník (databáza). Registračný uzol po prijatí správy `getlist` overuje, čí žiadajúci peer patrí medzi peerov daného uzla - ak nie, je odoslané chybové hlásenie peerovi, ktorý ho na svoj výstup vypíše. Pri úspechu kontroly je odoslaná správa typu `list`, ktorá obsahuje záznamy o mapovaní peerov. Registračný uzol periodicky dostáva správy `hello` od svojich peerov. V prípade, že sa zistí, že daný peer ešte nie je v záznamoch registračného uzla, je do nich pridaný. Ďalej je možné, že sa zmenilo mapovanie a po detekcii nezhody medzi starými a novým údajmi sa uložia tie nové. Taktiež sa mohol peer odpojiť, vtedy poslal správu `hello` s nulovými položkami na mieste IPv4 adresy a portu a registračný uzol si ho následne odstráni zo záznamov. Vo všetkých troch prípadoch, keďže došlo k zmene v záznamoch, je vykonaná synchonizácia záznamov so susednými uzlami, tj. pošle sa správa `update` susedným uzlom. Vynútiť synchronizáciu záznamov so susedmi je možné aj pomocou RPC príkazu `sync`.

Najzaujímavejšia správa je správa typu `update`. Po jej obdržaní si uzol aktualizuje záznamy o peeroch tohto uzla, ak tento uzol patrí medzi susedov registračného uzla. Záznamy od susedných uzlov sa nazývajú ako autoritatívne záznamy. ďalej, ak v správe `update` nájde záznamy o uzloch, ktoré nepozná, naviaže s nimi susedstvo pomocou správy `connect` a takto sa vytvára plne prepojená „full-mesh“ sieť. Susedstvo je možné naviazať aj pomocou RPC príkazu `connect`.

Pri ukončení registračného uzla pomocou Ctrl-C (SIGINT) sa posiela správa `disconnect` všetkým svojim susedom, vypínajú sa časovače a zatvárajú sa sockety. Správu `disconnect`

je možné vyvolať aj RPC príkazom `disconnect` - v tom prípade sa kontroluje, či potvrdenie na `disconnect` dorazilo, pri ukončení klasickým spôsobom sa prijatie potvrdenia nekontroluje.

4.3 RPC

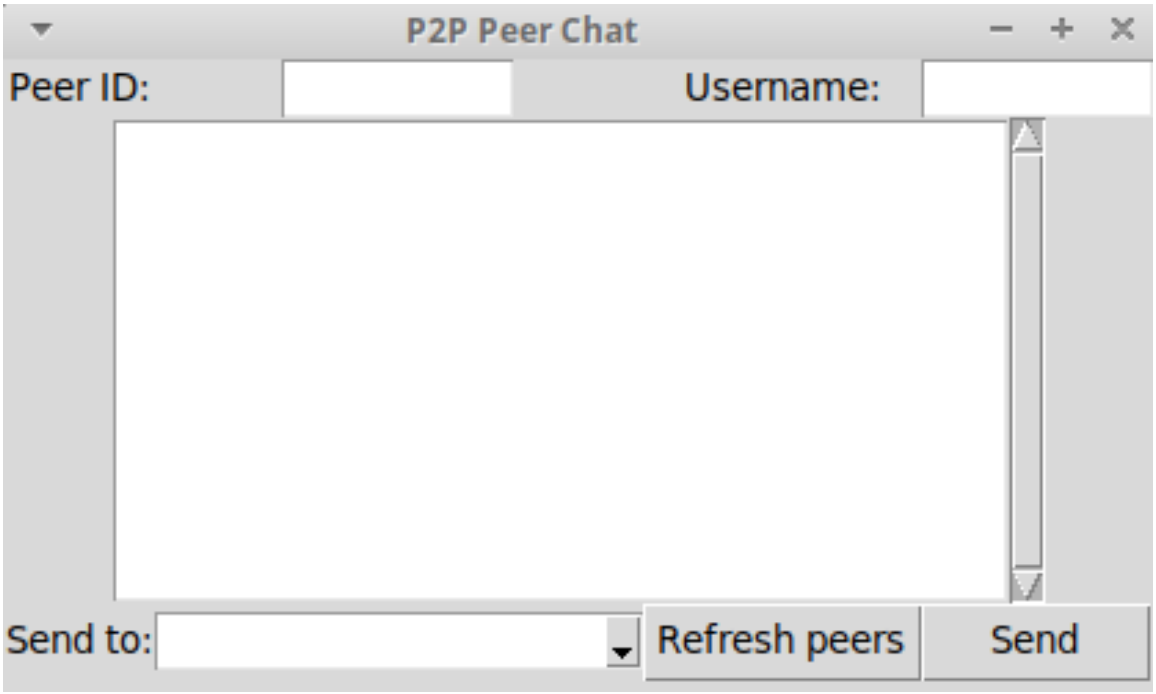
Naimplementované riešenie RPC v súboroch `pds18-rpc.py` a `rpc.py` používa TCP spojenie medzi RPC klientom a peerom / registračným uzlom na prenos príkazov a odpovedí. Pri spustení peera / registračného uzlu dochádza k vytvoreniu TCP serveru na voľnom porte, ktorý mu je pridelený operačným systémom, kde následne prijíma príkazy, ktoré vykonáva. Po spustení TCP serveru sa IPv4 adresa a port na pripojenie k serveru uložia do súboru `peer_%id.rpc`, resp. `node_%id.rpc`. Po zadaní RPC príkazu si RPC klient prečíta tento súbor a pripojí sa na získanú IPv4 adresu a port, kde pošle daný príkaz. Príkaz je zakončený špeciálnou značkou „$$“, aby bolo možné zložiť príkaz na strane peera či uzla v prípade, že prvé volanie `recv` u TCP by nevrátilo celý príkaz, ale by bolo nutné vykonať viac týchto volaní. Značka „$$“ je vybraná cielene, keďže sa nemôže vyskytovať v texte správy - shell nahradzuje „$$“ za PID procesu. Pri ukončení behu peera alebo registračného uzla je tento súbor vymazaný.

4.4 Bencode

Projekt obsahuje vlastnú implementáciu kódovania/dekódovania pomocou Bencode v súbore `bencode.py`. Implementácia bola vytvorená na základe dokumentov, ktoré obsahujúceho špecifikáciu tohto kódovania. Podporuje základné dátové typy potrebné pre účely navrhnutého protokolu zo zadania projektu. Implementácia kóduje aj Unicode znaky, no výsledný výstup nemusí byť kompatibilný s inými implementáciami (napr. v jazyku C) - nezhoda vzniká pri otázke, či má byť dĺžka u Unicode znaku 1 alebo 2. Špecifikácia bencode hovorí o kódovaní znakov z množiny ASCII, čo v tejto implementácii funguje správne a implementácie v rôznych jazykoch sú kompatibilné. Nehovorí však nič o znakoch mimo ASCII, a preto nepovažujem prípad nekompatibility pri Unicode znakoch za problém, tj. čo nie je špecifikované je nedefinované a je na implementácii, ako sa k tomu postaví (a pre úplnosť, populárna knižnica bencode-python3 dostupná cez pip3 implementuje rovnaké chovanie pri kódovaní týchto znakov ako moja implementácia).

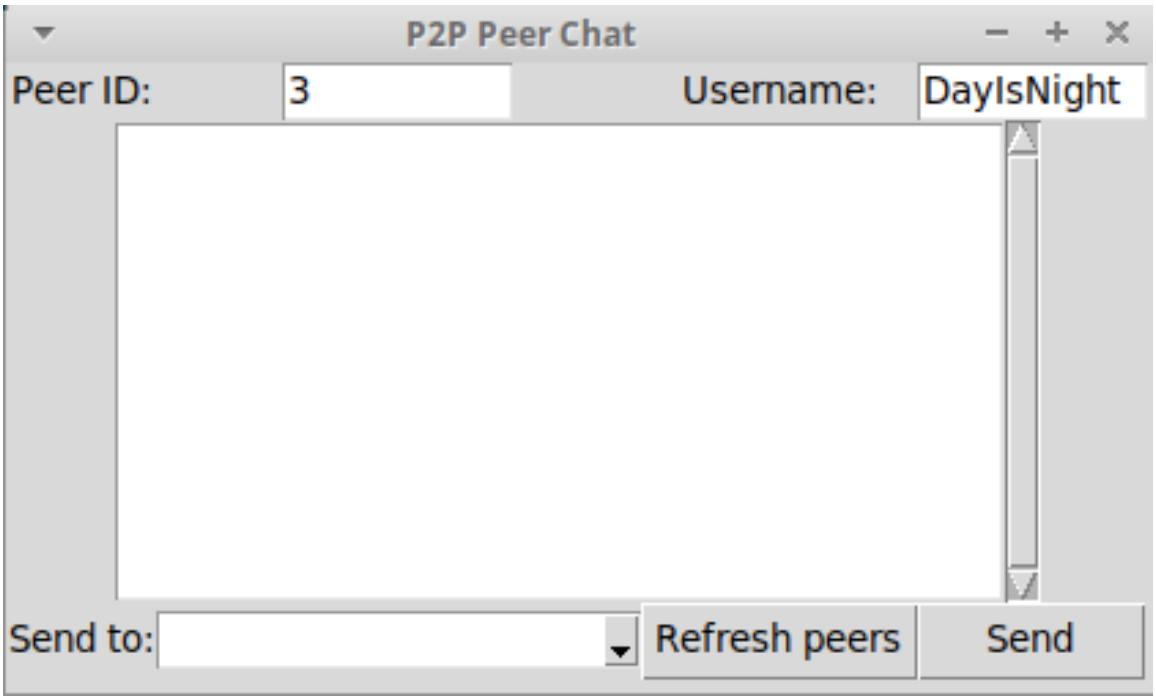
4.5 Grafické užívateľské rozhranie

Ako rozšírenie projektu som implementoval aj jednoduché GUI na posielanie správ v Python Tkinker - `pds18-gui.py`, ktorá využíva mechanizmy implementované v RPC kliente. Po spustení pomocou príkazu `python3 pds18-gui.py` sa zobrazí aplikácia.



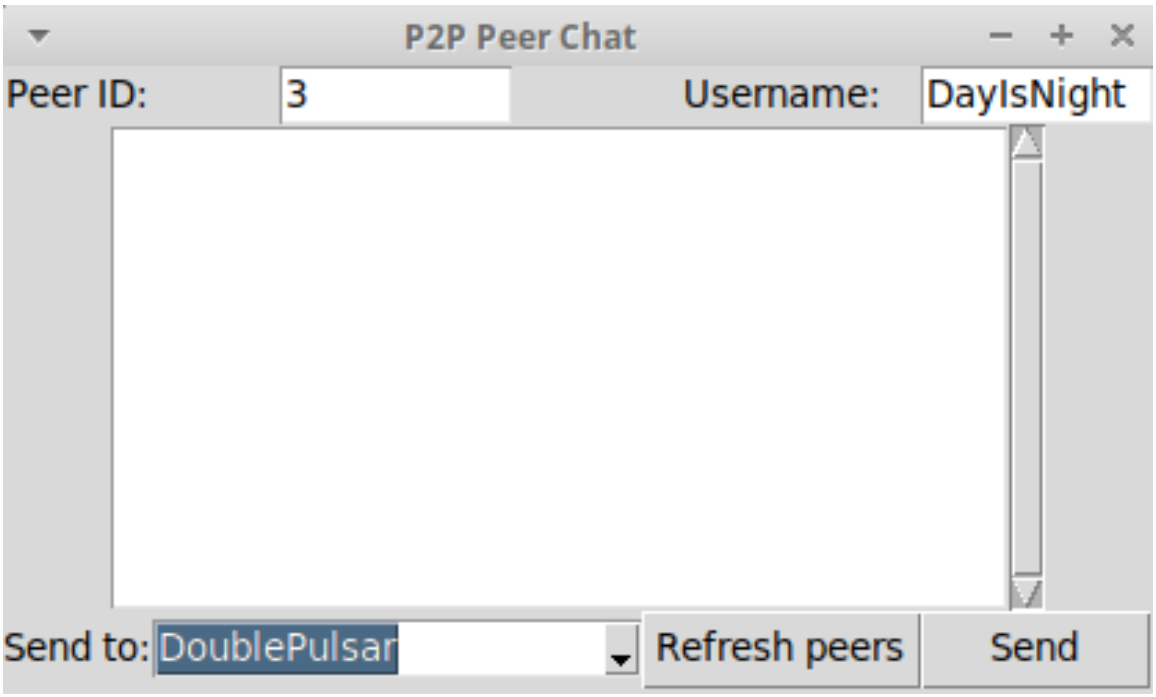
Obr. 4.1: P2P chatovacia aplikácia

V hornej časti obrazovky je potrebné zadať číselné ID instancie peera, ktorý je už spustený, a jeho používateľské meno (toto meno sa použije v poli odosielateľ u odoslanej správy).



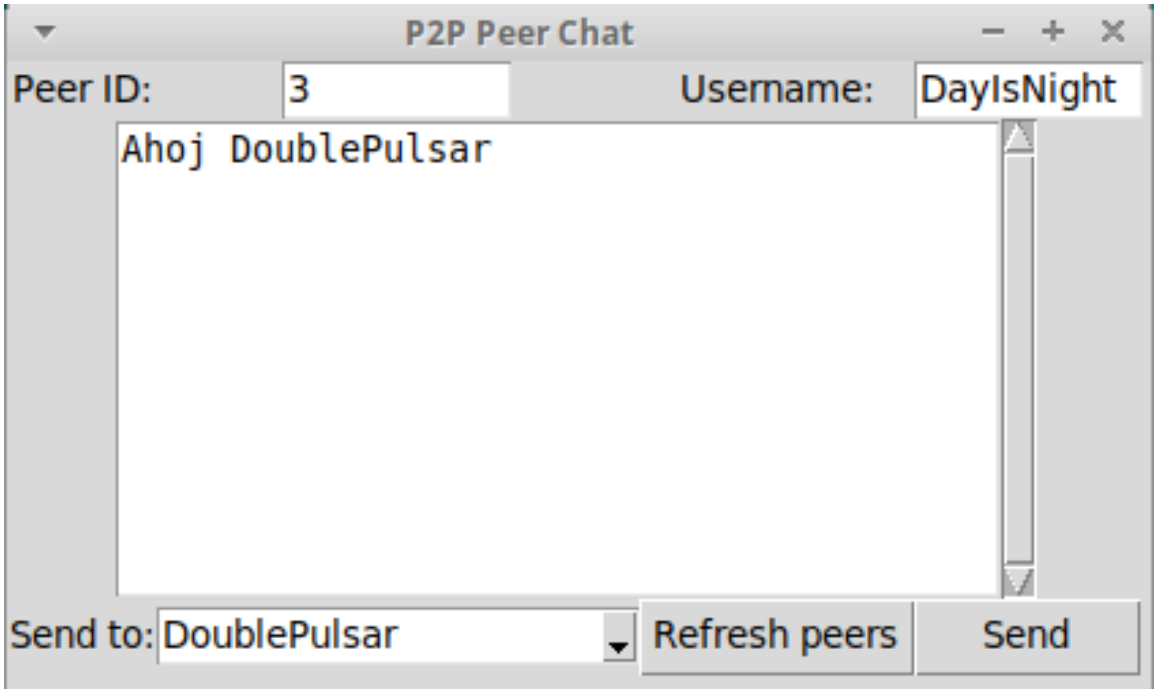
Obr. 4.2: Údaje o odosielateľovi

Po prvom spustení je zoznam peerov, ktorým je možné poslať správu, prázdny, a preto je potrebné tento zoznam aktualizovať pomocou tlačidla „Refresh peers“.



Obr. 4.3: P2P chatovacia aplikácia

Následne si používateľ môže zoznamu peerov vybrať, komu chce správu adresovať a do textové poľa zadať obsah správy.



Obr. 4.4: Obsah správy

Ak nenastal žiadny problém pri posielaní správy a správa dorazila na cieľového peera, na strane príjemcu je možné vidieť prijatú stranu.

```
xbolva00@xbolva00-G551JW:~/WIPProjects/PDS$ python3 pds18-peer.py --id 2 --usern
ame DoublePulsar --chat-ipv4 147.229.206.39 --chat-port 9002 --reg-ipv4 147.229.
206.39 --reg-port 6000
Message from DayIsNight: Ahoj DoublePulsar
```

Obr. 4.5: Prijatá správa

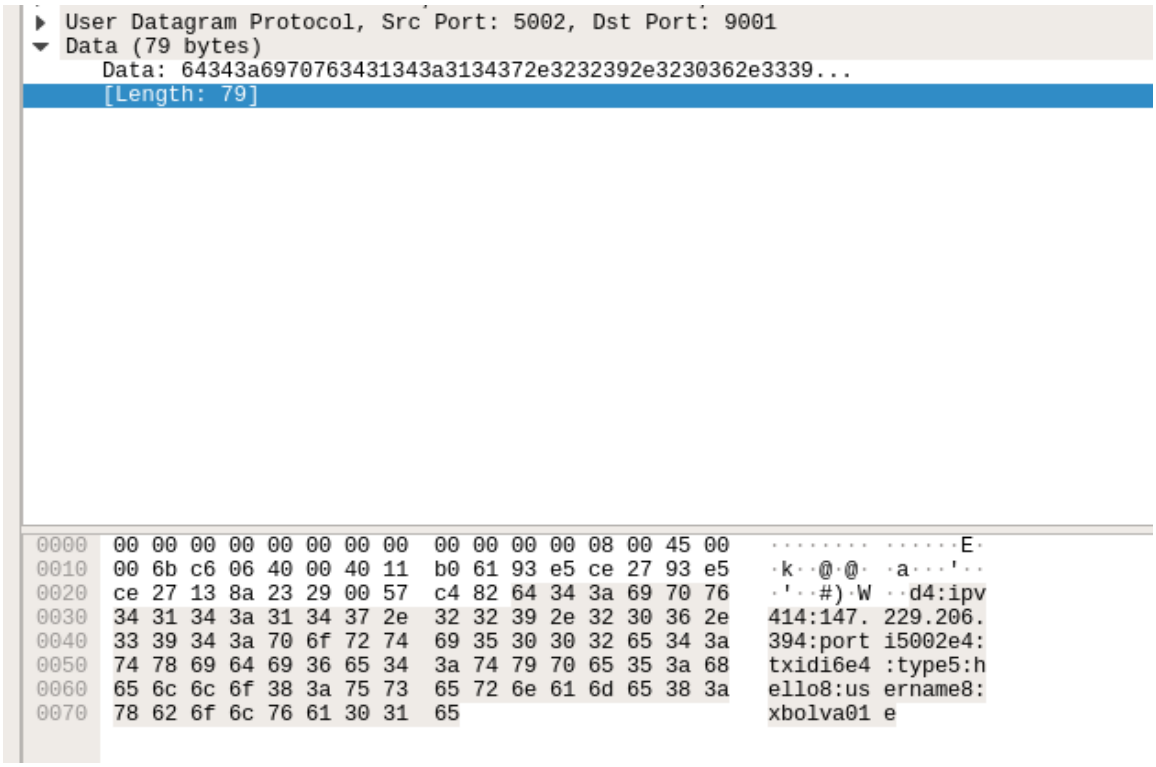
Kapitola 5

Ladenie a testovanie

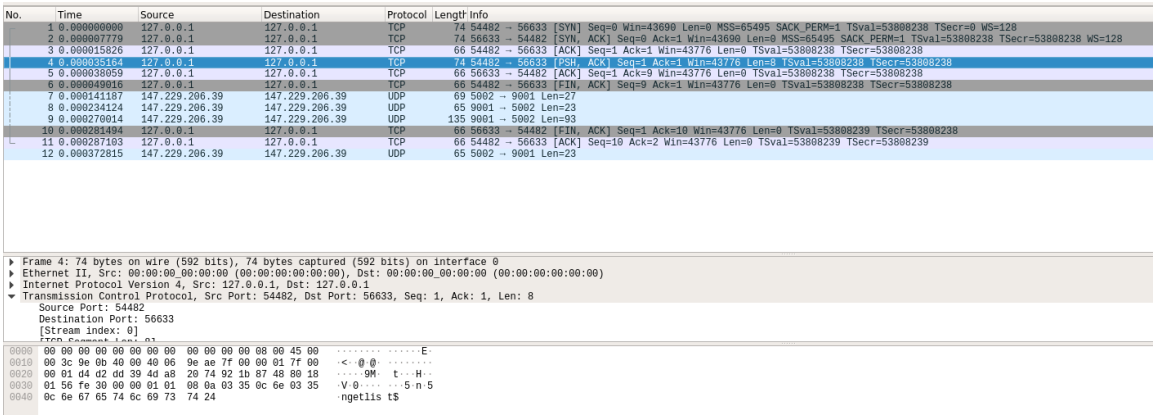
Priebežné ladenie a testovanie implementácie prebiehalo na mojom notebooku so systémom s Ubuntu 18.04 LTS a Pythonom 3.6. Takmer finálna implementácia bola následné otestovaná spolu s implementáciami ďalších spolužiakov, kde sa opravili nájdené chyby. Na záver prebehlo záverečné testovanie implementácie aj na referenčnom virtuálnom stroji pre tento projekt.

5.1 Individuálne ladenie implementácie

Vytvorené programy som priebežne ladil prevažne pomocou ladiacich výpisov. Prebiehajúcu UDP/TCP komunikáciu som sledoval pomocou nástroja Wireshark.



Obr. 5.1: Príklad sledovanie komunikácie vo Wiresharku - HELLO správa (peer → registračný uzol)

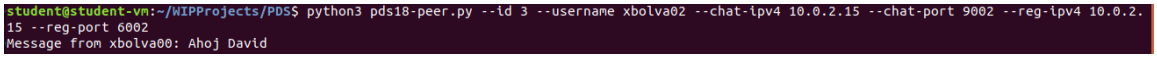


Obr. 5.2: Príklad sledovania komunikácie Wiresharku - RPC príkaz GETLIST. V záznamoch z Wiresharku je možné vidieť moment poslania príkazu GETLIST peerovi cez RPC, tj. naviazalo sa TCP spojenie medzi RPC a peerom. V záznamoch nasleduje UDP komunikácia, kde peer poslal správu GETLIST svojmu registračnému uzlu, peerovi prišiel ACK na GETLIST a aj LIST od registračného uzla. Peer následne poslal ACK registračnému uzlu na prijatý LIST.

5.2 Testovanie na virtuálnom stroji

Dokončenú implementáciu som otestoval na referenčnom virtuálnom stroji s Ubuntu OS pre tento projekt. Okrem rôznych testovaných scenárov zmienim asi tej najhlavnejší - poslanie správy v plne prepojenej sieti. Spustil som tri registračné uzly a troch peerov. Každý peer sa napojil na jeden zo spustených uzlov. Následne som prepojil prvý uzol s druhým, a druhý s tretím. Prvý registračný uzol sa následne vďaka správam `update` dozvedel o treťom uzle a naviazal s ním susedstvo. Ďalej som si vypísal databázy a susedov pre kontrolu stavu siete. Následne som otestoval najpodstatnejšiu službu hybridnej chatovacej P2P siete - posielanie správ. Po vytvorení plne prepojenej siete som teda poslal úspešne správu z prvého peera na tretieho. Nasledujúce obrázky 5.6 a 5.4 dokumentujú priebeh tohto testovacieho scenáru v termináli na virtuálnom stroji.

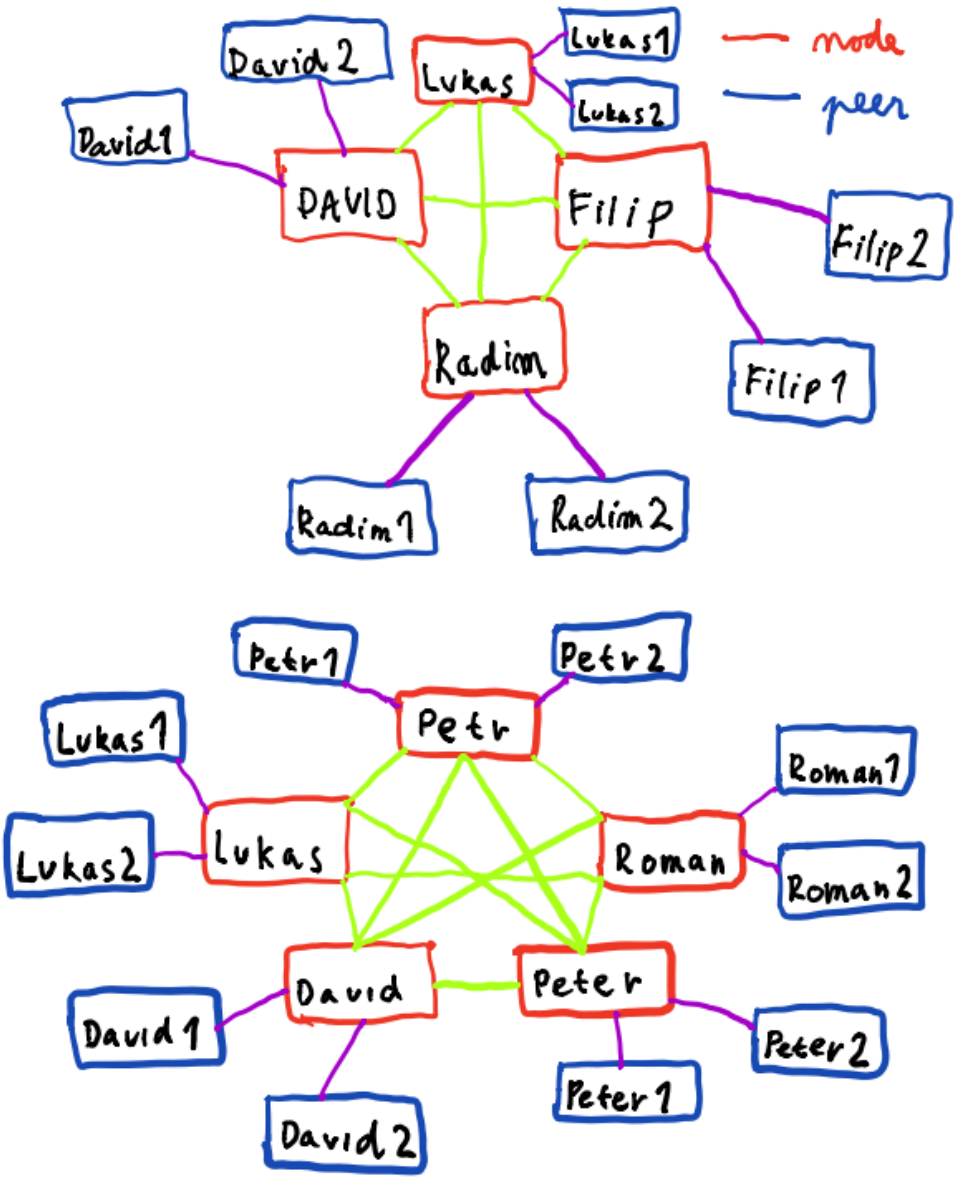
Obr. 5.3: Priebeh vytvorenia plne prepojenej siete a následne poslanie správy pomocou RPC



Obr. 5.4: Prijatie správy od prvého peera na strane tretieho peera

5.3 Skupinové testovanie implementácie

Validačné a verifikačné testovanie kompatibility a funkčnosti implementácie prebiehalo spolu s ďalšími spolužiakmi. Dňa 23. 4. 2019 som testoval projekt so 3 spolužiakmi, menovite Radimom Červinkom (xcervi21), Filipom Greplom (xgrepl05) a Lukášom Dekrétom (xdekre00). O deň neskôr som testoval s ďalšími spolužiakmi, a to s Petrom Šuhajom (xsuhaj02), Lukášom Dekrétom (xdekre00), Romanom Dobiášom (xdobia11) a Petrom Rusiňákom (xrusin03). Obrázok 5.5 zobrazuje schému siete, z ktorej sa odvíjali tieto dva testovania. Všetci zmienení spolužiaci, s ktorými som testoval, mali svoje implementácie napísané v jazyku Python.



Obr. 5.5: Schéma siete a prepojenia registračných uzlov a peerov na začiatku testovaní

Následne v piatok 26. 4. 2019 prebehlo tretie testovanie, kde už sa objavila aj implementácia v jazyku C++.

5.3.1 Prvé hromadné testovanie implementácie

Pri prvom testovaní sme ja a Radim boli pripojení na Wi-Fi sieť Eduroam v CVT na FIT VUT. Filip mal registračné uzly a peerov spustených na externom serveri. Lukáš bol pripojený ethernetom na internáte. Všetci sme si spustili registračný uzol a dvoch peerov, ktorých sme napojili na svoj registračný uzol. Následne sme prepojili naše registračné uzly, tj. vytvorili sme full-mesh sieť a následne skúšali rôzne príkazy a scenáre. Tu sme odhalili problém Wi-Fi siete v CVT, ktorá zabraňovala v priamej komunikácii medzi mnou a Radi-

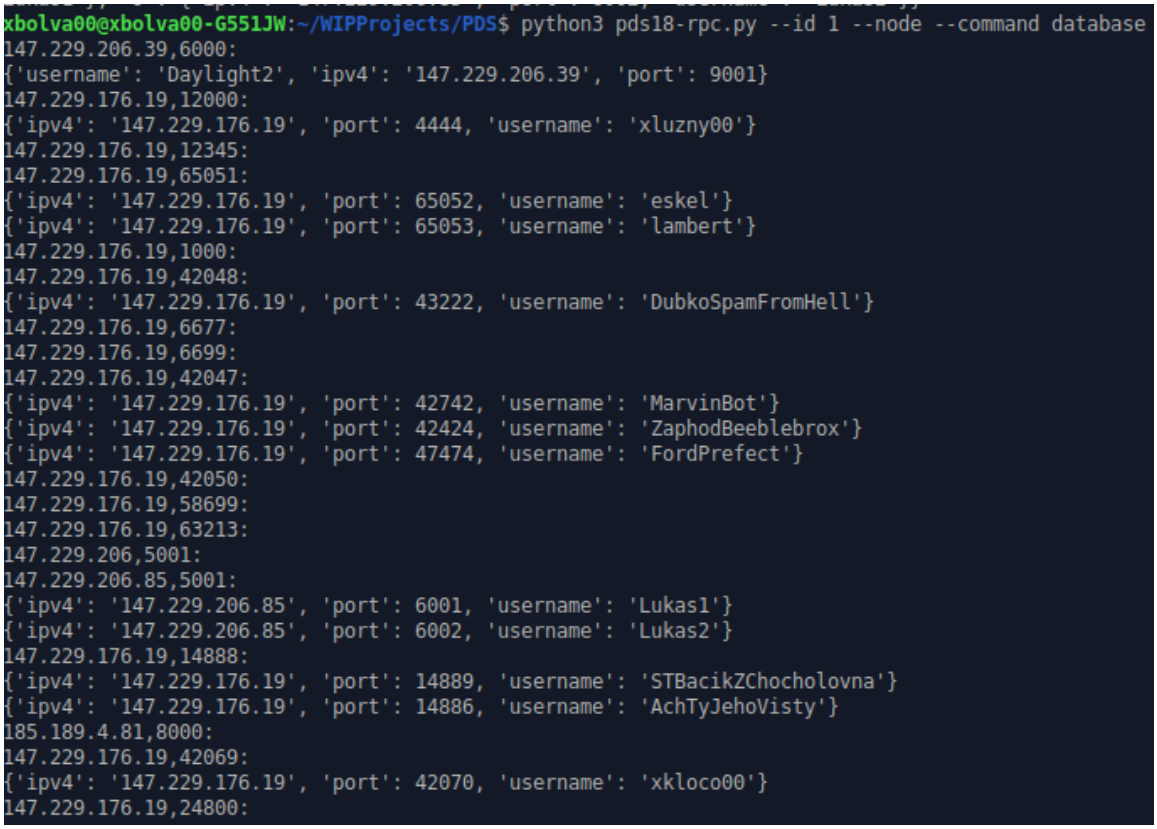
mom. Komunikácia s ďalšími spolužiakmi prebiehala úplne v poriadku. Keďže sme skúšali aj scenáre, ktoré mali viesť k chybovým hláseniam, pár zmien, ktoré vyplynuli z tohto testovania sa týkalo práve obsahu hlásení na problém pri benkodódovaní, či na nesprávne správy, ktoré nespĺňajú definovaný komunikačný protokol, aby obsahovali čo najviac informácií pre druhú stranu, ktorá poslala chybnú správu. Záverom tohto prvého (skoro dvojhodinového) testovania je, že implementácie fungovali správne pri všetkých testovaných scenároch, čo nám napadli.

5.3.2 Druhé hromadné testovanie implementácie

Druhé testovanie (približne tiež dvojhodinové) prebiehalo na internátoch. Organizácia a koordinácia testovania prebieha cez skupinový hovor na Facebook Messengeri. Všetci sme si spustili registračný uzol a dvoch peerov. Najskôr sme svojich peerov napojili na svoj uzlov a následne sme skúšali rôzne príkazy a scenáre na tejto full-mesh sieti. Až na jeden prípad, keď spolužiak potvrdzoval správy nesprávnym číslom, tj. nepoužil sekvenčné číslo zo správy, na ktorú reaguje, sme nezaznamenali žiadne problémy v implementáciách.

5.3.3 Tretie hromadné testovanie implementácie

V piatok 26. 4. 2019 zverejnil Martin Vlnas (xvlnas00), ktorý projekt implementoval v C++, IPv4 adresy a porty svojho registračného uzlu a peera. Svojho registračného uzla s jedným peerom som teda napojil na jeho registračný uzol a spolu s Petrom Rusiňákom (xrusin03), Filipom Greplom (xgrepl05) a Dominikom Polehňa (xpoleh00) - obaja s Python implementáciami - sme vytvorili plne prepojenú sieť. Druhého peera som napojil na Petrov registračný uzol. Následne sme sa prepojil s registračnými uzlami iných spolužiakov, ktorí ich mali pustené na merline. Vyskúšali sme všetky príkazy a posielanie správ. Fungovalo to správne. Následne sa pripojil do siete Lukáš Dekrét (xdekre00). Z prvého peera som mu poslal správu úspešne, no z druhého nie. Príkaz `peers` vypisoval rozdielne zoznamy aktuálnych peerov. Zistil som, že môj node posiela kompletné záznamy o peeroch, no Petrov posielal záznamy bez informácii o Lukášových peeroch. Zistenú chybu som mu nahlásil, aby sa na to pozrel. Okrem toho nenastali žiadne ďalšie problémy pri testovaní.



Obr. 5.6: Výpis databáze pri treťom testovaní na mojom registračnom uzle

Kapitola 6

Použitie programov

Projekt sa skladá s troch programov - `pds18-peer`, `pds18-node` a `pds18-rpc`. Programy sa spúšťajú cez terminál. Pri zadaní prepínača `-h/--help` sa vypíše informačný text o programe a jeho prepínačoch. V prípade neznámeho či chybne použitého prepínača (nesprávna/chýbajúca hodnota prepínača) alebo pri akejkoľvek chybe v sieťovej komunikácii (napr. problém pri bindovaní, problém napojenia RPC na peera/uzol) sa program ukončí a o probléme informuje používateľa správou na štandardný chybový výstup. Programy sa ukončujú pomocou trl-C, resp. pomocou príkazu „kill -INT <pid>“ v termináli.

6.1 Chatovací peer (pds18-peer)

Použitie: pds18-peer.py [-h] –id ID –username USERNAME –chat-ipv4 CHAT_IPV4 –chat-port CHAT_PORT –reg-ipv4 REG_IPV4 –reg-port REG_PORT

–id ID je číselný identifikátor instancie peera
–username USERNAME je unikátne používateľské meno identifikujúce peera v rámci chatu
–chat-ipv4 CHAT_IPV4 a –chat-port CHAT_PORT je IPv4 adresa a port, na ktorom peer naslúcha a prijíma správy od ostatných peerov alebo uzlov
–reg-ipv4 REG_IPV4 a –reg-port REG_PORT je IPv4 adresa a port registračného uzlu, na ktorý peer bude: 1) pravidelne zasielať HELLO správy; a 2) odosielať správu GETLIST k zisteniu aktuálneho mapovania.
-h|--help zobrazenie informácii o programe a o prepínačoch

6.2 Registračný uzol (pds18-node)

Použitie: pds18-node.py [-h] –id ID –reg-ipv4 REG_IPV4 –reg-port REG_PORT

–id ID je číselný identifikátor instancie uzla
–reg-ipv4 REG_IPV4 a –reg-port REG_PORT je IPv4 adresa a port registračného uzlu, na ktorom prijíma registrácie uzlov a synchronizácie databázy s ostatnými uzlami
-h|--help zobrazenie informácii o programe a o prepínačoch

6.3 RPC (pds18-rpc)

Použitie: pds18-rpc.py [-h] –id ID [–peer] [–node] –command COMMAND [COMMAND_ARGS]

–id ID je číselný identifikátor instancie uzla
–peer alebo –node určuje, či sa jedná o príkaz pre instanciu peera alebo registračného uzla
–command COMMAND a zoznam parametrov COMMAND_ARGS určujúcich príkaz a parametre vzťahujúce sa k danému RPC príkazu
-h|--help zobrazenie informácii o programe a o prepínačoch

6.3.1 Príkazy RPC

–peer –command message –from <username1> –to <username2> –message <správa>, ktorý sa pokúsi odoslať správu
–peer –command getlist, ktorý vynúti aktualizáciu zoznamu v sieti známych peerov
–peer –command peers, ktorý zobrazí aktuálny zoznam peerov v sieti
–peer –command reconnect –reg-ipv4 <IP> –reg-port <port>, ktorý sa odpojí od súčasného registračného uzlu a pripojí sa k uzlu špecifikovanému v parametroch príkazu
–node –command database, ktorý zobrazí aktuálnu databázu peerov a ich mapovanie
–node –command neighbors, ktorý zoznam aktuálnych susedov registračného uzla
–node –command connect –reg-ipv4 <IP> –reg-port <port>, ktorý sa pokúsi naviazať susedstvo s novým registračným uzlom
–node –command disconnect, ktorý zruší susedstvo so všetkými uzlami a odpojí uzol od siete
–node –command sync, ktorý vynúti aktualizáciu databáze s uzlami, s ktorými uzol aktuálne susedí

6.3.2 Ukážky výpisov

Nasledovné ukážky majú za cieľ oboznámiť používateľa RPC klienta s obsahom a formátom odpovedí od peera / registračného uzla pri použití niektorých RPC príkazov.

- Ukážkový výpis pre príkaz `python3 pds18-rpc.py -id 1 -node -command database`

147.229.206.39,6000:
'ipv4': '147.229.206.39', 'port': 9000, 'username': 'xbolva00'
147.229.206.39,6001:
'ipv4': '147.229.206.39', 'port': 9001, 'username': 'xbolva01'
147.229.206.39,6002:
'ipv4': '147.229.206.39', 'port': 9002, 'username': 'xbolva02'

- Ukážkový výpis pre príkaz `python3 pds18-rpc.py -id 1 -node -command neighbors`

```
147.229.206.39,6001
147.229.206.39,6002
```

- Ukážkový výpis pre príkaz `python3 pds18-rpc.py -id 1 -peer -command peers`

```
'0': 'ipv4': '147.229.206.39', 'port': 9000, 'username': 'xbolva00',
'1': 'ipv4': '147.229.206.39', 'port': 9001, 'username': 'xbolva01',
'2': 'ipv4': '147.229.206.39', 'port': 9002, 'username': 'xbolva02'
```

6.4 Príklady použitia

- `python3 pds18-peer.py -id 1 -chat-ipv4 147.229.206.39 -chat-port 9000 -username xbolva00 -reg-ipv4 147.229.206.39 -reg-port 6000`

  Spustí instanciu peera s identifikátorom 1 a používateľským menom *xbolva00*, ktorá sa napojí na daný registračný uzol.

- `python3 pds18-node.py -id 1 -reg-ipv4 147.229.180.26 -port 6000`

  Spustí instanciu registračného uzla s identifikátorom 1, ktorý bude počúvať na danej IPv4 adrese a porte.

- `python3 pds18-rpc.py -id 1 -node -command connect -reg-ipv4 147.229.206.39 -reg-port 6000`

  Naviaže susedstvo s registračným uzlom definovaným parametrami príkazu.

- `python3 pds18-rpc.py -id 1 -peer -command message -from xbolva00 -to Jan -message Ahoj`

  Pokúsi sa poslať správu „Ahoj“ peerovi s používateľským menom *Jan*.

Kapitola 7

Záver

V rámci projektu bola naimplementovaná, otestovaná a zdokumentovaná hybridná chatovacia P2P sieť skladajúca sa z peerov a registračných uzlov. Implementácia bola napísaná v jazyku Python a následne bola podrobená testovaniu kompatibility a samotnej funkčnosti s ďalšími spolužiakmi. Vďaka testovaniu sme si všetci mohli otestovať implementácie a opraviť zistené chyby a nezrovnalosti tak, aby implementácia spĺňala špecifikáciu protokolu a chovanie hybridnej chatovacej P2P siete zo zadania projektu.

Literatúra

[1] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC, RFC Editor, December 2017.
URL https://ietf.org/rfc/rfc8259.txt

[2] Camarillo, G.: Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. RFC, RFC Editor, November 2009.
URL https://ietf.org/rfc/rfc5694.txt

[3] Cohen, B.: The BitTorrent Protocol Specification. BEP, BitTorrent.org, Január 2017.
URL https://ietf.org/rfc/rfc5694.txt

[4] Ing. Petr Matoušek, M., PhD.: Siete peer-to-peer (P2P). PDS prednáška, Marec 2019.
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FPDS-IT%2Flectures%2Fp2p.pdf

[5] Postel, J.: User Datagram Protocol. RFC, RFC Editor, August 1980.
URL https://www.ietf.org/rfc/rfc768.txt