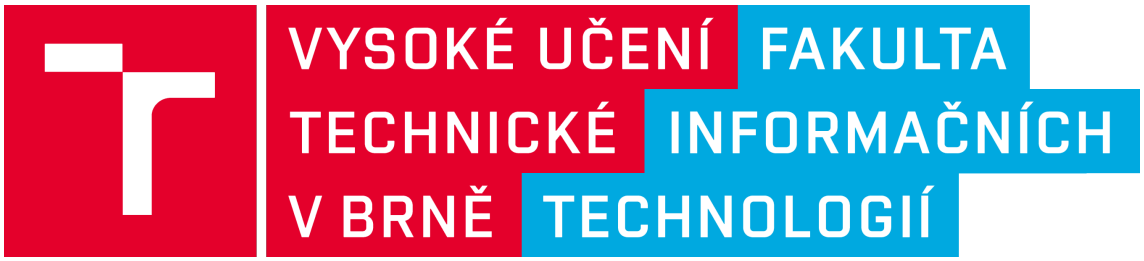



# Přenos dat, počítačové sítě a protokoly
# Hromadný projekt - Hybridní chatovací P2P síť

28. dubna 2019

Jiří Matějka (xmatej52)

## Obsah


**1 Úvod** **2**

**2 Struktura sítě** **2**

2.1 Peer . . . . . 2

2.2 Uzel . . . . . 2

**3 Komunikace** **3**

3.1 Komunikační prokol . . . . . 3

3.1.1 ACK . . . . . 3

3.1.2 HELLO . . . . . 3

3.1.3 UPDATE . . . . . 4

3.1.4 DISCONNECT . . . . . 4

3.1.5 LIST . . . . . 4

3.1.6 MESSAGE . . . . . 5

3.1.7 ERROR . . . . . 5

**4 Implementace** **5**

4.1 Registrační uzel . . . . . 6

4.2 Peer . . . . . 7

4.3 RPC . . . . . 8

**5 Testování** **9**

**6 Závěr** **9**

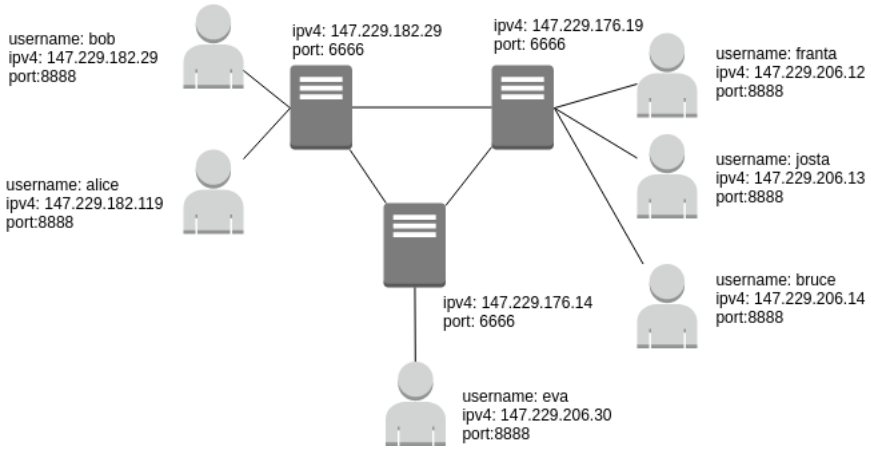
# 1 Úvod

Tato práce vznikla jako projekt do předmětu Přenos dat, počítačové sítě a protokoly na škole Vysoké učení technické v Brně. Práce se zabývá implementace chatovací sítě nad transportním protokolem UDP. Síť se skládá z libovolného počtu peerů a minimálně jednoho registračního uzlu.

# 2 Struktura sítě

Jak již bylo zmíněno v úvodu, celá síť se skládá z několika peerů a a alespoň jednoho registračnního uzlu. K registračním uzlům se připojují jednotliví peeři (uživatelé) a právě od registračního uzlu získává každý uživatel informace o dalších uživatelích v síti.

Uzel dále může navázat spojení s jiným uzlem a navzájem si tak vyměnit záznamy o registrovaných uživatelích a známých uzlech. Díky spojení mezi jednotlivými uzly může následně registrační uzel poskytnout informace i o uživatelích, které nejsou registrovaní přímo u něj, ale registrovali se k některému z jeho sousedů.




Obrázek 1: Ukázka struktury sítě

## 2.1 Peer

Každý uživatel v síti je reprezentován peerem. Peer je v síti identifikován unikátním uživatelským jménem, ipv4 adresou a portem. V rámci operačního systému je identifikován pomocí unikátního ID. Každý uživatel je registrován k právě jednomu registračnímu uzlu, který mu poskytuje informace o ostatních uživatelých v síti.

## 2.2 Uzel

Uzel je v síti identifikován svou ipv4 adresou a portem. Každý z uzlů si vede databázi známých uzlů a peerů. Uzel poskytuje informace o uživatelých v síti pouze těm

uživatelům, kteří jsou registrováni u něj. Uzly si navzájem mezi sebou pravidelně vyměňují databáze peerů i uzlů. Pokud uzel obdrží informaci o novém uzlu v síti, naváže s ním spojení.

# 3 Komunikace

Komunikace jednotlivých prvků v síti je implementována pomocí jednoduchého komunikačního protokolu. Každá zpráva mezi dvěma prvky sítě je přenášena skrz UDP a před přenosem samotné zprávy je její obsah bencodován.

## 3.1 Komunikační prokol

V rámci komunikačního protokolu bylo implementováno 8 různých zpráv. Každá z těchto 8 zpráv má syntaxi JSON a všechny jejich atributy jsou povinné. Společnými atributy je potom typ zprávy, který identifikuje, o jakou zprávu se jedná a txid, který nese unikátní identifikátor zprávy.

### 3.1.1 ACK

Zpráva `ACK` slouží k potvrzení doručení zpráv typu `GETLIST`, `LIST`, `MESSAGE` a `DISCONNECT`. V komunikačním protokolu je implementována zejména proto, že UDP negarantuje doručení. Po odeslání výše uvedených zpráv se na zprávu `ACK` čeká maximálně po dobu 2 sekund, poté se předpokládá, že zpráva nebyla doručena.

Zpráva má dva atributy – txid a type, kde txid nese identifikátor zprávy, kterou zpráva `ACK` potvrzuje. Výsledná struktura zprávy tedy bude vypadat následovně:
`{"type":"ack", "txid":<ushort>}`

### 3.1.2 HELLO

Zpráva `HELLO` slouží k registraci peeru k uzlu. Pro udržení spojení mezi peerem a uzlem odesílá peer zprávu `HELLO` každých 10 sekund. Zpráva obsahuje kromě atributů type a txid další 3 atributy identifikující uživatele – username, ipv4 a port. Pokud uzel neobdrží od uživatele po dobu 30 sekund žádnou zprávu `HELLO`, uživatele odhlásí. Uživatele odhlásí i tehdy, pokud obdrží zprávu `HELLO` s nulovou ipv4 adresou a portem.

Struktara zprávy `HELLO` vypadá následovně:

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

### 3.1.3 UPDATE

`UPDATE` slouží k navázání spojení mezi 2 uzly a výměny si informací o registrovaných peerech a známých uzlech. Zpráva `UPDATE` tedy v sobě nese seznam všech známých uzlů odesílatele včetně seznamu peerů, které jsou k jednotlivým uzlům registrovány. Zpráva update se odesílá každé 2 sekundy a pokud uzel neobdrží od některého ze svých sousedů zprávu `UPDATE` po dobu 10 sekund, přeruší komunikaci s tímto uzlem.

Struktara zprávy `UPDATE` vypadá následovně:

```
{
    "type":"update",
    "txid":<ushort>,
    "db":{
        "<dotted_decimal_IP>,<ushort_port>":{
            "<ushort>":{
                "username":"<string>",
                "ipv4":"<dotted_decimal_IP>",
                "port":<ushort>
            }
        }
    }
}
```

### 3.1.4 DISCONNECT

Po obdržení zprávy `DISCONNECT` přeruší uzel komunikaci s odesílatelem (za předpokladu, že odesílatel je registrační uzel). `DISCONNECT` obsahuje pouze atributy typ a txid a její struktara vypadá následovně: `{"type":"disconnect", "txid":<ushort>}`

### GETLIST

Příkazu `GETLIST` využívá peer k aktualizaci svého seznamu známých uživatelů v síti. Zpráva `GETLIST` obsahuje pouze atributy txid a type a její strukturlze popsat následovně: `{"type":"getlist", "txid":<ushort>}`

### 3.1.5 LIST

Zpráva `LIST` je odpovědí uzlu na příkaz `GETLIST` od peera. Zpráva nese v sobě seznam všech známých uživatelů v síti bez ohledu na to, k jakému uzlu jsou registrovaní.

Zpráva `LIST` vypadá následovně:

```
{
    "type":"list",
    "txid":<ushort>,
```

```
    "peers":{
        "<ushort>":{
            "username":"<string>",
            "ipv4":"<dotted_decimal_IP>",
            "port":<ushort>
        }
    }
}
```

#### 3.1.6 MESSAGE

Zprávu `MESSAGE` použije peer, pokud chce zaslat zprávu jinému peeru. Posílání zpráv `MESSAGE` probíhá pouze mezi peery navzájem a registračních uzlů se tato zpráva vůbec netýká. `MESSAGE` kromě atributů txid a type nese ještě atribut identifikující odesílatele (from), příjemce (to) a atribut nesoucí obsah zprávy (message).

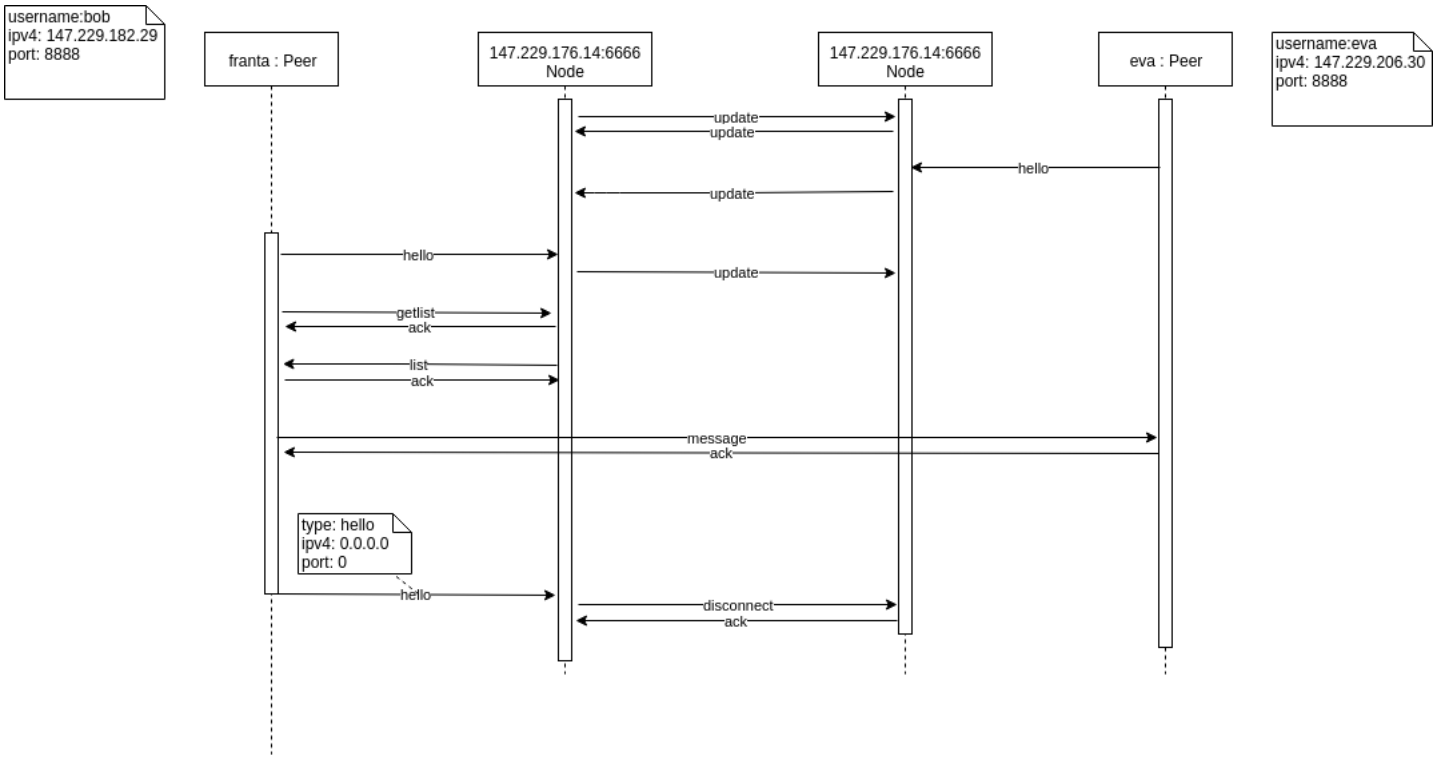
```
{
    "type":"message",
    "txid":<ushort>,
    "from":"<string>",
    "to":"<string>",
    "message":"<string>"
}
```

#### 3.1.7 ERROR

Tato zpráva se odesílá jako oznámení o chybě (neobdržení zprávy `ACK`, špatný formát přijaté zprávy, neregistrovaný užiatel odeslal uzlu příkaz `GETLIST` apod.). Kromě atributů txid a type se ve zprávě nachází jště atribut verbose, jehož hodnota nese textový popis chyby. Zpráva `ERROR` vypadá tedy následovně: `{"type":"error", "txid":<ushort>, "verbose":"<string>"}`

## 4 Implementace

Projekt je implementován v jazyce python3 a samotná implementace je rozdělena do 7 modulů, které jsou společné pro uzly i peery. Veškerý příjem zpráv je implementován v modulu `Receiver.py`, odesílání zpráv v modulu `Sender.py`, registrace k uzlu, navázání a udržení spojení uzly a udržení spojení mezi uzlem a peerm v modulu `ConnectionKeeper.py`. Komunikační protokol je implementován v modulu `Protokol.py`. Zbylé 3 moduly jsou `Functions.py`, kde jsou obsaženy užitečné funkce používané napříč mezi moduly (např. zpracování argumentů), `FileLock.py` sloužící k výlučnému přístupu k souborům a `InputReader.py` který načítá příkazy ze souborů a standartního vstupu.




Obrázek 2: Ukázka komunikace

### 4.1 Registrační uzel

Aplikace registračního uzlu je implementována v souboru `pds18-node.py`. Aplikaci lze ovládat zadáním příkazů na standartní vstup nebp pomocí aplikace RPC, která je popsána níže. Spuštění aplikace:

```
pds18-node.py --id N --reg-ipv4 IPV4 --reg-port PORT
```

- `--id` – Unikátní identifikátor uzlu v rámci operačního systému
- `--reg-ipv4` – IP adresa uzlu, na kterém bude uzel přijímat zprávy
- `--reg-port` – Port uzlu, na kterém bude uzel přijímat zprávy

Po spuštění aplikace uzel vytvoří vlákno, které pravidelně kontroluje databázi známých sousedů a případně odesílá zprávy typu `UPDATE`. Dále vytvoří vlákno, které pravidelně kontroluje databázi registrovaných peerů a známých uzlů a případně tuto databázi promazává. Další vytvořené vlákno zpracovává přijaté zprávy a tvoří na ně odpovědi. V neposlední řadě se vytvoří další vlákno, zpracovávající databázi očekávaných `ACK` zpráv. A poslední 2 vlákna, co vzniknou, načítají příkazy ze souboru (kam je zadává aplikace RPC) a ze standartního vstupu (kam je zadává sám uživatel). Rodičovský proces pouze tyto příkazy předává ostatním vláknům a případně vypisuje chyby nebo zpracované informace na chybový nebo standartní výstup. Uzel tedy vytvoří dohromady 6 vláken.

Během registrace peeru k uzlu nebo synchronizaci 2 uzlů mohou nastat kolize mezi identifikátory jednotlových uživatelů. Pokud uzel detekuje kolizi během registrace

peeru, odešle chybovou zprávu uživateli a registraci odmítne. Při kolizi během synchronizace databáze 2 uzlů odmítne pouze synchronizovat záznam uživatele, který způsobuje kolizi. V tomto případě vygeneruje zprávu na chybový výstup a odešle chybovou zprávu druhému uzlu. Synchronizace ostatních peerů proběhne normálním způsobem.

Podporované příkazy zadané na standartní vstup:

- `\c ipv4 port` – Naváže spojení se zadaným uzlem
- `\s` – Vynutí synchronizaci s ostatnímy uzly
- `\l` – Vypíše aktualní databázi uzlů a jejich peerů
- `\n` – Vypise databázi znamých uzlů
- `\d` – Odpojí se od ostatnich uzlů
- `\h` – Vypíše nápovědu
- `\exit` – Ukončí aplikaci

### 4.2 Peer

Implementace apliakce peeru se nachází v souboru `pds18-peer.py`. Stejně jako uzel, i tuto aplikaci lze ovládat pomocí RPC nebo zadávání příkazů na standartní vstup. Spuštění aplikace:

```
./pds18-peer.py --id N --reg-ipv4 IPV4 --reg-port PORT
                --chat-ipv4 IPV4 --chat-port PORT --username USERNAME
```

- `--id` – Unikátní identifikátor peeru v rámci operačního systému
- `--reg-ipv4` – IP adresa uzlu, ke kterému se peer zaregistruje
- `--reg-port` – Port uzlu, ke kterému se peer zaregistruje
- `--chat-ipv4` – IP adresa peeru, na které bude přijímat zprávy
- `--chat-port` – Port peeru, na kterém bude přijímat zprávy
- `--username` – Unikátní uživatelské jméno v rámci celé sítě

Po spuštění aplikace se vytvoří vlákno, které provede registraci peeru k uzlu a udržuje spojení pravidelným odesíláním zpráv typu `HELLO`. Dále se vytvoří vlákno, které zpracovává přijaté zprávy a vlákno, které zpracovává databázi očekávaných `ACK` zpráv. Protože ovlání aplikace je velmi podobné uzlu a je implementováno v jednom modulu, i zde je zapotřebí dalších 2 vláken, která zpracovávající příkazy od uživatele a RPC. Dohromady je pro jeden peer potřeba 5 vláken.

Podporované příkazy zadané na standartní vstup:

- `\l` – Vypíše seznam známých uživatelů

- `\u` – Aktualizuje seznam uživatelů
- `\r ipv4 port` – Připojí se na zadaný uzel
- `\w username message` – Odešle zprávu zadanému uživateli
- `\h` – Vypíše nápovědu
- `\exit` – Ukončí aplikaci

### 4.3 RPC

RPC slouží k ovládání uzlu nebo peeru, pokud není možnost zadat příkazy na standartní vstup (například když proces uzlu běží na pozadí). Implementace RPC je umístěna v souboru `pds18-rpc.py` a soubor lze spustit pomocí příkazu:

```
./pds18-rpc --id N --peer|node --command CMD --PARAMETR_1 HODNOTA_1 --PARAMETR_2 HODNOTA_2 ...
```

- `--id` – Identifikátor peeru nebo uzlu, kterému bude předán příkaz
- `--peer` – Určuje, že se jedná o příkaz pro peer
- `--node` – Určuje, že se jedná o příkaz pro uzel
- `--command` – Identifikátor příkazu
- `--PARAMETR_1` – Parametr příkazu a jeho hodnota

Chování RPC je jednoduché. RPC po spuštění zpracuje zadané argumenty a zapíše příkaz do souboru. Peer nebo uzel následně soubor otevře a načte příkazy a vykoná je. Veškeré výpisy provádí peer nebo uzel, RPC pouze zadává příkazy do souboru.

Na peeru jsou momentálně podporovány příkazy `message` (odeslání zprávy), `getlist` (aktualizace databáze známých uživatelů), `peers` (vypíše seznam peerů) a `reconnect` (změna registračního uzlu). Tyto příkazy se potom zadají při spuštění RPC následujícím způsobem:

- `-command message --from USERNAME --to USERNAME --message TEXT`
- `--command getlist`
- `--command peers`
- `--command reconnect --reg-ipv4 IPV4 --reg-port PORT`

Co se uzlu týče, jsou implementovány příkazy `database` (Zobrazí aktualní databázi peerů), `neighbors` (zobrazí aktuální databázy známých uzlů), `connect` (naváže komunikaci se zadaným uzlem), `disconnect` (přeruší komunikaci se všemy uzly) a `sync` (vynutí synchronizaci databází s ostatnímy uzly).

- `--command database`
- `--command neighbors`
- `--command connect --reg-ipv4 IPV4 --reg-port PORT`
- `--command disconnect`
- `--command sync`

## 5 Testování

Testování projektu bylo provedeno ve spolupráci s Lucií Pelantovou (implementace v jazyce python), Janem Žárským (imlementace v jazyce python), Martinou Zembjakovou (implementace v jazyce python), Pavlem Dohnalíkem (implementace v jazyce Java) a Václavem Martinkou (implementace v jazyce C++). Testování ukázalo, že mé řešení projektu je kompatibilní s řešeným mých spolužáků. Během testováni byla odhalena v mé implementaci jen 1 chyba, která se projevovala při odhlášení uživatele z uzlu. Testování také prokázalo použitelnost implementovaných nástrojů nezávisle na RPC.

## 6 Závěr

Pomocí nástrojů vyvynutých v rámci tohoto projektu lze sestavit chatovací síť stávající se z uzlů a peerů. Implementace komunikačního protokolu je bez problému kompatibilní od implementace ostatních autorů. Díky načítání příkazů ze standartního vstupu lze uzel i peer jednoduše ovládat nezávisle na RPC. Díky RPC lze řídit jednoduše činnosti uzlů a peerů i jiným procesem a není problém je ovládat, když jsou spuštěny napozadí.