VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MONITOROVANIE PEEROV BITTORRENT NA ZÁKLADE INFORMÁCIÍ Z DISTRIBUOVANEJ HAŠOVACEJ TABUĽKY

BITTORRENT TRACKER-LESS PEER MONITORING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN VAŠKO**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20527/2017/xvasko12

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Vaško Martin**
Obor: Informační technologie
Téma: **Monitorování peerů BitTorrent na základě informací z distribuované hašovací tabulky**
BitTorrent Tracker-Less Peer Monitoring
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s protokolem BitTorrent, zaměřte se na použití s distribuovanou hašovací tabulkou (DHT).
2. Nastudujte metody pro detekci peerů zaměřující se na DHT.
3. Navrhněte metodu pro monitorování peerů sdílející konkrétní torrenty a konzultujte ji s vedoucím práce.
4. Návrh implementujte.
5. Implementaci otestujte a zhodnoťte vytvořené nástroje.
6. Navrhněte možná rozšíření projektu.

Literatura:

- Emil Sit and Robert Morris: Security Considerations for Peer-to-Peer Distributed Hash Tables. In Peer-to-Peer Systems. LNCS 2429. ISSN 0302-9743 ISBN 3-540-44179-4 Springer-Verlag Berlin Heidelberg New York. 2002
- Petar Maymounkov and David Mazires: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In Peer-to-Peer Systems. LNCS 2429. ISSN 0302-9743 ISBN 3-540-44179-4 Springer-Verlag Berlin Heidelberg New York. 2002
- Liang Wang and Jussi Kangasharju, "Real-world sybil attacks in BitTorrent mainline DHT," *2012 IEEE Global Communications Conference (GLOBECOM)*, Anaheim, CA, 2012, pp. 826-832.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Polčák Libor, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom tejto práce je monitorovanie peerov v rámci siete BitTorrent. Pozornosť sa upiera na distribuované hašovacie tabuľky, konkrétne MLDHT. V poslednej dekáde sa pozornosť zameriava aj na to, ako efektívne dokážeme v tejto sieti monitorovať. Existujúce algoritmy vôbec nezaisťujú aktuálnu efektivitu, a preto sa odporúča zaviesť odhad presnosti pomocou Bernoulliho procesu. Cieľom monitorovania je zaistiť čo najvyššiu úspešnosť vyhľadania peerov. Práca sa venuje návrhom aplikácie a implementácií metód s modelom Bernoulliho procesu. Z meraní vyplýva, že prehľadávanie do hĺbky (LIFO) je lepšie z krátkodobého hľadiska. Experimenty sa venujú efektivite monitorovania v čase, odhadom chyby prehľadávania a efektivite modelovanej pomocou Bernoulliho procesu. Výsledkom práce je odporúčaná doba monitorovania s vysokou efektivitou.

Abstract

The goal of this thesis is to effectively monitor peers within BitTorrent network. This research is based on distributed hash tables, specifically MLDHT. In the last decade, focus has also been on efficiency of monitoring within network. Existing algorithms do not provide actual efficiency therefore it is recommended to introduce an estimate accuracy using Bernoulli process. The goal of monitoring is to ensure the highest success of peer search. The work focuses on designing and implementing Bernoulli process model. The measurements show that, the in-depth search (LIFO) is better in the short run. The work focuses on the efficiency of time monitoring, estimation of the peer search error and efficiency using the modeled Bernoulli process. Result of this work is the recommended monitoring time with high efficiency.

Kľúčové slová

BitTorrent, monitorovanie, distribuovaná hašovacia tabuľka, peer, MLDHT, LIFO, FIFO, Bernoulliho proces

Keywords

BitTorrent, monitoring, distribued hash table, peer, MLDFT, LIFO, FIFO, Bernoulli process

Citácia

VAŠKO, Martin. *Monitorovanie peerov BitTorrent na základe informácií z distribuovanej hašovacej tabuľky.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Monitorovanie peerov BitTorrent na základe informácií z distribuovanej hašovacej tabuľky

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Libora Polčáka Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Vaško
15. mája 2018

Poďakovanie

Chcel by som poďakovať všetkým svojím vyučujúcim počas bakalárskeho štúdia za odborné prednášky a vedúcemu bakalárskej práce za odbornú pomoc.

Obsah

Kapitola 1

Úvod

Zdieľanie súborov na internete je v dnešnej dobe jedna z vecí, ktoré potrebuje každý z nás. Týka sa to zdieľania dokumentov, zdrojových súborov pre rôzne technické účely, ale aj zdieľanie nelegálneho obsahu napr. software, filmy alebo hudba.

V tejto práci je nutné vysvetliť dva pojmy:

- `Peer` je klient/server načúvajúci na TCP alebo μTP porte, ktorý implementuje BitTorrent protokol.
- `Uzol` (ang. node) je klient/server načúvajúci na UDP porte implementujúci protokol distribuovanej hašovacej tabuľky.

Ak softvérová aplikácia obsahuje implementáciu distribuovanej hašovacej tabuľky, tak jeden klient predstavuje zároveň peera aj uzol. Distribuovaná hašovacia tabuľka (ďalej už len `DHT`) je zostavená z uzlov a ukladá si umiestnenia peerov. BitTorrent klienti predstavujúci DHT uzol, ktorý je používaný na kontaktovanie iných uzlov v DHT, lokalizujú peerov.

V tejto práci sa snažím o monitorovanie uzlov, ktorí zdieľajú umiestnenia peerov v takzvaných *peer–2–peer* sieťach pomocou DHT. Kapitola 2 sa opiera o základné definície siete, formát súborov, ich špecifikáciu a formát správ v tejto sieti. DHT sa zaviedla kvôli decentralizovaniu záťaže serverov, na ktoré sa denne posielajú milióny správ. Kademlia je konkrétna implementácia DHT, ktorou sa inšpirovala BitTorrent sieť. Použitím Kademlie povoľujeme zdieľanie údajov o iných uzloch v rámci distribuovanej siete. Vlastnosťami ako je použitie `XOR metriky`, alebo rýchlosť vyhľadávania uzlu podľa pozície vo vedre, predbehla bežné DHT. Stala sa základným kameňom pre vyhľadávanie a zasielanie informácií o uzloch v implementáciach ako je napr. Vuze, KAD, Mainline DHT (ďalej už len *MLDHT*).

Kapitola 3 obsahuje algoritmy pre detekciu uzlov v rôznych systémoch. Metódy sú založené na správach Kademlia. Odlišovať sa budú hlavne počtom vyhľadaných uzlov za jednotku času a systémom, v ktorom sa uzly nachádzajú. Cieľom je nájsť metódu, ktorou bude možné rýchlo a efektívne vyhľadať potrebný počet uzlov.

Návrh riešenia, ktorý bol implementovaný sa nachádza v kapitole 4. Kapitola sa zaoberá vstupom, výstupom, krátkym algoritmom, ktorý slúži ako návrh pre konkrétnu implementáciu.

Zhrnutie konkrétnej implementácie aj s triednym návrhom je načrtnuté v kapitole 5.

Na základe testovania sa implementácia upravovala, aby sa dosiahlo čo najlepších výsledkov. Testovanie v kapitole 6 obsahuje navrhnuté experimenty a testy. Tieto experimenty overujú funkciu DHT a snažia sa určiť efektivitu a efektívnu dĺžku prehliadania. V predposlednej kapitole 7 sú navrhnuté možné rozšírenia a zhrnutie práce sa nachádza v záverečnej kapitole.

Kapitola 2

BitTorrent a distribuovaná hašovacia tabuľka

Protokol BitTorrent bol jeden z prvých pokusov, kde sa od staromódneho protokolu na zdieľanie obsahu FTP snažili vývojári presadiť nový systém zdieľania obsahu, aby neboli servery s týmto obsahom tak zaťažené. Protokol sa ujal hlavne kvôli svojej vlastnosti zdieľania. Je založený na tom, že peer optimalizuje distribúciu súboru do takzvaného roja užívateľov, a zároveň užívatelia, ktorí si už stiahli určitú časť súboru ponúkajú túto časť do roja pre ostatných. `Roj` užívateľov (ang. swarm) predstavuje skupinu peerov, zdieľajúcich jeden konkrétny súbor vrámci BitTorrent siete. V dnešnej dobe sa veľmi rozšírila a vyvinula distribuovaná varianta. Takže sa rozložila záťaž viac medzi klientov, ktorí ponúkajú obsah po stiahnutí z centrálnych serverov.

2.1 Tracker

Tracker[1] je centrálny server pre daný torrent súbor. Tracker si udržuje informácie o všetkých peeroch, ktorí majú Vami sťahovaný súbor. Každý peer komunikuje s trackerom a žiada si od neho dáta o iných peeroch. Po tejto fáze nastáva TCP komunikácia medzi uzlami, kde sa daný obsah zdieľa (v tejto fáze sa server nezúčastňuje výmeny, iba informatívne). Možnosť pripojenia sa k roju je sprostredkovaná pomocou klientskej aplikácie, ktorá spracuje .torrent súbor, alebo magnet-odkaz. Tejto variante sa taktiež hovorí *tracker torrent*.

2.2 Distribuovaná hašovacia tabuľka

BitTorrent používa *distribuovanú hašovaciu tabuľku* na uloženie kontaktných informácií o peerovi pre *tracker-less* torrent. V tomto prípade sa každý peer stáva trackerom. Protokol je postavený na Kademlia [18] a je implementovaný cez UDP. Kademlia obsahuje vedrá, ktoré reprezentujú to, do koľkých častí je rozdelená `smerovacia tabuľka uzlu`. Prázdna smerovacia tabuľka má 1 vedro. Každé takéto vedro zároveň dokáže udržať K uzlov.

[1] https://www.howtogeek.com/141257/htg-explains-how-does-bittorrent-work/

2.3 Prehľad implementácie Kademlia

Každý uzol má globálne jedinečný identifikátor tiež známy ako identifikátor uzlu (node ID). Identifikátory uzlu sú vyberané náhodne z rovnakého 160-bitového priestoru ako BitTorrent `infoheš`. Tento identifikátor uzlu je zhodný `s kľúčom Kademlia` (pre vyhľadávanie v DHT sa používa termín kľúč, pre BitTorrent infoheš). Pre pripájanie ku konkrétnym súborom je potrebné vedieť infoheš súboru, ktorý je ukrytý v súbore .torrent alebo v magnet-odkaze. Infoheš je tvorený pomocou `SHA1` hešovacím algoritmom s vysokou mierou entropie, aby bola zaistená `jedinečnosť infohešu` pre daný súbor.

Metrika vzdialenosti je používaná na porovnanie dvoch identifikátorov uzlov alebo identifikátora uzlu a infohešu, pre zistenie ako je uzol blízko danému infohešu resp. uzlu. Uzly musia udržiavať smerovaciu tabuľku obsahujúcu kontaktné informácie pre malý počet iných uzlov. Smerovacia tabuľka sa v čase stáva podrobnejšiou.

Dnes je možné získať infoheš už väčšinou len z takzvaného `magnet-odkazu`[2]. Magnet-odkaz funguje ako URI schéma a odkazuje sa na konkrétny súbor podľa kryptografického odtlačku. Tento spôsob je abstrakciou torrent súborov, aby si nemusel používateľ sťahovať žiadne súbory. Otvorením magnet-odkazu v klientskej aplikácií pre sťahovanie torrent súborov sa priamo pripojí do roja, kde sa zdieľa daný obsah. Týmto infohašom dokážeme znížiť zaťaženie trackera, ktorým sa dopytujeme, kto daný obsah zdieľa.

Pre vytvorenie súboru BitTorrent s podporou DHT slúži schéma 1. Súbor musí uviesť miesto kľúča *announce* (označuje tracker), kľúč *nodes* [18]. Nodes kľúč musí byť nastavený na k najbližších uzlov. Torrent súbor má infoheš skrytý v info sekcii, v časti *'pieces'*. Infoheš v magnet-odkaze sa nachádza vždy za sekvenciou *xt=urn:btih:hash*.

Schéma 1 Upravená schéma v súbore .torrent pre vyhľadávanie DHT uzlov

nodes = [[„<host>”, <port>], [„<host>”, <port>], ...]
nodes = [[„127.0.0.1”, 6881], [„your.router.node”, 4804]]

`príklad .torrent súboru s infohešom a DHT uzlami`

```
{
    'creation date': 1515929601,
    'info':
    {
        'name': "debian-503-amd64-CD-1.iso",
        'piece length': 262144,
        'length': 678301696,
        'pieces': "841ae846bc5b6d7bd6e9aa3dd9e551559c82abc1...
        d14f1631d776008f83772ee170c42411618190a4",
        'nodes': [["147.229.216.41", 6843]]
    }
}
```

Nasledujúci príklad poukazuje na to ako môže vyzerať konkrétny magnet odkaz. Schéma 2 zahŕňa to, čo všetko magnet-odkaz môže obsahovať.
`príklad magnet-odkazu iba s infohešom:`

```
magnet:?xt=urn:btih:d2474e86c95b19b8bcfdb92bc12c9d44667cfa36
```

[2] https://en.wikipedia.org/wiki/Magnet_link

Schéma 2 obsah magnet-odkazu

magnet:? xl = [Veľkosť v Bytoch] & dn = [(URL kódované meno súboru)] & xt = urn: tree: tiger: [TTH hash (Base32)]

2.3.1 Metrika a vyhľadávanie

Uzol pre nájdenie peera zdieľaného torrentu použije vzdialenostnú metriku, aby porovnal infoheš torrentu s identifikátormi uzlov v jeho smerovacej tabuľke. Dostane sa do kontaktu s uzlami, o ktorých vie, že ich identifikátori sú najbližšie pri sebe vzhľadom k infohešu. Tento uzol pri vyhľadávaní podá dopyt na zistenie kontaktnej informácie o peeroch, ktorí momentálne sťahujú torrent.

✓ Kontaktovaný uzol vie o peeroch, ktorí aktuálne sťahujú torrent. Údaje kontaktovaného peera sa vrátia ako odpoveď dotazu.

✗ Kontaktovaný uzol musí odpovedať s kontaktnými informáciami uzlov, ktoré má uložené vo svojej smerovacej tabuľke a sú najbližšie infohešu torrentu. Pôvodný uzol opakovane vyhľadáva uzly, ktoré sú k cieľovému infohešu najbližšie. Po vyčerpaní vyhľadávania klient vloží kontaktné informácie peera pre seba na odpovedajúce uzly s čo najbližším identifikátorom k infohešu torrentu.

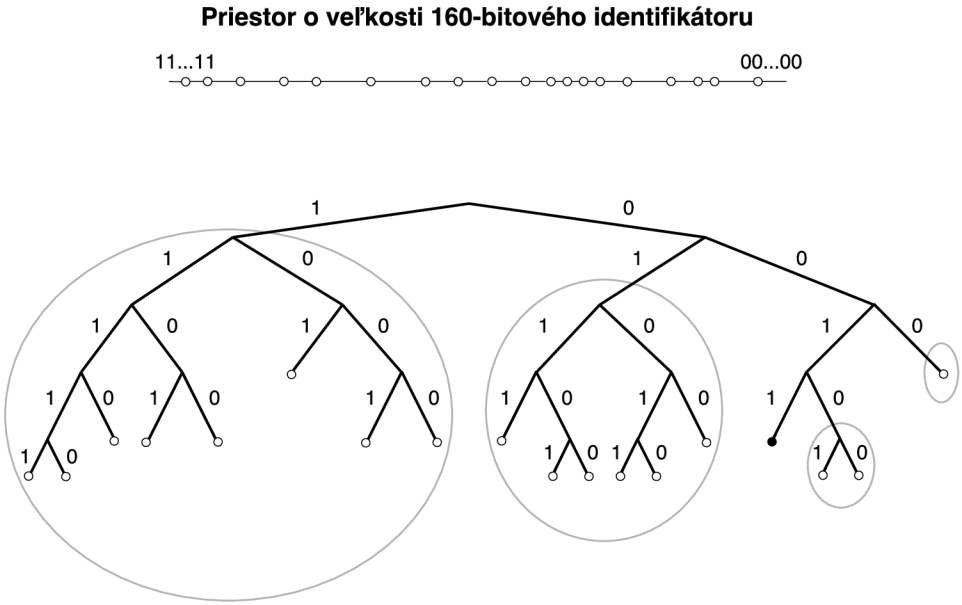
V Kademlií je vzdialenostná metrika *XOR* a výsledok je interpretovaný ako celé kladné číslo. Vzdialenosť z A do B sa počíta ako $vzdialenosť(A,B) = A \oplus B$.

Kademlia efektívne zaobchádza s uzlami, ktoré sú reprezentované ako listy v binárnom strome. Ako je vidieť z obrázku 2.1, umiestnenie každého uzla je určené najkratším jedinečným prefixom jeho ID. Zobrazuje tak polohu uzla s jedinečným prefixom `0011` v strome na obrázku. Pre každý dodaný uzol rozdeľujeme binárny strom na sériu postupne nižších podstromov, ktoré neobsahujú dodaný uzol [19]. Najvyšší podstrom pozostáva z polovice binárneho stromu, ktorý neobsahuje uzol 0011. Ďalší podstrom sa skladá z polovice zostávajúceho stromu, ktorý neobsahuje čierny uzol na obrázku. Takto sa pokračuje, kým sa uzol nedopracuje k najmenšiemu podstromu.

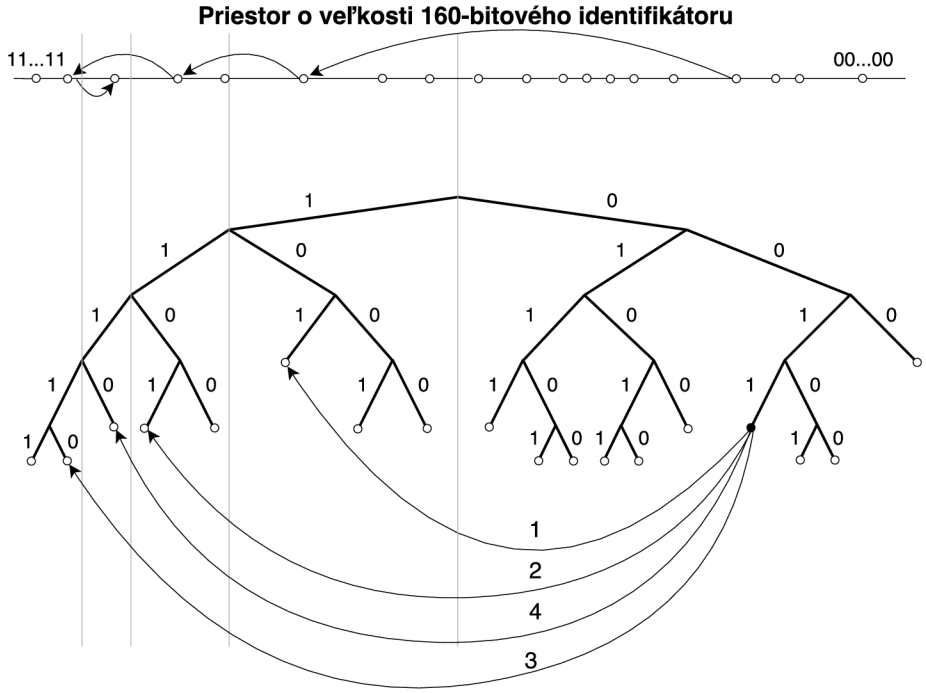
Kademlia je úložisko objektov kľúč-hodnota. Každý objekt je uložený v k najbližších uzloch, kde k je počet najbližších uzlov k identifikátoru objektu. V príklade uzla 0011 sú podstromy zobrazené v sivom ovále a pozostávajú zo všetkých uzlov s prefixami 1, 01, 000 a 0010.

Každý uzol musí spravovať k-vedro v svojej smerovacej tabuľke. K-vedro je zoznam referencií na uzly so vzdialenosťou medzi 2^i až 2^{i+1} pre i=1 do i=N. Každé takéto vedro má maximálne k záznamov (uvádza sa aj termín n zóna [16]).

Kademlia protokol zaručuje, že každý uzol pozná `aspoň jeden uzol` v každom podstrome, ak podstrom obsahuje najmenej jeden uzol. To je zárukou toho, že každý uzol môže nájsť iné ľubovoľné ID uzla z akéhokoľvek podstromu. Obrázok 2.2 znázorňuje príklad uzlov 0011 a 1110 postupným vyhľadávaním najlepšieho uzla, o ktorom vie, že nájde kontakty opakovane v menších podstromoch. Nakoniec vyhľadávanie konverguje k cieľovému uzlu.



Obr. 2.1: Kademlia binárny strom. Čierna bodka znázorňuje uzol s prefixom 0011 v strome. Sivé ovály ukazujú na podstromy(vedrá), v ktorých uzol 0011 musí poznať aspoň 1 kontakt [19].



Obr. 2.2: Hľadanie uzlu podľa ID. Čierna bodka znázorňuje uzol 0011, ktorý nájde uzol s prefixom 1110. Dosiahne to postupným učením a dopytovaním sa uzlov bližších k cieľovému uzlu. Vrchná sekvencia označuje ID priestor zložený z 160 bitov infohešu a znázorňuje konvergenciu vo vyhľadávaní k cieľovému uzlu. Prvá správa pre vyhľadávanie je určená uzlu s prefixom 101, ktorý uzol 0011 už pozná. Ďalšie správy sú určené uzlom, ktoré dostal uzol 0011 ako odpoveď predchádzajúcich správ [19].

Kademlia má niekoľko žiadúcich funkcií, ktoré nie sú zároveň ponúkané žiadnym predchádzajúcim DHT.

1. Minimalizuje počet konfiguračných správ, ktoré uzly posielajú, aby sa učili o sebe navzájom. Konfiguračné informácie sa automaticky šíria ako vedľajší efekt vyhľadávania kľúčov.

2. Uzly majú dostatočnú vedomosť a flexibilitu, aby smerovali dotazy po cestách s nízkou latenciou. Kademlia navyše používa paralelné asynchrónne dotazy, aby zabránila oneskoreniu z nefunkčných uzlov.

3. Každé opakovanie vyhľadávania sa priblíži o 1 bit bližšie k cieľu. Základná sieť Kademlia je tvorená 2^n uzlami, kde v najhoršom prípade nájdeme uzol za n krokov.

4. Algoritmus, s ktorým sa uzly navzájom kontaktujú, odoláva niektorým základným útokom DDoS (distributed denial of service).

Protokolové správy v nemodifikovanej verzii Kademlia sú 4. Dve správy slúžia na udržiavanie aktuálnosti smerovacej tabuľky. Zvyšné dve správy slúžia k získaniu dostupnosti uzla a nájdeniu uzla. Všetky tieto vyhľadávania sa opakujú. To znamená, že najprv sa vyhľadávajú vzdialené uzly a vyberajú sa z nich tie `bližšie` v súbore získaných hodnôt kľúčov od uzla. Súbor obsahuje jak aktuálny stav posledného prehliadania, tak aj predošlé kroky.

2.3.2 Protokolové správy modifikovanej Kademlia

Protokolové správy sú označované ako KRPC (Kademlia Remote Procedure Call). Je to jednoduchý mechanizmus zasielania správ v rámci distribuovaného systému, keď necháme vzdialený uzol spracovávať požiadavky a on nám zašle odpoveď [4]. Kademlia používa benkódované[3] slovníky, ktoré sa posielajú cez UDP. Posiela sa jeden datagram dopytu, na ktorý je vždy zaslaný jeden datagram ako odpoveď. Správy sú troch typov: dopyt, odpoveď a chyba. Pre Kademlia protokol existujú štyri správy dopytu: `ping`, `find_node`, `get_peers` a `announce_peer`.

KRPC správa je slovník s tromi kľúčmi pre každú správu a podľa typu správy môže obsahovať prídavné kľúče. Každá správa má kľúč „t", ktorý je typu reťazec a reprezentuje ID transakcie. Toto ID je generované uzlom, ktorý generuje dopyt, a je volané v odpovedi. Odpoveď je možné korelovať rekurzívnym dopytovaním iných uzlov. ID transakcie by malo byť kódované ako krátky reťazec zložený z binárnych číslic, typicky sú 2 znaky dostačujúce, aby pokryli 2^{16} zvyšných dopytov [18]. Každá správa obsahuje kľúč „y" s jedným znakom, ktorý popisuje typ správy. Je to buď „q" pre dopyt, „r" pre odpoveď a „e" pre chybu. Kľúč „v" by mal byť zahrnutý v každej správe s verziou reťazca klienta. Tento reťazec by mal obsahovať 2 znaky klientskeho identifikátoru. Nasleduje prídavný kľúč „a" (Argumenty), ktorý sa líši podľa typu správ:

1. `PING` ≫ najzákladanejšia správa v DHT. Správa sa označuje ako „q"=„ping", čo znamená ping query [18]. Má jeden argument *id*, ktorého hodnota je 20 bajtový reťazec v sieťovom poradí obsahujúci identifikátor uzlov odosielateľov. Odpoveďou zistíme, či je daný uzol stále v stave online. Argumenty správy <*id*>. Odpoveď <ID uzlu>.

2. `FIND_NODE` ≫ Táto správa sa používa pre nájdenie kontaktných informácií uzlu s daným identifikátorom. V správe sa označuje ako „q"=„find_node" má dva argumenty,

[3] https://en.wikipedia.org/wiki/Bencode

id argument obsahujúci identifikátor odosielateľa a *target* obsahujúci identifikátor vyhľadávaného (cieľového) uzla [18]. Odpoveďou je kľúč *nodes*. Jeho hodnota obsahuje kompaktné informácie o cieľovom uzle alebo K najbližších uzlov v cieľovej smerovacej tabuľke. Argumenty správy <*id*, *target*>. Odpoveď <ID uzlu, *nodes*(kompaktné informácie o uzloch)>.

3. `GET_PEERS` ≫ Správa tohoto typu je pre nájdenie uzlov a peerov spojených s infohešom. Správa má tvar „q"=„get_peers" a má dva argumenty. *id* argument obsahujúci identifikátor odosielateľa a *info_hash* obsahujúci infoheš torrentu [18]. Kladná odpoveď uzlov s daným infohešom sa vráti iba vtedy, ak poznajú peerov s daným infohešom. Neúspech vracia kľúč *nodes* obsahujúci K uzlov, ktoré sú v smerovacej tabuľke príjemcov najbližšie k infohešu. Záznam *token* je tiež obsahom odpovede. Hodnota token je potrebná pre ďalšiu správu announce_peer. Argumenty správy <*id*, *info_hash*>. Obsah kladnej odpovede <ID uzlu, *token*, peer_info>. Obsah zápornej odpovede <ID uzlu, *token*, *nodes*>.

4. `ANNOUNCE_PEER` ≫ Oznámenie peera, ktorý ovláda uzol dopytovania, že aktuálne sťahuje torrent. Táto správa obsahuje štyri argumenty <*id*, *info_hash*, *port*, *token*>. Odpoveďou je <ID uzlu>. Správa má voliteľný argument *implied_port* typu bool. Nenulová hodnota tohoto argumentu znamená, že je ignorovaný argument *port* a zdrojom portu UDP datagramu bude port peera. Tento mechanizmus je prospešný pre peerov za NAT, ktorí nepoznajú vonkajší port a podporujú μTP, ktorý zabezpečuje riadenie zahltenia, zníženie odozvy atp. [18].

2.3.3 Obsluha uzlov pri zdieľaní

Obrázok 2.3 zobrazuje bežnú obsluhu v Kademlií. Predpokladáme, že máme 3 uzly A, B a C. A má súbor s infoheš hodnotou $x = 59$. Povedzme, že uzol B je zodpovedný za ukladanie hodnoty x, pretože je najbližšie infohešu 59. Uzol C chce stiahnuť tento súbor [28].

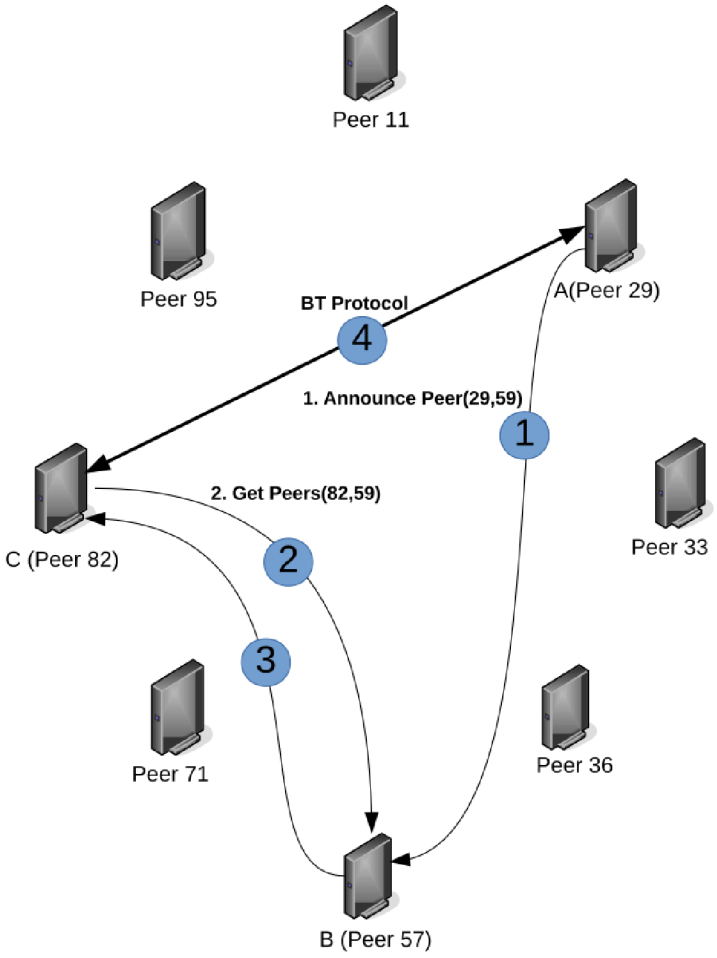
1. A chce súbor uverejniť uložením infohešu x na uzol B. V tomto prípade zavolá A procedúru `GET_PEERS` opakovane dopytovaním uzlov zo súboru .torrent. Postupne sa tak dostane bližšie k B, až kým ho nedosiahne (vysvetlené nižšie). Vtedy A použije procedúru `ANNOUNCE_PEER`, aby ohlásil možnosť stiahnutia (zdieľania) súboru s infohešom x uzlu B. B si uloží kontaktné informácie o A do príslušnej peer množiny pre x. Keďže uzol A je jediný vydavateľ infohešu x, v tejto množine sa nachádza sám. [28]

   Keď A zašle znova `GET_PEERS` správu, môžu sa vyskytnúť dve rôzne situácie.

   - Ak uzol dopytu už pozná daný infoheš a uloží peerov do korešpondujúcej množiny, odpoveďou bude množina peerov.
   - Ak uzol nepozná infoheš, odpovedá s k najbližšími uzlami k infohešu z jeho smerovacej tabuľky. Takýmto spôsobom sa bod X dostane bližšie a bližšie k bodu Y, až úplne dosiahne bod Y [28].

2. Ak chce uzol C stiahnuť súbor musí najprv dostať x napr. z .torrent súboru. Platí preň to isté ako pre A. C použije `GET_PEERS`, aby sa dostalo k B.

3. Keďže B má už uloženú množinu peerov pre x, C získa počiatočnú množinu peerov od B (v odpovedi dostane uzol A).



Obr. 2.3: Bežná obsluha uzlov

4. *C* sa prihlási do roja užívateľov, nastaví TCP (dnes μTP) spojenie s peermi v množine a získa metadáta od ostatných peerov používajúcich BitTorrent [11, 10]. Týmto momentom sa začína proces sťahovania [28].

2.4 Kademlia implementácie a ich špecifikácia

Distribuovaný systém Kademlia má tri populárne implementácie BitTorrent: KAD, Vuze[4] a MLDHT[5]. Každá z týchto implementácií beží nad operáciami Kademlia DHT. Najpopulárnejšou peer-to-peer implementáciou je MLDHT napriek už existujúcim výskumom, ktoré tvrdia opak [20]. Mnoho aplikácií ho dnes podporuje. Druhou populárnou implementáciou je Vuze. Napriek tomu, že obe implementácie sú založené na Kademlia, sú v zásade nekompatibilné protokoly, pretože sa odlišujú v obsahu správ uvedených v 2.3.2.

V každom BitTorrent systéme je peer zapojený do dvoch operácií. Operáciami sú *search* pre hľadanie uzlov a *publish* pre zdieľanie obsahu. Uskutočnením týchto operácií nad zada-

[4]https://www.vuze.com/
[5]https://en.wikipedia.org/wiki/Mainline_DHT

ným identifikátorom uzla dochádza k príslušným akciám. Vyhľadávacia sekvencia je vždy definovaná sekvenciou KRPC správ. KRPC Sekvencia často navštevuje kontakty, ktoré sa ešte nenachádzajú v smerovacej tabuľke [3]. V týchto systémoch sa očakáva, že získame ID zo zoznamu *nodes* z .torrent súboru alebo magnet-odkazu. Očakávané ID sa bude nachádzať veľmi blízko zadanému uzlu, alebo to bude zadaný uzol zo zoznamu.

Podľa nedávneho príspevku v konferencií [16] boli metódy pre monitorovanie `uzlov` do roku 2012 dosť nepresné. Tento jav vznikol kvôli tomu, že sa nezamýšľalo nad tým, koľko uzlov je možné v rámci jedného vedra vynechať a koľko nájsť. Keďže nemusíme nutne stihnúť detegovať všetky uzly v určitom časovom kvante, dochádza k rozdeleniu priestoru DHT v jednotlivých priechodoch vyhľadávania. Preto je potrebné rozlišovať počet možných uzlov vo vedre a počet uzlov, ktoré sú neaktívne. Tento jav objasňuje `Bernoulliho proces` [16]. Predpokladáme, že náš prehľadávač bude vždy vynechávať niektoré ID. Potom môžeme modelovať vzorkovací proces prehľadávača ako Bernoulliho proces[6]. To znamená, že pre každý identifikátor v zóne existujú dve možnosti:

1. ID je vybrané. Bernoulliho proces zvýši počet výskytov v zóne o 1.
2. ID chýba. Bernoulliho proces zaznamenal stratu v zóne. Počet výskytov sa nezvýši.

Pravdepodobnosť výberu uzlu označme ako p. Pravdepodobnosť chýb je potom $1-p$, takže všetko čo potrebujeme zistiť je presný odhad počtu uzlov v zóne. Odhadom je pravdepodobnosť p.

2.4.1 MLDHT

Uzol je identifikovaný 160 bitovým ID, ktoré `nie je perzistentné` tj. ak sa uzol pripojí do systému, vygeneruje mu klientska aplikácia nový identifikátor. Táto skutočnosť bude mať za následok nevýhody v detegovaní dĺžky komunikácie medzi peermi, ktoré bude spomenuté v podkapitole 3.2.

MLDHT sa líši od pôvodnej Kademlia v štruktúre vedra. Účasť v MLDHT začína iba s `jedným vedrom`. Ak je vedro naplnené (presiahneme limit $\mathcal{O}(log_2(n/k))$) a chceme vložiť nový identifikátor, nastane jedna z týchto možností:

- Vedro sa rozdelí iba vtedy, ak sa náš identifikátor nachádza v rozsahu vedra. Rozdelené vedro sa nahradí dvoma novými, z ktorých každé má polovicu rozsahu starého vedra a uzly starého vedra sa rozdelia automaticky podľa rozsahu. Do nového vedra vložíme nový identifikátor.
- Na všetky uzly mimo rozsahu vedra je zaslaná správa typu ping, aby sme zistili dostupnosť. Ak uzol neodpovedá zmažeme ho z vedra a vložíme nový identifikátor.

Z týchto informácií vyplýva, že MLDHT má menšie pamäťové nároky na smerovaciu tabuľku ako pôvodna Kademlia (zvyčajne menej než 160 vedier). Pre približne 7 miliónov uzlov v sieti MLDHT pri veľkosti vedra k=8 je počet vedier 20-22.

Podrobný popis o smerovaní v rámci systému popisuje špecifikácia [3].

2.4.2 KAD

Každý uzol KAD má globálny identifikátor, KAD ID, ktorý má 128 bitov a je náhodne generovaný kryptografickou hešovacou funkciou. Tento identifikátor si klientská aplikácia pri

[6] https://math.tutorvista.com/statistics/bernoulli-process.html

prvom štarte vygeneruje a `permanentne` uloží. Zmena ID je možná iba zmazaním aplikácie alebo zmenou nastavení. Takto je možné sledovať peera napriek zmene jeho IP adresy. Popularita tejto implementácie postupne klesá, lebo má permanentné ID, čím je možnosť dohľadať konkrétneho používateľa veľmi vysoká.

2.4.3 Vuze

Implementácia klienta Vuze okrem Kademlia zahŕňa implementáciu Vivaldi [8]. Jedná sa o `sieťový koordinačný systém`. Tento systém priradí každému uzlu v sieti súradnice, ktoré umožňujú presné predpovedanie oneskorenia v sieti medzi ľubovoľným párom uzlov. Pri vyhľadávaní obsahu sa používateľ obracia na peerov jak pomocou trackerov tak pomocou DHT. Uzol v kliente Vuze má kľúč ID o veľkosti 160 bitov. Identifikátor je unikátny a je to SHA-1 heš z dvojice *IP adresa:port*.

Z dôvodu existujúceho koordinačného systému je Kademlia modifikovaná. Systém so zmiešaním Kademlie a koordinácie musí ukladať do vyrovnávacej pamäte údaje o uzloch po ceste a šifrovať prenos údajov. Smerovacia tabuľka ostavá v tvare stromovej štruktúry, ktorá pozostáva z vedier. Každé vedro zodpovedá dĺžke Vuze ID prefixu a obsahuje až 20 uzlov. Detailnejší popis Vivaldi a princíp funkcie smerovania vo Vuze popisuje [6].

2.5 Prehľad existujúcich implementácií

Stručný prehľad Kademlia implementácií je ponúknuty v tabuľke 2.1.

Systémy	Popularita	Typ ID uzlu	Vedro	Kladné vlastnosti
MLDHT	Najvyššia	Nové generovanie ID pri pripojení.	Začína iba s 1 vedrom, ak je naplnené zisťuje sa buď dostupnosť alebo sa vedro rozdelí na polovicu ak vybraný identifikátor spadá do tohto vedra [24].	Menšie pamäťové nároky
		160 bit, Neprezistentné ID.		
KAD	Priemerná	Pezistentné ID o veľkosti 128 bitov.	Kademlia o veľkosti 50 vedier [26].	Možnosť sledovať peera aj po zmene IP.
Vuze	Vysoká	ID o veľkosti 160 bitov.	Kombinuje sieťový koordinačný systém (Vivaldi) s Kademlia.	Systém, ktorý umožňuje predpovedanie oneskorenia medzi ľubovoľnými 2 uzlami.
		ID je heš kombináciou IP:port.	Kademlia o veľkosti 20 vedier [6].	

Tabuľka 2.1: Prehľad implementácie DHT

Kapitola 3

Metódy pre detekciu peerov

Monitorovanie peerov za pomoci Kademlia, bez zisťovania konkrétnych BitTorrent informácií, sa zameriava iba na 2 typy správ: get_peers a find_node. Pre zistenie dodatočných informácií, prípadne monitorovanie priebehu sťahovania, sa kombinujú s použitím samotného BitTorrent protokolu. Preto sú ďalej zmienené metódy rozdelené podľa prípadu použitia, konkrétnej špecifikácií systému (KAD, Vuze, MLDHT) a efektivity (vhodná zóna na monitorovanie a pod.).

Metódy pre detekciu môžeme rozdeliť na rôzne kategórie. Pomocou použitej metodiky plná detekcia alebo detekcia na základe k-bitovej zóny (výrez DHT priestoru). Klientské aplikácie sa stretávajú väčšinou s implementáciou MLDHT. Ak je nutné použiť akúkoľvek neštandardizovanú komunikáciu, táto komunikácia môže byť označená ako `podozrivý klient`.

3.1 Nežiaduce javy pri monitorovaní pomocou DHT

Nezávislý príchod a odchod peerov sa nazýva v terminológii distribuovaných systémov *churn* (vír). Vír je hlavnou výzvou pre peer-to-peer systémy, pretože ovplyvňuje stabilitu celého systému. Týka sa to dostupnosti peerov a zdieľaných objektov. Vír navyše sťažuje detegovanie peerov. Aby sme získali konzistentný snímok systému v danom okamihu, je potrebné vykonať danú detekciu v čo najmenšom čase [27].

Jav je ilustrovaný tak, že ak sa prechádza systém cez rôzne uzly v rôznych procesoch, môže v priebehu procesov dôjsť k odpojeniu užívateľa. Proces 1 odpojeného užívateľa monitoroval v čase t a prehlásil ho za aktívny. Proces 2, ktorý monitoroval rovnakého užívateľa v čase $t+x$ ten istý uzol prehlásil za odpojený (nestihol ho počas krátkeho časového úseku nájsť/skontaktovať). Takto vzniknuté 2 množiny monitorovaných uzlov ponúkajú rôzne pohľady na systém, kde sa rozhoduje, ktorý z týchto snímkov je korektný (prechod peera bol zo stavu neaktívny na aktívny alebo naopak). Ak sa zopakuje tento prechod, je možné dostať viac rovnakých množín než tých, v ktorých sa identifikátory nachádzaju v inom poradí alebo nenachádzajú vôbec. Následne sa prehlásia monitorovania, kde sa početne opakovali peerovia, za úspešné. Peerov, ktorých program našiel iba raz je možné označiť ako nejasných. Všetko rozhodovanie je relevantné, ak existuje predpoklad, že sa monitorovala tak veľká sieť tak rýchlo, že sa zachovala konzistencia snímky.

3.2 Detekovacie metódy

Prehľadávač, ktorý implementuje dané detekčné metódy sa označuje ako *crawler*. Najviac efektívne metódy sú rozdelené podľa toho, akým klientom (v akej sieti) komunikujú jednotlivý peeri. V každej z týchto metód zmienené v podkapitolách 3.3, 3.4, 3.5 je zohľadnené, to čo je pre daný systém najlepším možným riešením.

Všetky existujúce detekčné metódy sú založené na štarte z k-bitovej zóny. Všetky uzly v tejto zóne zdieľajú prefix o veľkosti k bitov v ich identifikátoroch. Keďže je ťažké zistiť, ktoré zariadenie má akú IP adresu, tak väčšina metód pre monitorovanie začína na už existujúcom verejne prístupnom zariadení. Z tohto zariadenia sa monitoruje smerom k cieľovým identifikátorom. Pre monitorovanie sa vychádza z predpokladu, že infoheš je možné dostať z .torrent súboru alebo magnet-odkazu. Z týchto prístupných miest sa dokáže pomocou správy find_node dostať datagram do cieľa. Cieľ pošle odpoveď, ktorá v sekcií r obsahuje kľúč *nodes*, v ktorom sú uložené uzly v tvare `infoheš, IP adresa:port`. Postupným zväčšovaním hodnoty k sa zmenšuje daná zónu peerov. Ak sa hodnota k zmenší, zóna peerov sa zväčší. Experimentálnym spôsobom znižovania a zvyšovania parametru k je možné dôjsť pri prehľadávaní k výsledku, akým veľkým k by prehľadávanie malo započať. Použitím vhodne veľkej zvolenej zóny sa dokáže urobiť lepší odhad veľkosti distribuovanej siete. Z týchto údajov je zistená jedna z podstatných informácií a to odvodenie chyby v konkrétnej metóde.

Z teórie prehľadávania stavového priestoru poznáme dve stratégie pri prehľadávaní. `Prehľadávanie do šírky` (BFS) a `prehľadávanie do hĺbky` (DFS). Pri prvej stratégií sú cieľové ID vybrané zo systému rovnomerne. Kontakty v každej odpovedi sa môžu distribuovať do ďalších zón. Nastáva zber peerov rovnomerne z celého ID priestoru. Pre implementáciu prehľadávania do šírky sa používa FIFO fronta.

Pre DFS sú kontakty na peerov v každej odpovedi väčšinou distribuované v rovnakej zóne ako je cieľ (nerovnomerne). Takto sa rýchlo dá dostať k danému infohešu. Problémom metódy je vyčerpanie prehľadávania. Preto je nutné metódu DFS upraviť tak, aby bolo možné od cieľového uzlu vychádzať z DHT priestoru naspäť k indetifikátorom s horšou vzdialenosťou ako cieľový infoheš. Modifikácia DFS je vykonaná kvôli faktu, že peerov nevracia iba cieľový infoheš ale aj infoheše blízko cieľa. Implementácia DFS spočíva v používaní LIFO fronty.

Výsledkom detekčných metód je množina peerov a uzlov, ktoré peerov v odpovediach správy get_peers dodali.

3.3 Metóda pre detekciu peerov v systéme KAD

Na začiatku prehľadávania k-bitovej zóny, sa zhromažďujú iba peerovia, ktorí majú rovnaký KAD ID prefix. Napríklad prehľadávač Blizzard [20] potrebuje iba 2.5 sekundy na kompletné prehľadanie 8-bitovej zóny. Nie je možné však získať úplnú snímku, keďže prehľadávanie 8-bitovej zóny má informácie iba o 256-tich peeroch. Na druhú stranu kompletné prehliadanie zóny podlieha víru. Pre náhľad, aké algoritmy je potrebné implementovať v KAD sieti, je potrebné implementovať časť správy DHT. Táto správa je zhrnutá v prehliadavači Rainbow [17] alebo Blizzard. Algoritmus 1 a 2 je inšpirovaný prehľadávačom Rainbow. Z pôvodného algoritmu sú vynechané časti správy zdieľanej mapy.

1. *mpeers*: zdieľaná mapa medzi vláknami zložená z dvojice <infoheš, peer>.
2. `UDP_Responded` status uzlu, ktorý už odpovedal a je uložený v log súbore.

Algoritmus 1: Vlákno send UDP

Vstup : k Počiatočných peerov z lokálnej smerovacej tabuľky, log súbor posledného prehliadania, status UDP_Responded, zdieľaná mapa $mpeers$, n je maximálna veľkosť mapy(pre 8 bitovú zónu n=256)

Výstup: aktualizovaná množina *qpeersUDP*

```
1  Inicializácia qpeersUDP s k počiatočnými peermi;
2  prečítaj tých peerov, ktorých status = UDP_Responded z posledného log súboru do
     množiny qpeersUDP{pozitívna spätná väzba};
3  while size(mpeers) ≤ n do
4      repeat
5          pošli UDP správu typu dopyt prvému elementu p z qpeersUDP množiny;
6          vymaž p z qpeersUDP;
7          if p ∈ mpeers then
8              p.status ← UDP_Requested
9          end
10     until size(qpeersUDP) == 0;
11 end
```

Algoritmus 2: Vlákno recieve UDP

Vstup : UDP správa typu odpoveď (UDP_response_message), časový interval T_i

Výstup: aktualizovaná množina *qpeersUDP* a *mpeers*

```
1  while true do
2      čakaj na UDP správu typu odpoveď;
3      if wait(UDP_response_message) > T_i then
4          vezmi vzorku peerov, ktorý neodpovedali z množiny mpeers a pošli UDP
             správu typu dopyt k nim {neaktívny mechanizmus};
5      end
6      forall p ∈ UDP_response_message do
7          qelem ← p.info, key ← p.info, peer ← p.info;
8          peer.status ← UDP_Responded;
9          if key ∉ mpeers then
10             mpeers.add(<key,peer>);
11             qpeersUDP.add(qelem);
12         end
13     end
14 end
```

Algorimy obsahujú dva mechanizmy

1. Mechanizmus pozitívnej spätnej väzby. Spočíva v čitaní správ odpovedajúcich peerov z najnovšieho prehliadania do množiny *qpeersUDP*, aby urýchlil prehliadanie.

2. Stimulačný mechanizmus. Zabraňuje náhodnému spomaleniu alebo pozastaveniu (vyčerpaniu) prehliadania. Ak *recieve UDP* vlákno nedostane správu včas, z odpovedí UDP sa vyberú vzorky peerov, ktorý neodpovedali a posiela im opätovne správy typu dopyt UDP.

Tieto dva mechanizmy zabezpečia robustnosť prehľadávania v sieti KAD.

3.4 Metóda pre detekciu peerov v systéme Vuze

Vyhľadanie pre sieť klientov Vuze by malo byť paralelne a cyklické. ID v DHT typu Vuze označme ako T. Uvedieme algoritmus 4 pre vyhľadávanie uzlov v systéme vuze a algoritmus 3 ako implementáciu funkcie triedenia uzlov pomocou XOR metriky. Tento algoritmus nie je obohatený o koordinačný systém Vivaldi, ktorý implementuje vuze klient.

Algoritmus 3: Vuze triedenie pomocou XOR a Vivaldi

```
1  Function List::sortVuze:
2      for closest=1 to 20 do
3          vzdialenosť = vypočítaj vzdialenosť medzi T a peerSetVuze[closest] pomocou
            XOR
4          if vzdialenosť ≤ peerSetVuze.min then
5              /* usporiadaj množinu tak aby bol najbližší prvok na prvom mieste.
                peerSetVuze[closest] má menšiu alebo rovnú vzdialenosť */
6              peerSetVuze.sort()
7              process.add(peerSetVuze[closest])
8          end
9      end
10     for closest=1 to 10 do
11         peerSetVuze.vivaldi.sort()
12     end
13
```

Algoritmus 4: Vuze smerovací algoritmus, pre prehľadávanie uzlov

Vstup : Zoznam *peerSetVuze*, ktorý obsahuje uzly, Vuze ID T, limit Exceeded
Výstup: Uzol s ID T alebo nastala chyba a uzol sa nenašiel

```
1  Function routingVuze:
2      peerSetVuze.sortVuze;
3      // Zašli 5 dopytov v rámci zasielacieho vlákna a zoraď podľa XOR metriky
4      sendQuery.send(5)
5      while Time ≤ Exceeded or numberOfMessages ≠ 0 or T ≠ process.first.id do
6          if recieveQuery.get() then
7              process.add(recieveQuery.get())
8              process.sortVuze
9          else
10             mutex_lock()
11             if numberOfMessages < 5 then
12                 sendQuery(1)
13             end
14             mutex_unlock()
15         end
16     end
17     if T = peerSetVuze.first.id then
18         return T
19     else
20         return Null
21     end
```

3.5 Metóda pre detekciu peerov v systéme MLDHT

Metóda zmienená od pána Wang Liang [16], sa spoliehaja na to, že identifikátory sú rovnomerne rozložené v Kademlia vedrách. Principiálne MLDHT protokol by mal garantovať rovnomerné rozloženie ID uzlov [16]. Podstatou metódy pre detekciu peerov je správne zvolenie zóny, v ktorej budeme detegovať peerov.

Po výbere náhodného identifikátoru zvoleného uzlu, sa spustí procedúra na zbieranie všetkých blízkych uzlov využitím operácie `FIND_NODE`. Odporúča sa použiť napr. FIFO frontu a algoritmus BFS cyklicky (ale aj iné typy front), keďže pri použití tejto metódy by mali byť uzly rozložené rovnomerne. Smerovacie informácie o uzloch by sa mali ustáliť približne každých 15 minút[5], ale to záleží na implementácii klientskej aplikácie.

Pri monitorovaní týchto sietí, je cieľom dostať všetkých peerov zdieľajúcich torrenty. Takéto monitorovanie má tiež svoju nevýhodu a to je problém *missing node-issue* v každom prechode prehľadávača. Ak je zvolená veľmi malá zóna (veľké k), mierne kolísanie spôsobené chýbajúcimi uzlami sa po zväčšení zóny zvýši. Zóna sa zväčšuje, kvôli získaniu čo najväčšieho počtu peerov. Toto zväčšenie vedie k významnému (a náhodnému) rozdielu v počte peerov (oproti variante s nižším k). Obrovské kolísanie môže dokonca vyčerpať detekciu. Je to v dôsledku veľkej sady rovnakých vzoriek. Ďalšie nežiadúce účinky na kolísanie môžu byť firewally, preťaženie liniek, abnormálne správanie peerov, užívateľské stanice nachádzajúce sa za NAT-om a pod.

Pre systém MLDHT nebol navrhnutý žiaden algoritmus. Implementácia neobsahuje koordinačný systém takže algoritmus bude veľmi podobný predošlému algoritmu z KAD. V praxi je rozdiel medzi KAD a MLDHT iba v infoheši. Implementácia obsahuje všetky aspekty z Kademlie a z oficiálnej špecifikácie [5].

3.5.1 Súhrn MLDHT

Pre určenie presnosti prehľadávania je potreba vyhnúť problému s *missing node-issue* [16] . Tento termín zahŕňa tieto aspekty:

- Pohybovanie siete: príchod a odchod uzlov. Tento problém sa sťažuje s dĺžkou prehliadania.
- Strata správ kvôli preťaženiu siete.
- Rôzne protekčné mechanizmy: napr. čierna listina, zákaz komunikácie podozrivých uzlov, zahadzovanie deformovaných (malformed) správ, malá veľkosť vedra, malá veľkosť fronty.
- Neaktuálne informácie v smerovacej tabuľke, hlavne kvôli implementačným detailom Kademlia.
- Prehľadávač nie je dosť efektívny z hľadiska rýchlosti a výberovosti uzlov (prehliadač sa obmedzí aby si nebral uzly, ktoré vyzerajú podozrivo). To sa zaistí tak, že sa je snaha prijímať iba odpovede zo zaslaných dopytov a cudzie sa zahadzujú.
- Problémy s firewallom.

3.6 Zhrnutie existujúcich metód pre monitorovanie BitTorrent

Monitorovanie veľkých sietí je `veľmi náročné`. Dôvodom je, že zóna je veľká a je potrebné dostať čo najviac uzlov. Čas detekcie, ako aj prevádzkové náklady na detekciu sa zvyšujú exponenciálne. Zároveň je veľa uzlov nenájdených, pretože výkon prehľadávača je obmedzený. Počas prehľadávania veľkej zóny je možné ovplyvniť výsledky detekcie peerov, čo vedie k ťažko odhaliteľným chybám [16].

Dôvodom neefektívneho výkonu metód je výsledok obranných mechanizmov v dnešných aplikáciách. Sú nimi:

1. Zákaz podozrivých peerov
2. Filtrovanie nesprávne vytvorených správ

Tieto obranné mechanizmy môžu vážne zhoršiť výkonnosť a presnosť prehliadania. Vhodná veľkosť fronty, vhodná veľkosť vedier, komplikované operácie smerovacej tabuľky a funkcie spätného volania (callback) zvyšujú efektívnosť vyhľadávania.

Pre vyhľadávanie je najpoužívanejšou metódou získanie množiny uzlov z určitej zóny pomocou DHT. Rýchlosť prehľadávania je najdôležitejším faktorom vzhľadom na konvergenciu presnosti prehľadávania daného systému. V distribuovanom systéme s vysokou mierou *vírenia* uzlov, určuje kvalitu snímky to, `ako rýchlo dokážeme dostať danú snímku`.

Z tabuľky 3.1 je vidieť prehľad dvoch metód DHT pre monitorovanie BitTorrent. Pre trvalé DHT je zástupcom systém KAD 3.3. Pre dynamické ID to je systém MLDHT a Vuze.

Metódy	Výhody	Nevýhody	Prebrané z
Čiastočné ID(DHT)	1. Možnosť rýchleho a komplexného prehľadávania,	1. Štúdium komunikácie, získanie konzistentného pohľadu	[20], [14]
	2. Pohľad na úrovni systému	2. Náročnosť monitorovania obsahu	
Trvalé ID(DHT)	Možnosť rýchleho prehľadávania. Pohľad na úrovni systému	Náročnosť monitorovania obsahu, získanie konzistentného pohľadu	[15]

Tabuľka 3.1: Zhrnutie metód

Kapitola 4

Návrh monitorovania peerov zdieľajúcich torrent

Pre monitorovanie peerov som po dohode s vedúcim práce vybral systém MLDHT (a jemu odvodené systémy). Jeho popularita a už sprostredkovaný výskum ponúkol možnosť monitorovať efektívne napriek nepriaznivým javom. Ak sa skombinuje Bernoulliho proces čo bolo doporučené v podkapitole o MLDHT 3.5, je možné zistiť ako presne sa monitoruje. Cieľom je v tejto kapitole načrtnúť vstupy a výstupy programu, jeho architektúru a testovaciu sadu. Taktiež je kladený dôraz na prenositeľnosť programu na rôzne platformy (táto záležitosť sa pravdepodobne rieši buď virtualizáciou alebo kontajnerizáciou [9]). Na túto implementáciu využijem jazyk python, ktorý má knižnice pre zostavovanie správ protokolu Kademlia. XOR metrika sa využíva iba pri uzloch, ktoré sú veľmi blízko vyhľadávanému infohešu. Akonáhle sa prehľadáva pri cieli podstatne blízko je vyššia pravdepodobnosť, že väčšina peerov sa bude nachádzať pri tomto uzle.

4.1 Vstupy a výstupy programu

Program je navrhnutý tak, aby prijímal rôzne vstupy a možné upravenie konfigurácie správ. Ponúka tak čo najlepšiu variantnosť monitorovania. Upravením konfigurácie je myslené napr. prehľadávanie do hĺbky, šírky, zmena výstupného formátu a pod. Monitorovacím vstupom pre prehľadávanie peerov je infoheš. Ten môžeme do programu dostať viacerými spôsobmi.

1. Vstupom môže byť 20 bajtové hexa číslo s infohešom torrentu. (40 číslic).
2. Magnet-odkaz, z ktorého sa vyextrahuje infoheš za značkou *urn:btih:*.
3. Torrent súbor, ktorý obsahuje meta informácie o torrente. Z tohto súboru je možné taktiež po zadaní ďalších dodatočných konfigurácií zobraziť názov torrentu a pod.
4. Ak na vstupe infoheš nie je, program si vygeneruje vlastný.

Pre overenie implementácie je žiadúce začínať z verejne prístupného uzla. Preto neexistuje varianta so začatím monitorovania z konkrétneho uzlu. Pri globálnom prehľadávaní sú odporúčané tracker servery ako napr. router.bittorrent.com a iné. Tieto servery udržujú informácie o rôznych uzloch v DHT. V prípade prehľadávania lokálneho priestoru je potrebné zaistiť, aby vstupný uzol vedel o ostatných a plnil rovnakú funkciu ako tieto centralizované

servery. To vie takýto klient zaistiť, ak implementuje protokol systému MLDHT, je práve pripojený na daný port a ak si ukladá svoju smerovaciu tabuľku.

Výstupom programu je množina uzlov, ktoré program našiel. Ak použijeme konkrétny torrent súbor môžu sa na začiatku monitorovania zobraziť dodatočné údaje ako názov torrentu, počet súborov, a zoznam aktuálnych uzlov, ktoré tento súbor ponúkajú. Pre účely rozšírenia práce a kompatibility s databázou sú pridané 2 výstupné formáty.

1. Databázový formát. Tento formát sa určil pre kompatibilitu s databázou, ktorá sa riešila v rámci diplomovej práce. Formát je zobrazený nižšie na schéme 3.

2. Formát v tvare: počet nájdených uzlov, počet peerov, % úspešnosť Bernoulliho procesu a aktuálna dĺžka fronty.

Schéma 3 Výstupný databázový formát.

{„infohash”:
{„peers”: [
{„timestamp”: timestamp_value, „addr”: [peer_ip_addr, peer_port]},
...
]
„nodes”: [
{„timestamp”: timestamp_value, „nodeID”: nodeID_which_respond_with_peers, „node-Addr”: [node_ip_addr, node_port]},
...
]
„name”: torrent_name
}
}

4.2 Architektúra

Princíp prehliadania spočíva v tom, že na začiatku sa stanoví odozvu, počas ktorej sa generujú konzistentné snímky aktuálneho systému. Snímky sú vytvárené pomocou 2 metód (funkcií). Vysielač prehľadávania, prijímač správ z prehliadania. Správy get_peers sa používajú kvôli zisteniu peerov a uzlov pre konkrétny infoheš. Ak uzol obdrží správu get_peers a pozná k tomuto infohešu peerov, vráti ich ako odpoveď. V opačnom prípade uzol vracia uzly, najbližšie hľadanému infohešu. Obsah tejto odpovede sa rozbalí a ak obsahuje uzly, naplní sa fronta FQ, v opačnom prípade metóda našla peerov a pridá ich do množiny. Základný algoritmus vyzerá nasledovne (Algoritmus 5).

Algoritmus 5: Vlákno Vysielač správ

Vstup : Nainicializovaný objekt triedy prehliadač (self), zdieľaná fronta FQ, infoheš id, metrika, port, IP adresa, odozva T_i

Výstup: Aktualizovaná zdieľaná fronta uzlov

```
1  while self.aktuálna_odozva ≤ T_i do
2      uzol = vyber uzol z fronty FQ;
3      if (self.id ⊕ uzol.id >> 148) == 0 then
4          //zašli dopyt typu get_peers na uzol["host"], uzol["port"] s infohešom
             uzol["id"];
5          for count = 1 to 5 do
6              get_peers(uzol["host"], uzol["port"], uzol["id"]);
7          end
8      end
9      else if self.veľkosť_zóny < zadaná_veľkosť then
10         zašli dopyt typu get_peers na uzol["host"], uzol["port"], self.id;
11     end
12 end
```

Algoritmus 6: Vlákno Prijímač správ

Vstup : Inicializovaný objekt triedy prehliadač, správa typu odpoveď, uzol_pool, peer_pool

Výstup: Aktualizovaný pool uzlov a peerov, aktualizovaná zdieľaná fronta FQ

```
1  while odozva ≤ T_i do
2      správa, adresa = self.recieve_message();
3      id, typ, obsah = self.krpc.decode(správa);
4      if "nodes" ∈ obsah then
5          uzly = obsah.rozbal_uzly() ;
6          uzol_pool.add(uzly) ;
7          self.respondent += 1;
8      end
9      if "peers" ∈ obsah then
10         // obsah má vnútri tela peerov. Rozbaľ odpoveď;
11         peers = obsah.rozbal_uzly_peerov() ;
12         peer_pool.add(peers) ;
13     end
14     // Kvôli filtrácií vyplníme adresy uzlov, ktoré sme už skontaktovali;
15     self.Pool[adresa,port] = označ adresu časovou značkou;
16 end
```

Toto je najmenšia možná metóda na prehliadanie pomocou vlákien so zdieľanou frontou. Neobsahuje žiadne protekčné mechanizmy a má XOR metriku. Cieľom návrhu je rozšíriť možnosti prehliadania o abstrahovanie rôznych informácií z IP adresy. Abstrakcia informácií sa v rámci tejto bakalárskej práce implementuje zároveň s prehľadávačom.

Podpornou metódou je meranie. Meranie určuje presnosť prehliadania, teda výpočet pravdepodobnosti p. Prehliadanie je riadené Bernoulliho procesom. Zo získaných výsledkov z vláken prijímania a odosielania správ sa počíta pravdepodobnosť p.

4.3 Testovanie a testovacia sada

Testovanie bude prebiehať v 3 fázach.

1. Statická analýza kódu a jednoduché testy funkčnosti modulov (jednotkové testy a pylint).

2. Globálne testovanie v rámci distribuovanej siete, z rôznych trackerov a cez rôzne magnet-odkazy a torrent súbory.

Statická analýza nám zistí či je program dobre navrhnutý a je možné ho používať. Moduly budú testované samostatne, pomocou správne navrhnutej testovacej sady na zistenie konvergencie monitorovania. Po tejto fáze budú moduly postupne spájané do celku. Kontajnerom sa simululuje beh na iných distribúciách aby bolo dokázané, či sa program chová podľa predpokladov na rôznych systémoch. Statická analýza bude sprostredkovaná pomocou nástroja pylint a kód bude upravený štýlom PEP8 [25]. Cieľom je dostať hodnotenie viac ako 8/10 nad každým zdrojovým súborom.

Testovanie vrámci *distribuovanej siete* (globálne) pozostáva z troch úloh. Akých peerov pre zadané súbory najčastejšie uzly ponúkajú, ako efektívne je program schopný monitorovať uzly a peerov a *missing-node* efekt spomenutý v podkapitole 3.5.1. Budem sa snažiť aspoň týždňovým prehliadaním docieliť žiadaných výsledkov.

Kapitola 5

Implementácia

Kapitola je rozdelená do 6 základných častí. V prvej podkapitole sa zameriava na triedny návrh a budú popísané interakcie medzi triedami. Ďalším míľnikom je priblíženie implementácie vláken. V predposlednej podkapitole je zahrnutá úspešná integrácia nástroja s diplomovou prácou. Posledná podkapitola obsahuje zhrnutie výhod i nevýhod takejto implementácie. V zložke *dht_crawler/* nájdeme zdrojové kódy a spustiteľný program *exec.py* v jazyku python pre verziu 3 a vyššiu. Dôveryhodnosť a sofistikovanosť riešena implementácie je zaistené tak, že máme na vstupe dostatočne veľkú vzorku nových a starších torrentov.

5.1 Interakcia tried a triedny návrh

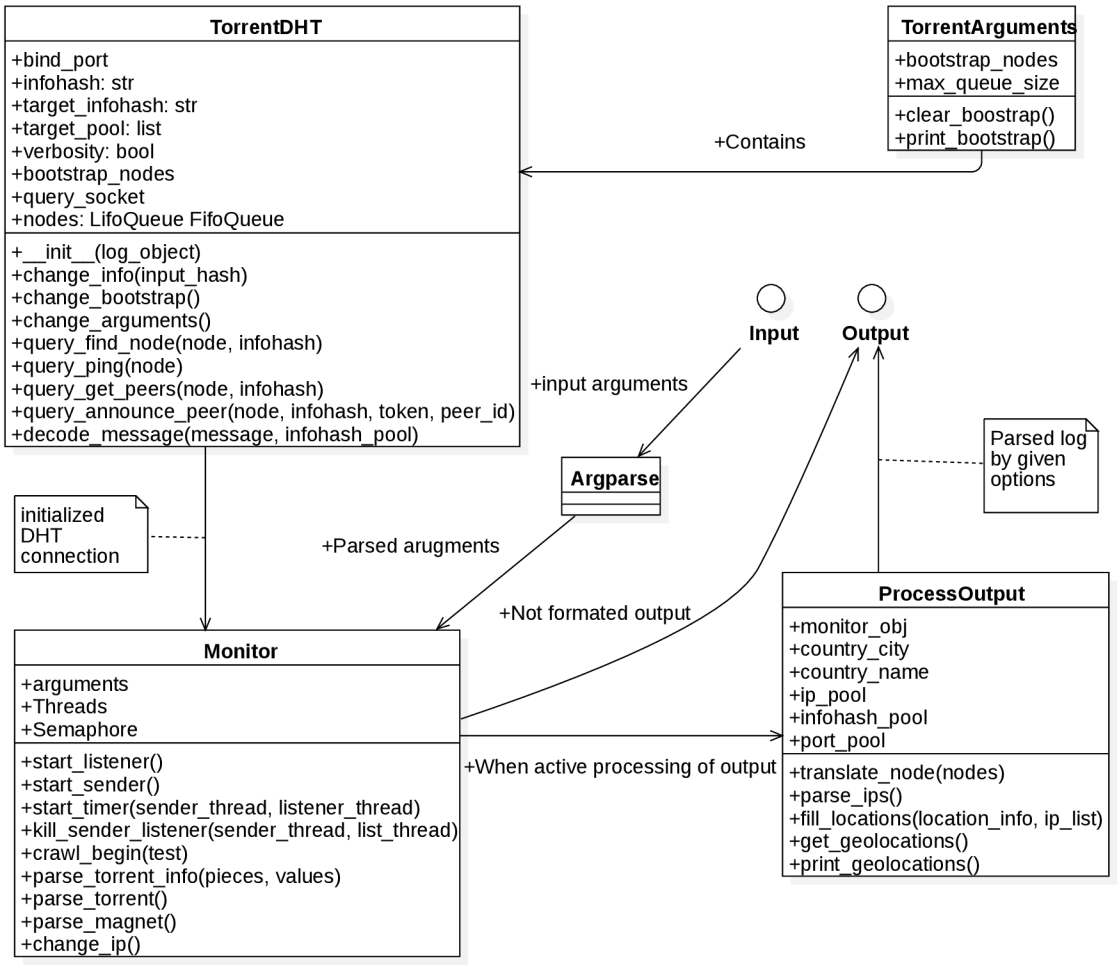
Triedny návrh je podstatnou súčasťou implementácie. Triedy boli rozdelené podľa toho s ktorými inštanciami monitorovania interagovali (vstup, DHT, výstup atď.). Výsledok návrhu je vidieť na obrázku 5.1. Trieda Argparse je vstavaná trieda jazyka, ktorá sa stará o analýzu vstupných argumentov z príkazového riadku. To vstupuje ako spracovaný vstup do triedy Monitor. Táto trieda riadi celý beh prehliadania. Je zložená zo správy TorrentDHT a ProcessOutput.

TorrentDHT sa stará o vytvorenie schránky UDP a podporu všetkých MLDHT správ pre IPv4 verziu. Inicializuje sa pomocou triedy TorrentArguments, ktorá obsahuje uzly, ktorých sa máme dopytovať ako prvých. Taktiež je tam definovaná dĺžka front, ktorá sa môže menit aj parametrom zo vstupu. Dôležité atribúty TorrentDHT triedy sú target_infohash, target_pool, nodes a query_socket, ktorý sa spája s parametrom bind_port. Na zadaný bind_port je schránka pripojená na prijímanie správ. Nodes je zdieľaná fronta medzi vláknami prijímania a odosielania pre inštanciu triedy monitor.

Trieda ProcessOutput sa stará o stav, keď žiadame výsledné dáta z monitorovania formátovať. Formát odpovedá výstupu ako je uvedené v podkapitole 4.1. Výstup v databázovom formáte špecifikujeme pomocou prepínača *--db_format*. Nešpecifikovaním formátu dostávame na výstup druhý formát z podkapitoly 4.1. Je možne použiť aj tretí formát, ktorý na výstupe vypisuje konkrétne informácie o peeroch a preklad na geolokáciu. Ten sa špecifikuje pomocou *--print_as_country* alebo sa realizuje pomocou externého nástroja.

5.1.1 Spustenie programu

Exec.py spúšťa najprv analýzu vstupných argumentov. Vytvorí na základe argumentov príslušnú DHT schránku. Túto triedu vloží aj s argumentmi do objektu monitoru, kde sa nainicializujú všetky atribúty, ktoré korigujú monitorovanie. Pred spustením je možné, že na



Obr. 5.1: Diagram tried.

vstupe je torrent súbor alebo magnet-odkaz. Dodatočne sa zavolajú metódy parse_torrent a parse_magnet, pretože je podpora zadania viac súborov a kombinácia týchto parametrov. Posledným krokom je spustenie prehľadávania pomocou metódy start_sender. Vytvorí príslušné vlákna na počítanie času, odosielanie a prijímanie správ.

5.2 Implementácia vláken a spracovania argumentov

Skript má na vstupe delič argumentov, ktorým sa prehľadávanie koriguje, skracuje, prípadne predlžuje a je možné nastaviť dĺžku odozvy. To docielime argumentmi:

- *--duration hodnota*. Tento argument určuje dĺžku monitorovania siete. Bez zadania arugmentu je dĺžka monitorovania 600 sekúnd.
- *--counter hodnota*. Špecifikujeme ním dĺžku odozvy medzi zasielaním správ. Bez zadania arugmentu je nulová odozva.

Jadro programu sa upiera na metódu *crawl_begin*. Táto metóda má ako parameter ľubovoľný počet torrent súborov, magnet-odkazov, ktorý chceme monitorovať. Monitorovanie

pomocou infohešu je možné iba jedno. Na predaný torrent súbor, magnet-odkaz alebo infoheš sa vytvorí vlákno pre zasielanie a prijímanie odpovedí. Vlákno *duration_thread* nám meria čas. Ak toto vlákno prekročí stanovený čas, tak končí prehľadávanie, vlákna sa zabijú, uvoľnia prostriedky a volá sa spracovanie výsledkov.

5.2.1 Implementácia prijímania a odosielania v rámci vláken

Pre odosielanie a prijímanie boli zaviedené metódy *sender* a *reciever*. Tieto dve metódy fungujú ako vlákna. Zdieľajú medzi sebou jednu frontu, z ktorej odosielacie vlákno (sender) prijíma nové uzly. Prijímacie vlákno (reciever) do tejto fronty pridáva nové uzly. Tento princíp zdieľania sa nazýva producent-konzument [12].

Odosielateľ vyberie zo zdieľanej fronty uzol, porovná infoheš uzlu s cieľovým infohešom pomocou XOR metriky.

- Ak je uzol blízko roja, zasiela 5 správ.
- Ak je uzol ďaleko, zasiela 1 správu.

Pre Prijímač, pri použití nespojovanej služby UDP, je funkcia recvfrom blokujúca. Je nutné túto blokáciu ošetriť tak, že je nastavená odozva na schránku. Problémom je, že datagram môže po prekročení tejto odozvy prísť na sieťové rozhranie ale program to už nespozná. Môže nastať situácia, že správy nedôjdu k cieľu a tým pádom daný datagram je stratený. Problém straty datagramov je obsiahnutý v kapitole 6. Po prijatí datagramu je nutné ho dekódovať a dekódovať všetky jeho zložky podľa typu odpovede. Následne sú aktualizované fronty uzlov a peerov a aj slovníky s uzlami a peerami.

5.3 Implementácia KRPC správ

Trieda TorrentArguments obsahuje argumenty ako je dĺžka fronty a počiatočné uzly, z ktorých sa prehľadávanie začína (zasielanie KRPC správ). Zvolil som 3 konkrétne počiatočné uzly. Sú nimi router.bittorent.
com, dht.transmissionbt.com a router.utorrent.com. Na tieto servery môže byť poslaný ľubovoľný dotaz KRPC (v našom prípade get_peers). Vyplnená get_peers správá je pred odoslaním transformovaná pomocou bencode. Pri prijímaní správ používa táto trieda metódu *decode_message*, ktorou sa filtruje obsah odpovede a na základe tohto filtra sa zavolá príslušná dekódovacia funkcia. Ako výsledok metódy sa vrátia nové prvky, ktoré má program prehľadávať alebo konkrétny peerovia, ku ktorým je možné pripojenie ak je záujem o sťahovanie obsahu.

5.4 Spracovanie adries po prehľadávaní

Výsledkom z monitorovania je množina uzlov a peerov. Množiny je potrebné pred ukončením monitorovania skontrolovať, či peerovia/uzly neprekročili časovú značku. Ak je prekročená značka o 10 minút, peer je vyhodený z množiny, inak ostáva v peer množine. Tento spôsob je veľmi rýchly no nie je úplne korektný z hľadiska konektivity peerov. Preto je potrebné brať údaj o peeroch ako odhad s určitou odchýlkou, ktorou sa zaoberalo v kapitole 6. Je jednoduchšie nechať na výstupe väčšiu vzorku peerov pre torrent než získať presnú snímku, pretože je tam striktná časová záležitosť medzi získaním konzistentnej snímky

a jej úpravou. Úpravu množín budem ďalej prezentovať aj ako *preriedenie slovníka* alebo *preriedenie množiny*.

Pre lepšiu prezentáciu výsledkov práce som skonštruoval nástroj, ktorý má za úlohu zo získaných IP adries z monitorovania dostať informácie, ktoré sú voľne dostupné. Sú nimi whois služba na zistenie, kde bola daná IP adresa registrovaná, aká spoločnosť je registrátorom a pod. Reverzná DNS rezolúcia je ďalšou informáciou, ktorú je možné získať z IP adresy. Doménové meno získane z IP adresy často chýba. Tento DNS preklad je preto cennou súčasťou a ponúka lepší pohľad na vývoj siete ako celku. To celé je zaobalené časovou značkou aby sme vedeli, ku ktorému dátumu s presným časom je tento údaj o IP preklade platný. V poslednej časti sa vykonáva preklad na geolokáciu. Preklad na geolokáciu poskytuje údaje o zemepisnej šírke a dĺžke. V dôsledku tohto údaju máme zhruba pojem o tom, kde sa takýto uzol/peera nachádza. Celý tento nástroj je implementovaný v samostatnom programe ip_detail[1].

5.5 Integrovanie nástroja do diplomovej práce

Nástroj bol úspešne integrovaný a otestovaný do diplomovej práce. Bolo preto potrebné upraviť výstupný formát a v integračných testoch zahrnúť test s vytvorením relevantného výstupu. Pomocou zadania argumentu `--db_format` a pustením klienta z diplomovej práce, ktorý súbor s výsledkom monitorovania predal na server, bolo prevedené integrovanie. Skontaktovaný server spracoval nájdené uzly pre daný infoheš do databáze.

5.6 Výhody a nevýhody implementácie aktívneho monitorovania

Výhodou implementácie je fakt, že trackery a rôzne prvky v BitTorrent sieti udržiavajú logovacie súbory o tom koľko peerov sťahovalo daný torrent. Logy trackerov sú aktuálne iba pre určitý časový okamih a po prekročení 15 minút sú irelevantné. Vyplýva to zo špecifikácie DHT [5], kde sa uzol v smerovacej tabuľke trackeru/uzlu musí po prekročení časovej značky 15 minút vyhodiť. Preto sa tieto logovacie súbory musia často aktualizovať. Implementáciou monitorovania máme zaistený rovnaký pohľad na sťahovanie, dokonca aj lepší. Lepší kvôli tomu, že tracker dokáže v rámci logu získať iba tých peerov, ktorý ho skontaktovali. Na jeden torrent je však bežne sprevádzkovaných 5 až 10 trackerov, kde sa informácie duplikujú.

Problémom implementácie je pomerne veľká záťaž prenosového pásma. Záťaž je hlavne z dôvodu neustáleho dopytovania sa, prijímania odpovede a rozčleňovania sekcií v odpovedi. Procesor je zaťažený touto réžiou po celu dobu behu programu.

Kvôli záťaži rozčleňovania sekcií je nástroj odľahčený od akejkoľvek kontroly čiernej listiny (blacklistu), filtrovanie deformovaných správ a podobných mechaník. Vďaka tomu sa síce počet peerov zväčšil no zavádza to menej bezpečné monitorovanie.

Problémom, ktorý sa v rámci distrubovanej siete BitTorrent nedá vyhnúť je vírivý efekt. Neustále pripájanie a odpájanie uzlov sťažuje odhad dĺžky prehľadávania. Ak nástroj pustíme 100 krát a každý z týchto spustení bude prehľadávať o sekundu dlhšie ako predošlé spustenie, tak počet peerov je kolísavý. Čím novší torrent súbor máme na vstupe, tým je fluktuácia vyššia.

[1] https://github.com/Matovidlo/ip_detail

Kapitola 6

Testovanie, experimenty a získané výsledky

Počas testovania sa začalo sprevádzkovaním jednotkových testov, kde som otestoval vytvorenie tried, vlákien a zasielanie správ. Bolo potrebné otestovať analyzátor magnet-odkazu a torrent súboru. Dôvod testovania tohto analyzátora je v tom, že štandardy sa upravujú a menia sa typy súborov resp. ich štruktúra. Preto je pravdepodobné, že životný cyklus takéhoto nástroja nemusí byť dlhý. Preto bolo vhodné urobiť taktiež testovanie nástroja a jeho balíčkov ako takých a testovanie úspešnosti správ.

V kapitole sa nachádza 5 typov experimentov. Tieto experimenty overili implementáciu DHT a funkčnosť týchto tabuliek. V závere je zhrnutie všetkých testov a experimentov.

6.1 Jednotkové testy

Pre účely práce som otestoval triedne metódy pomocou `jednotkových testov`. Testov bolo 6, kde sa skúšali analyzátory vstupov, vytvorenie objektov z tried, metód pre prehľadávanie priestoru, a testovanie spojenia. Tieto testy sa spúšťajú automaticky pri spustení skriptom a spustením zostavovacieho systému na stránke github. Týmto zostavovacím systémom sa udržuje aktualizovanosť repozitára (nástroja). Preto je do repozitára zahrnutý zostavovací systém *travis*[1], ktorý overuje, že skript (program) je stále aktuálny a jednotkové testy funkčné.

6.2 Experimenty

Meranie prebiehalo nad rôznymi torrent súbormi, z ktorých sme vytiahli informácie iba o infoheši. Chceli sme zistiť nakoľko dobre funguje DHT, ktorá nás má postupne previesť celým priestorom k správnemu infohešu. Monitorovanie bolo spustené dopytom typu get_peers na verejne prístupný uzol dht.transmissionbt.com.

Prvým experimentom bolo overenie funkcie DHT. Overenie DHT bolo navrhnuté tak, že sa otvoril torrent súbor s väčším počtom peerov (v tom čase 3367) a spustil prehliadanie. Výsledkom sú množiny uzlov a peerov v určitom časovom horizonte. Tým danú implementáciu overím a budem sa môcť venovať ďalším experimentom, ktorými sú:

[1] https://docs.travis-ci.com/user/for-beginners/

- Overenie funkcie DHT. Rozdiel pri použití LIFO a FIFO fronty. Určenie dôsledkov použitia týchto font a preukázanie validity.
- Odhad chybovosti prehľadávania.
- Percentuálny podiel medzi uzlami a peermi. Preukázanie validity FIFO a LIFO
- Porovnanie efektívnej dĺžky prehľadávania medzi populárnym a nepopulárnym torrentom.
- Použitie Bernoulliho procesu.
- Počet nezachytených správ a pomer aktívneho monitorovania ku čakacej dobe na správy.
- Experimentálny odhad efektívnej dĺžky prehľadávania. Prečo sa presný odhad určiť nedá a prečo je aktívne monitorovanie tak výhodné.

Overenie funkcie DHT, rozdiel pri použití LIFO a FIFO

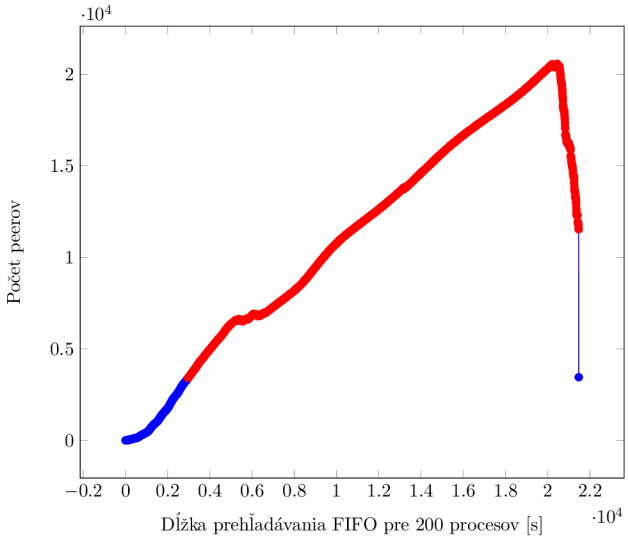
Motiváciou experimentu je overenie funkcie DHT. Predpokladom je začiatok prehliadania z verejne prístupného uzla a postupný prechod stavovým priestorom DHT.

Celý priebeh bol spustený z Kolejnetu na infoheš súboru *The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam*. Dňa 29.4.2018 o 7:10 mal niečo okolo 3367 peerov, ktoré ho zdieľali.

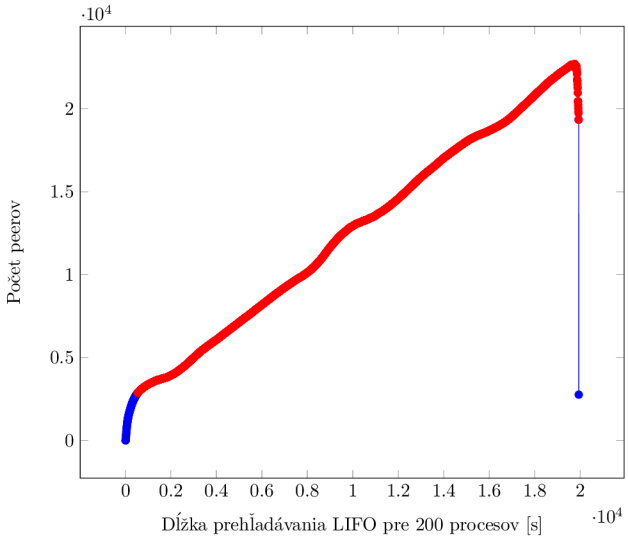
Pre graf 6.1 som použil LIFO frontu 3.2. Dĺžka prehľadávania je 6 hodín. Tento beh som pustil 200 krát aby sme urobili priemerný odhad peerov pri prehľadávaní.

Pre graf 6.2 bola použitá FIFO fronta 3.2. Beh trval rovnakú dobu a bol pustený v rovnakom čase ako LIFO.

Posledný bod v grafoch je preriedenie množiny peerov na základe prekročenej časovej značky 5.4. Pre presný odhad by bolo potrebné skontaktovať všetkých peerov pomocou TCP alebo μTP. Tento spôsob je zdĺhavý a je potrebné implementovať NAT traversal 7.1 rozšírenie.



Obr. 6.1: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, FIFO fronta, štart 29.4.2018 7:10, overenie DHT.



Obr. 6.2: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, LIFO fronta, štart 29.4.2018 7:10, overenie DHT.

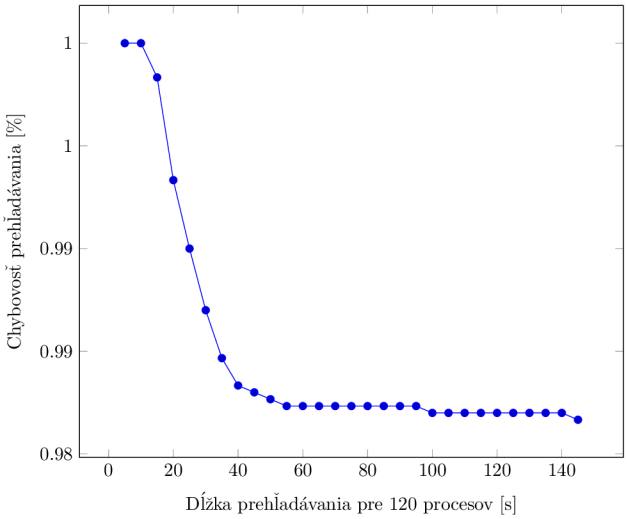
Výsledky ukazujú, že prehľadávanie do hĺbky ma nesporne lepší efekt z krátkodobého prehľadávania. Prehľadávanie do šírky pomocou FIFO fronty je lepšie z hľadiska zistenia, ktoré uzly nám vrátili konkrétnych peerov. Čím dlhšie je spustený prehliadač tým sa viac vypláca použiť FIFO fronta, pretože je možnosť vyčerpania prehľadávania pri použití LIFO fronty. Prehľadávaním so spätným návratom pri použití DFS (LIFO), nám určité uzly (uzly reprezentujúce tracker) sú schopné pri dvoch a viacerých správach KRPC ponúknuť nových peerov. Preto spätný návrat nie je úplne možný, pretože stratíme množstvo peerov vrámci roja. Pri použití FIFO sa k roju postupne dopracujeme k týmto peerom. Tento problém sa vyskytuje zväčša u populárnych torrentoch (1500 peerov a viac), menej populárne torrenty týmto vyčerpaním netrpia.

Červenými bodmi je znázornený čas, kde program prehľadáva neefektívne resp. nekonzistentne. Je to v dôsledku toho, že peerov na tomto torrente bolo 3367, čo je vidieť ako posledný bod na grafe. To znamená, že noví peerovia sa pripájajú a už získaný sa buď odpájili, alebo sú stále peermi pre daný torrent. Z hľadiska získavania globálnych údajov ako je napr. počet všetkých možných peerov na torrent sú tieto červené body nežiadúce a mali by sa prefarbiť na modro.

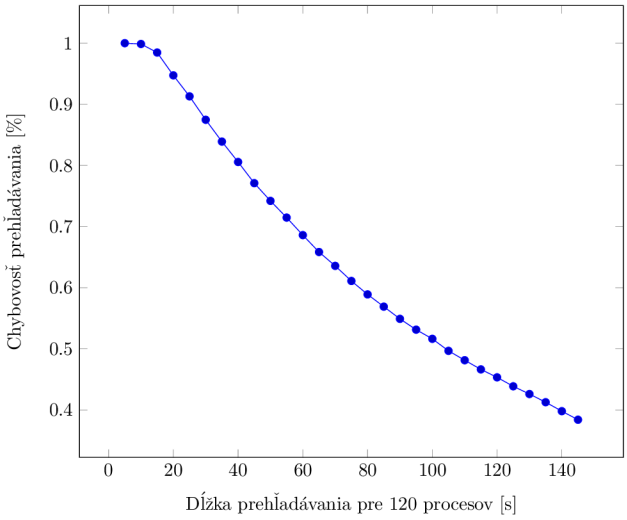
Odhad chybovosti

V tomto experimente je snaha určiť približnú chybovosť pre konkrétny beh. Odhad chybovosti je aplikovaný nad nepopulárnym a populárnym torrentom. Chybovosť je veľmi relatívna a mení sa podľa fluktuácie peerov, a na základe popularity daného torrentu, ako je vidieť vo výsledkoch nižšie.

Chybovosť prehliadača je 100 percentná na začiatku prehliadania. Čím viac peerov program nájde, tým sa chybovosť zmenšuje. Porovnávame LIFO a FIFO, aby sme potvrdili, že z krátkodobého hľadiska je užitie LIFO lepšie ako FIFO.



Obr. 6.3: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, FIFO fronta, štart 29.4.2018 16:30, chybovosť prehľadávania.

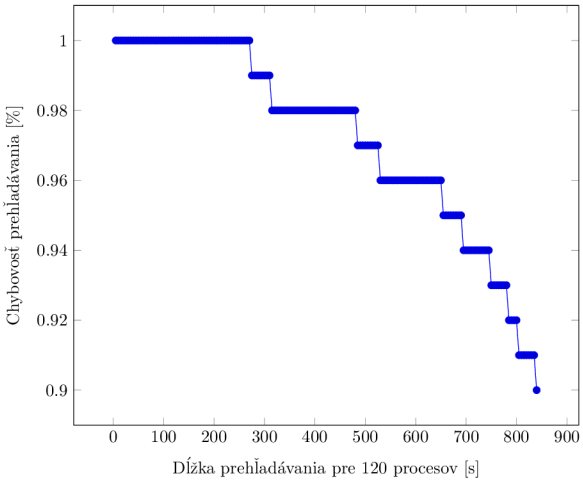


Obr. 6.4: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, LIFO fronta, štart 29.4.2018 16:30, chybovosť prehľadávania.

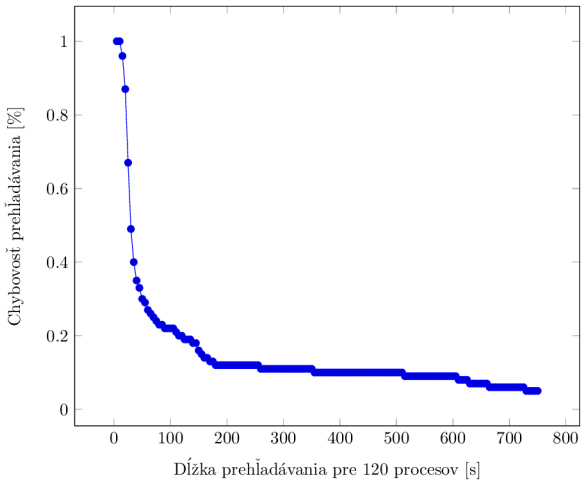
Na grafe 6.4 je vidieť rýchlo klesajúcu chybovosť prehľadávania. Predpokladom je, že počet peerov je v čase 16:30 3043 peerov. Týmto číslom však nie je možné počítať pri každom prehľadávaní. Číslo je pomerne ťažké dostať ale pokusom o zaslanie konektívneho dopytu (handshake) a technikou na prečistenie slovníka peerov, dokážeme tento odhad postupne zaviesť aj v rámci programu. Po použití implementovaného prečistenia pomocou časovej značky bol dosiahnutý počet peerov 3027, čo je o 16 menej než reálny aktuálny počet získaný z logu trackera.

Graf 6.3 ukazuje, ako sa na populárnom torrente nedokáže presadiť použitie FIFO fronty pre krátky časový okamih. Napriek vysokej popularite je chybovosť prehľadávania 99 %.

Tieto štatistiky sa venovali doteraz populárnemu torrentu. Aby boli výsledky validované, bola použitá rovnaká technika na nepopulárnom (málo populárnom) torrente.



Obr. 6.5: Torrent Fedora-LXDE-Live-x86_64-27, FIFO fronta, štart 29.4.2018 16:50, chybovosť prehľadávania.



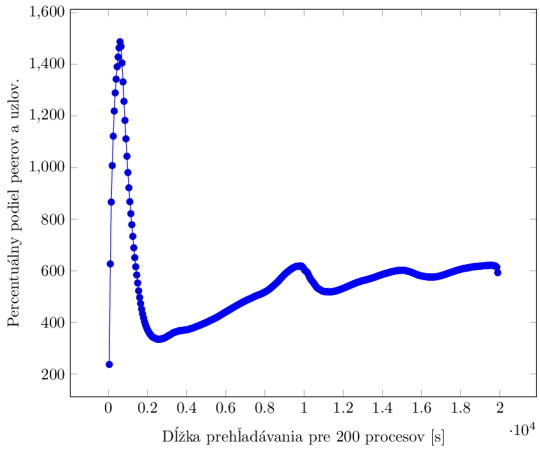
Obr. 6.6: Torrent Fedora-LXDE-Live-x86_64-27, LIFO fronta, štart 29.4.2018 16:50, chybovosť prehľadávania.

Napriek zväčšeniu časovej stopy z 140 sekúnd na 750 sekúnd za použitia nepopulárneho torrentu fedora (v tom čase mal 100 peerov) bolo overené, že použitie LIFO fronty je lepšie z krátkodobého hľadiska. Chybovosť FIFO fronty na obrázku 6.5 sa za 15 minút znížila iba o 10 %, kdežto s LIFO frontou na obrázku 6.6 je dosiahnuté takmer 98 percentné zníženie chybovosti v časovom horizonte 15 minút. Získané údaje sú veľmi priaznivé a monitorovanie siete BitTorrent pomocou DHT má veľkú efektivitu.

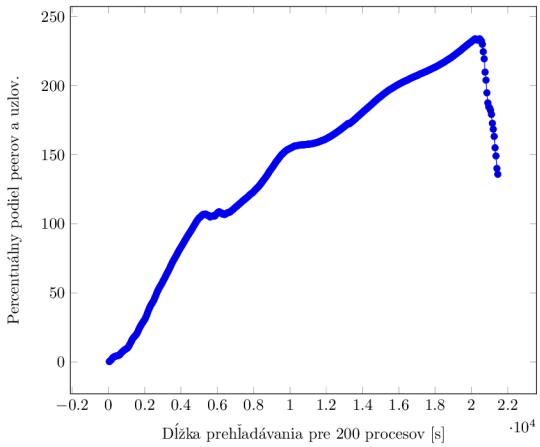
Percentuálny podiel medzi uzlami a peermi

Ďalší experiment, ktorý poukazuje na fakt, že medzi FIFO a LIFO frontou je rozdiel aj v percentuálnom podiele medzi uzlami a peermi. Je to v dôsledku toho, že program prehľadáva strašne rýchlo a na populárnych torrentoch sa nám po zaslaní 20-30 správ môže počet peerov prevýšiť počet uzlov až 20-násobne (závisí od počtu peerov na torrent). Tento jav

síce je očakávaný ale zaujíma nás, aký je tento percentuálny podiel, a ako sa postupným prechádzaním v stavovom priestore peerov a uzlov tento podiel znižuje.



Obr. 6.7: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, LIFO fronta, štart 29.4.2018 07:10, percentuálny podiel medzi peermi a uzlami.



Obr. 6.8: Torrent The Greatest Showman 2017 720p BluRay HEVC x265-RMTeam, FIFO fronta, štart 29.4.2018 07:10, percentuálny podiel medzi peermi a uzlami.

Obrázok s grafom 6.7 ukazuje vysoký pomer peerov k uzlom v krátkom čase. Postupne sa tento podiel zníži, pretože na vstupe je ešte málo vzoriek. Neskôr sa v 400 % akoby zastaví (závisí od popularity) a začne stúpať lineárne. Je vidieť, že na tento torrent sa pripojili nový peerovia a starých sme nevyradili. Správne by sa mali starý peeri vyradiť aby boli výsledky čistejšie. Časté prerieďovanie týchto množín peerov jemne spomaľuje monitorovanie. Tento pomer by mal byť pri častom čistení a určitom oneskorení zasielania správ už len znižovať.

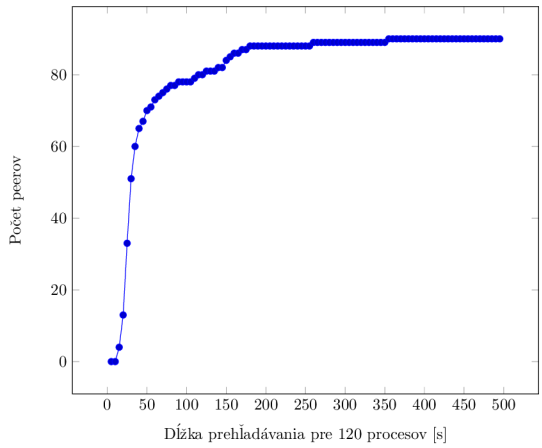
Prehľadávaním do šírky na obrázku 6.8 stojíme krátku časovú dobu na 0 % podiele. S postupom času sa prehľadávač dostáva k lepšiemu podielu v čase. To môže indikovať neprečisťovanie peerov s prekročenou časovou známkou.

Experimentom bolo cielene ukázať ako je dôležitá dĺžka monitorovania. Tá má za následok znižovanie kvality snímky. Je možné prehlásiť o získaných výsledkoch, že sú platné iba

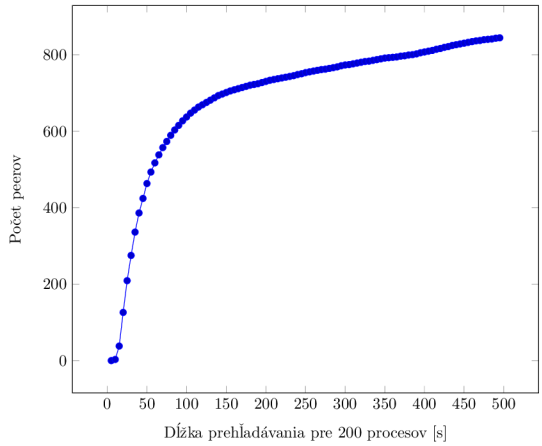
približne do 15 minúty pre LIFO frontu a 70 minúty pre FIFO. Po prekročení dĺžky monitorovania 15 minút je nutné časté prerieďovanie slovníkov, ktorá spomaľuje prehľadávanie.

Porovnanie efektívnej dĺžky prehľadávania

Výsledkom tohto experimentu by malo byť porovnanie efektívnej dĺžky prehľadávania za predpokladu použitia rôzne populárnych torrentov. Ak počet peerov na torrent stúpa, tak sa môže vyhľadávať dlhšie. Všetko to záleží na tom, ako blízko je prehľadávač pri roji, a aká je fluktuácia peerov.



Obr. 6.9: Torrent Fedora-LXDE-Live-x86_64-27, LIFO fronta, štart 29.4.2018 16:50, fluktuácia peerov.



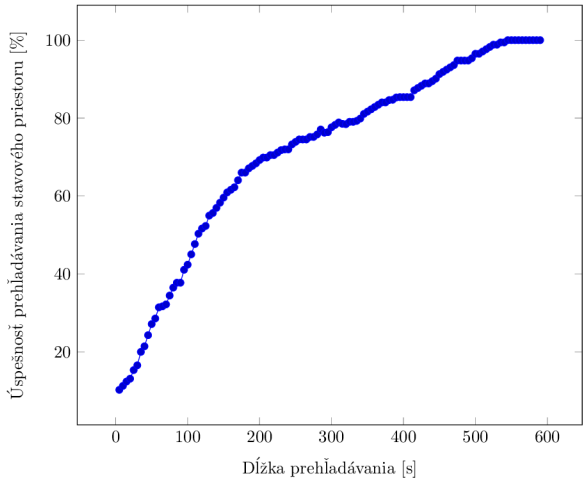
Obr. 6.10: Torrent ubuntu-17.04-desktop-amd64.iso, LIFO fronta, štart 29.4.2018 16:50, fluktuácia peerov.

Získané výsledky z grafov na obrázkoch 6.9 a 6.10 poukazujú na fakt, že monitorovanie záleží na fluktuácií peerov. Ak torrent obsahuje množstvo peerov, prehľadávanie sa predlžuje kvôli vyššej fluktuácií. Výsledok torrentu ubuntu ukazuje, že počet peerov v čase sa zvyšuje pomalšie, než pri torrente fedora, ktorý má menšiu fluktuáciu (menej peerov na torrent sa pripája a odpája).

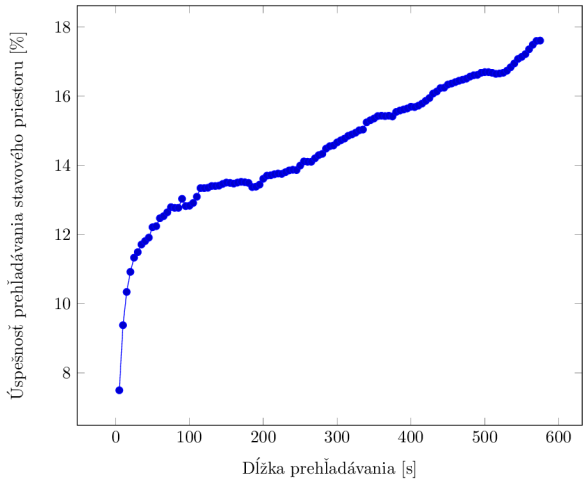
Bernoulliho proces

Použitím Bernoulliho procesu bola uvedená presnosť pre monitorovanie. Z hľadiska monitorovania peerov Bernoulliho proces nemá veľký význam, pretože sa týka predovšetkým monitorovania uzlov. Uzlov je v DHT priestore veľa a je preto potrebné udržiavať stav o efektívnosti aktuálneho prehliadania. Použitím LIFO fronty 6.11 a Bernoulliho procesu v priebehu 360 sekúnd monitorovanie dosiahne 100 percentnej pravdepodobnosti, pretože sme obmedzili DHT priestor iba na roj pre torrent na vstupe.

FIFO fronta prechádza cez celý stavový priestor postupne a nachádza veľa uzlov. Na základe výsledkov grafu z obrázku 6.12 vyplýva, že je potrebné aspoň 500 sekúnd pre dosiahnutie pravdepodobnosti Bernoulliho procesu na stav 100 %.



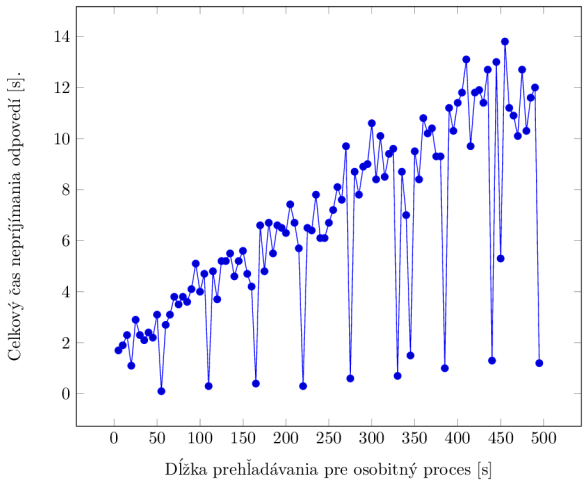
Obr. 6.11: Avengers: Infinity War (2018) English CAM x264 MP3 700MB, LIFO fronta, štart 11.5.2018 21:00, výsledky Bernoulliho procesu.



Obr. 6.12: Avengers: Infinity War (2018) English CAM x264 MP3 700MB, FIFO fronta, štart 11.5.2018 21:00, výsledky Bernoulliho procesu.

Počet nezachytených správ

Posledný experiment sa venoval počtu vypršaných (timed out) správ, ktoré nedošli na sieťové rozhranie v čase 0.1 sekúnd. Tento čas som zaviedol kvôli faktu, že väčšie časové medzery nepomohli pri prijímaní správ a zhoršovali výkonnosť monitorovania. Výsledky sa prezentujú ako $počet\_vypršaných\_správ * 0{,}1$ sekundy. Pomocou tohto vzťahu dostaneme čas, počas ktorého aplikácia čakala. Z grafu na obrázku 6.13 je vidieť, že čakáme maximálne 13.8 sekúnd, ak program beží 455 sekúnd. To činí po zaokrúhlení 3.032 %. Získaný pomer je najhorší možný, kedy monitorovanie je neaktívne. Zvyšné percento času monitorovanie prebieha na plný výkon. Testy sú automatizované a je vidieť aj prácu plánovača jadra operačného systému. Plánovač skresľuje niektoré výsledky, ale pre najhoršiu možnú chybu pri jednom pustení prehliadania je získaný výsledok dostačujúci. Preto je dôležité pustiť monitorovanie viac krát a urobiť priemer získaných výsledkov ako bolo možné vidieť vo výsledkoch predtým napr. 6.4.



Obr. 6.13: Avengers: Infinity War (2018) English CAM x264 MP3 700MB, LIFO fronta, štart 11.5.2018 21:00, dĺžka neaktivity monitorovania.

Experimentálny odhad efektívnej dĺžky prehľadávania

Na základe získaných výsledkov som uviedol vhodnú dĺžku pre prehľadávanie. To som určil iba približne na základe získaných výsledkov z experimentov. Pre LIFO frontu je odhadovaná doba efektívneho monitorovania `15 minút`. Graf na obrázku 6.1 je po čase 15 minút nekonzistený [5]. Je možné prehľadávať dlhšie, no kratšie časové úseky cez deň nám ponúkajú lepší prehľad o vývoji roja peerov a popularity torrentu, než prehliadanie v dlhých časových úsekoch.

Pre FIFO frontu platí, že 90 % úspešnosť prehľadávania nastáva až po približne `70 až 80 minútach` prehľadávania. Vyplýva to zo získaných výsledkov na obrázkoch 6.2 a 6.3.

6.3 Zhrnutie výsledkov z experimentov

V prvých dvoch experimentoch bolo poukázané na fakt, že použitie LIFO fronty je lepšie ako FIFO z krátkodobého hľadiska. Nežiadúcim javom pri použití LIFO je vidieť z prvého

grafu, kde po určitom čase sa akoby vyčerpá prehľadávanie a počet peerov stúpa veľmi pomaly. To sa pri FIFO fronte stane až po prehľadaní celého 160 bitového priestoru.

Experimentom so zistením percentuálneho podielu medzi uzlami a peermi som chcel poukázať na dva fakty:

- Pri prehľadávaní do hĺbky sa rýchlo presúvame stavovým priestorom k roju. To má za následok rýchle vyhľadávanie peerov. Preto je percentuálny podiel vyšší až do 20 násobku oproti uzlom.
- Použitím FIFO fronty na prehľadávanie do šírky spôsobuje počiatočnú réžiu, kde program vyhľadáva iba uzly a žiadnych peerov. Zhruba po jednej až dvoch minútach sa dostávame k prvý peerom, čím sa začína zdvíhať percentuálny podiel peerov k uzlom, až postupne dosiahneme potrebný počet peerov. Pri použití FIFO zavádzame redundanciu komplexným prehľadávaním priestoru 6.8. Preto je aj výsledna množna peerov odlišná od získanej množiny z FIFO, kvôli 50 minútovému rozdielu v prehľadávaní.

Pre určenie odhadu efektívnej dĺžky prehľadávania z existujúcich výsledkov je mnou odporúčaná doba efektívneho monitorovania pre LIFO frontu.

1. Ak je na vstupe populárny(1500 peerov a viac) torrent, odporúčaná doba je 15 až 20 minút.
2. Ak je na vstupe menej(0-1499 peerov) populárny torrent, odporúčaná doba je 12 až 18 minút.

Pre prehľadávanie do šírky platí, že monitorujem aspoň 70 minút pre získanie takmer úplných výsledkov.

Každé nové prehliadanie z rovnakého portu sa zavádza problém *účasti v DHT*. Táto účasť má za následok vysokého počtu požiadavkou na náš port od ostatných už kontaktovaných uzlov. Uzly v DHT chcú od aplikácie získavať informácie o DHT uzloch a peeroch pre daný torrent. Tento jav sa odstraňuje zmenením portu, na ktorom správam monitor načúva. Tento efekt je nežiaduci a prehľadávanie dokáže spomaliť natoľko, že neprijímame žiadne informácie a program by mal slúžiť iba na sprostredkovanie informácií v roji.

Napriek tomu by bolo vhodné zmeniť infraštruktúru informácií v DHT pre trackery. Každý tracker si momentálne uchováva svoju smerovaciu tabuľku aj s logmi, ktoré poskytujú rôznym webovým serverom pre získavanie informácií. Tieto informácie o peeroch sú často nepresné a je potrebné ich často meniť. Preto by sa mohli takéto redundantné logovacie súbory vymeniť za aktívne monitorovanie s vyššou odozvou (5 sekúnd). To nám zaistí, že nepotrebujeme pre distribúciu takýchto nepresných informácií logovacie súbory ale 1 aktívny prvok, ktorý monitoruje viacero súborov naraz aby ponúkal čo najpresnejšie informácie. Informácie z trackerov sú redundantné a obsahujú nakopírovaných peerov.

Kapitola 7

Možné rozšírenia monitorovania

Pre rozšírenie aplikácie som navrhol 7 možných variant, ktoré by nemali mať veľký dopad na réžiu. Sú nimi:

- Skontaktovanie peerov kvôli zlepšeniu konečných výsledkov prehľadávania.
- Implementácia IPv6 dopytov.
- DHT škrabance.
- DHT bezpečnostné podnety
- DHT infoheš číslovanie.
- Zistenie meta informácií, pri prehľadávaní do šírky.
- Ochrana proti DoS.

Najpodstatnejším rozšírením by malo byť dodatočné skontaktovanie peerov, kvôli zisteniu existencie v roji. To nebude mať za dopad zníženie rýchlosti alebo efektivity prehľadávania ale predĺžia celkové prehliadanie.

Všetky varianty rozšírenia okrem prvej sú `voliteľné` a aktuálne BitTorrent klienty podporujúce MLDHT ich nemusia nutne implementovať. DHT varianty týkajúce sa zmeny alebo rozšírenia DHT zvyšujú réžiu no možno zlepšujú monitorovanie. DHT bezpečnostné podnety slúžia síce k zhoršeniu prehľadávania siete, no zavádzajú bezpečnosť pri aktívnom monitorovaní.

7.1 Skontaktovanie peerov kvôli zlepšeniu výsledkov prehľadávania

Pri testovaní sa vyskytol problém ako prerieďť slovník výsledných peerov. Aktuálne riešenie preriedenia slovníka je pomocou prekročenej časovej známky pri zázname o peerovi. Skontaktovanie s množstvami peerov pomocou TCP alebo μTP nebolo počas implementácie efektívne. Bolo to v dôsledku toho, že buď peeri nepoznajú sémantiku týchto správ alebo sú za NAT-om resp. firewallom. To ma za následok jemnú odchýlku v získaných výsledkoch. Ak peer prekročí časovú známku, je vyhodený z množiny peerov. To však nemusí znamenať, že aktuálne peerom pre daný torrent nie je, naopak, môže peerom ďalej byť ale preto ho musíme skontaktovať buď pomocou TCP alebo μTP.

Po diskusii s vedúcim práce sme navrhli možné riešenie pomocou implementácie NAT-traversal [22]. Peer to peer systémy NAT-traversal zohľadňujú a implementujú. Táto bakalárska práca sa venovala monitorovaniu a popísaniu rozšírení, čím by mohol byť aj NAT-traversal. Pre príklad ak pre 3367 nájdených peerov z experimentu na obrázku 6.1 potrebujeme zaslať a priať odpoveď, tak je to 6734 požiadavkov na zaslanie a priatie datagramu. Preto je potrebné proces skontaktovania urýchliť alebo vymyslieť inú variantu prístupu k skontaktovaniu.

7.2 Implementácia dopytov IPv6

Existujú dva rôzne DHT: DHT IPv4 a DHT IPv6. Oba sú plne nezávislé, z čoho vyplýva, že v protokole DHT IPv4 nie sú uložené žiadne IPv6 dáta. Uzol, ktorý sa chce zúčastniť oboch DHT, musí udržiavať dve odlišné smerovacie tabuľky. V oboch tabuľkach by mal používať rovnaký identifikátor uzlu. Táto podmienka síce nie je nutná no zjednodušuje implementáciu, ladenie a mierne zvyšuje efektivitu. Správy obvykle obsahujú dáta v rodine adries, ktorá vyplýva z protokolu sieťovej vrstvy. V prípade požiadavkou find_node a get_peers je možné požiadať aby odpoveď obsahovala informácie o IPv4 uzloch, informácie o IPv6 uzloch alebo oboje varianty.

7.2.1 Výber zdrojových IP adries

Pri rozšírení DHT IPv6 musíme brať ohľad na Teredo [13]. Plne pripojenie IPv4 uzlu na IPv6, aj bez toho aby sme mali akékoľvek IPv6 pripojenie od operátora. Teredo podporuje Microsoft Windows.

Teredo spojenie môže byť účinnejšie ako prirodzená IPv6 konektivita [7], hlavne ak sa pripájame na iných užívateľov Teredo. Skúsenosti ukazujú, že smerovanie Teredo bývajú krehké [7]. Preto by sa mali uzly vyhýbať adresám Teredo vždy ak je to možné.

7.2.2 Zmena v sémantike a nové parametre

Nové parametre, ktoré sa musia nachádzať v správach sú: `nodes6` a `want`. Want parameter je povolený pre správy typu find_node a get_peers. Môže obsahovať jak reťazec „n4“ tak reťazec „n6“. To však nie je podmienkou. Je dostačujúce poslať tieto dopyty cez IPv4 schránku pre odpoveď „nodes“ a dopyt cez IPv6 schránku pre odpoveď s reťazcom „nodes6“. Pre podporu takéhoto mechanizmu je treba zmeniť správy typu find_node a get_peers, čo nie je kompatibilné s pôvodnou špecifikáciou.

7.3 DHT škrabance

Škrabance sú dôležitou súčasťou ekosystému BitTorrent, pretože umožňujú posúdiť stav torrentu a klientov, aby zistili, ktorý roj si vybrať z fronty. To všetko sa vykoná bez toho aby sa musel uzol zúčastniť v roji. Implementovaním tohto rozšírenia sa dokáže lepšie štrukturovať smerovacia tabuľka. Údržba smerovacej tabuľky je nepresná. Existujú prípady pri použití smerovacej tabuľky, že viac než tucet uzlov sa nachádza okolo cieľového kľúča bez toho aby uzol mal úplný súbor vrstovníkov. Pre zlepšenie je znova potrebné upraviť správy o 2 ďalšie položky, čo prináša réžiu v prenosovom pásme a implementovanie Bloom filtru [1]. Ak by sa nejednalo iba o aktívne monitorovanie peerov, toto rozšírenie bude mať veľký potenciál.

7.4 DHT bezpečnostné podnety

Účelom bezpečnostných podnetov je sťažovanie niekoľkých konkrétnych útokov proti DHT BitTorrentu a sťažiť prehľadávanie siete. Ovládaním mnohých IP adries by útočník mohol odmietnuť akýkoľvek infoheš. Ovládaním 8 násobku aktuálnych IP adries by sa útočník mohol zmocniť akéhokoľvek infohešu. S protokolom IPv4 by snooping útok vyžadoval /8 blok IP, čo by umožnilo prístup k 16,7 miliónov [21]. IP adries. Ďalším problémom s hešovaním IP je, že viacerí užívatelia NAT sú nútení spustiť svoje uzly DHT na rovnakom ID. Pre zlepšenie by sa malo miesto kryptografickej funkcie heš používať CRC32C (Castagnoli) [21]. Je to opodstatnené z týchto dôvodov podľa článku [21]:

1. Rýchla funkcia.
2. Produkuje dobre distribuované výsledky.
3. Potreba, aby hešovacia funkcia bola jednosmerná (vstupná sada je taká malá, že by každá hešovacia funkcia mohla byť obrátená).
4. CRC32C (Castagnoli) je podporovaný v hardvéri pomocou SSE 4.2 modulu, čo môže výrazne zrýchliť výpočet.
5. Existujú predovšetkým dva testy spustené na SHA-1 a CRC32C na určenie distribúcie výsledkov. Prvý je počet bitov vo výstupnej množine, ktoré obsahujú všetky možné kombinácie bitov. CRC32C má dlhší prefix než SHA-1. To znamená, že uzly budú mať stále rovnomerne rozložené identifikátory, a to aj vtedy, keď nie sú `použité IP adresy` rovnomerne rozložené.

7.5 DHT infoheš číslovanie

Toto rozšírenie umožňuje DHT uzlom načítať vzorku infohešu, ktoré majú iné uzly v úložišti. Toto indexovanie je už možné a v praxi sa uskutočňuje pasívnym sledovaním dopytov get_peers. Prístup je však neefektívny, pretože uprednostňuje indexátory s množstvom unikátnych IP adries, ktoré majú k dispozícii. Dokonca aj stimuluje škodlivé chovanie ako je spoofing identifikátorov uzlov a pokusy znečisťovať smerovacie tabuľky ostatných uzlov [28].

Rozšírením by mal jeden uzol sledovať celý DHT priestor behom niekoľkých hodín bez toho aby sa musel uchyľovať k nevyhovujúcemu chovaniu. Vzhľadom k tomu, že nie je možné priamo vyhľadávať konkrétne torrenty, neočakáva sa, že priemerní klienti skutočne používajú RPC, potrebujú iba podporu odpovedi pre akýkoľvek uzol.

Toto rozšírenie zavádza novú RPC správu `sample_infohashes`. Vzdialený uzol vráti reťazec viac spojených infohešov, pre ktoré ma hodnoty get_peers. Keď ľubovoľný uzol sleduje viac infohešov než by sa vošlo do paketu, vráti tak náhodne vzorkovanú podmnožinu infohešov. Po dosiahnutí limitu infohešov na paket, môžu uzly občas predkomplilovať podmnožinu a vrátiť ju namiesto výpočtu pre každý dopyt.

Pre ľubovoľný uzol to znamená ukladať pole s názvom *num* [2], ktoré indikuje koľko infoheš kľúčov je v úložišti uzla. Ak je toto číslo väčšie ako číslo vrátených vzorkov, potom indexer môže po určitom časovom intervale dostať ďalšie vzorky.

7.6 Zistenie meta informácií, pri prehľadávaní do šírky

Meta informácie o infoheši sa dajú jednoznačne zistiť iba po ustanovení spojenia. Časť z týchto meta informácií sa môže nachádzať v magnet-odkaze alebo .torrent súbore. Tieto meta informácie pomáhajú pri zisťovaní popularity torrentov, popularity použitých klientov, aktuálny proces iných uzlov, rýchlosť odosielania/prijímania dát a pod. Z tohto sa dajú zistiť zaujímavé informácie z hľadiska štatistiky. Z predošlých štúdií vyplýva napr.

- Popularita najnovších torrentov je najvyššia po 1 až 2 dňoch.
- Medzi 5 najpopulárnejších kategórií podľa Wolchka a Haldermana [29] patria - 1. Filmy (asi 50 %) , 2. Hudba (S príchodom Spotify a rôznych internetových rádií tento fenomén klesá), 3. neklasifikované, 4. Softvér, 5. Knihy.

7.7 Ochrana proti DoS

Útok typu DoS (denial of service) je veľmi nepríjemný z hľadiska užívania akejkoľvek verejne prístupnej služby. Principiálne sa jedná o zaslanie veľkého množstva legitímnych správ, čím sa zahltí prenosové pásmo medzi nami a útočníkom. Ak je aplikácia navrhnutá tieto správy prijímať bez predikcie tak sa úplne zasekne a bude musieť odpovedať týmto správam. Preto som obmedzil prijímanie správ iba na 1 typ odpovede, ktorá obsahuje peerov a uzly, ktorú dostaneme od uzlov po prijatí odpovede get_peers. V dnešnej dobe sa zaoberajú štúdie [23] týmto útokom. Ak sme cieľom týchto útokov, je potrebné pri detekcií DoS útoku tieto legitímne správy rozložiť distribuovane. Týmto faktom sa zaoberá aj článok [23].

Preto by mohlo byť možným rozšírením zaistenie predikcie takéhoto útoku a následne rozdistribuovanie správ prípadne či zmena portu pre vysielanie alebo prijímanie.

Kapitola 8

Záver

Cieľom tejto práce bola implementácia prehľadávača v súlade s princípom DHT v sieti BitTorrent pre prenos informácií. Práca sa zameriava na implementáciu a aplikáciu monitorovania v sieti MLDHT. V úvode práce sa venovalo problematike monitorovania v distribuovaných sieťach. Prebehla analýza jednotlivých metód pre monitorovanie pomocou DHT. V kapitole o návrhu architektúry sa zaoberalo architektúrou aplikácie, vstupom a výstupom programu, možnostiam testovania a prenositeľnosti monitorovacieho programu na rôzne platformy.

Implementáciou algoritmov z návrhu a správnym rozdelením tried je dosiahnutá vysoká efektivita monitorovania. Zaoberalo sa aj obmedzeniami a výhodami tejto aplikácie. Obmedzením aplikácie sú zahltenie prenosového pásma a robustnosť odpovedí, čo malo za príčinu pomerne vysoké vyťaženie aplikácie. Testovaním implementácie je zaistená prenositeľnosť programu pomocou kontajnerizácie na 2 rôzne platformy, ako aj používanie aplikácie v rámci kontajneru.

Výsledky práce priniesli pohľad na distribuovaný systém z rôzných strán. Na základe vykonaných experimentov je zistená účinnosť prehľadávania do hĺbky. Dokáže sa určiť chybovosť prehliadača v čase. Bernoulliho proces síce monitorovanie peerov neovplyvní zásadne, ale ponúka neodlúčiteľnú informáciu o tom ako efektívne monitorujeme v čase. Experimenty sú uskutočnené aj nad časom, keď je monitorovanie neaktívne (graf 6.13). To vzniklo v dôsledku užívania nespojovanej služby UDP, ktorá sama nezaistí, že datagram sa nemusí doručiť. Výsledkom experimentu je zavedenie 0,1 sekundového oneskorenia prehľadávania, počas ktorého čakáme na datagram. Oneskorenie malo za dopad to, že 3 % z času aktívneho monitorovania je program neaktívny. Experimentálne bol zavedený odhad pri monitorovaní s použitím LIFO a FIFO fronty (LIFO 15 minút a FIFO 70 minút). Tento čas by mal byť v súlade s určenou chybovosťou v čase získaného z grafov 6.4 a 6.2.

Prácu je možné ďalej rozšíriť o niektoré z uvedených rozšírení zmienených v kapitole 7. Je potrebné vždy dbať na rýchlosť a efektívnosť implementovaného riešenia. Čím pomalšie prehliadanie je, tým je ťažšie získať konzistentnú snímku. V experimentoch 6.1 a 6.2 bolo vidieť nekonzistenciu snímky, pretože sa prehľadávalo príliš dlho. Taktiež používateľ musí brať na ohľad, že pustením takéhoto klienta je v potenciálnom ohrození DoS útokom. Útok by mohol mať za následok vyradenie aplikácie a nulový dopad na monitorovanie alebo používateľovu interakciu so systémom.

Literatúra

[1] BEP 33. DHT Scrapes. 2010, [Online; navštevené 3.4.2018].
URL http://www.bittorrent.org/beps/bep_0033.html

[2] BEP 51. DHT Infohash Indexing. 2016, [Online; navštevené 3.4.2018].
URL http://www.bittorrent.org/beps/bep_0051.html

[3] MLDHT sanitazing-algorithms. 2017.
URL https:
//github.com/the8472/mldht/blob/master/docs/sanitizing-algorithms.rst

[4] How RPC Works. 28.3.2003.
URL https://technet.microsoft.com/en-us/library/cc738291(v=ws.10).aspx

[5] Andrew Loewenstern, A. N.: Bep 5: DHT Protocol. 2008.
URL http://www.bittorrent.org/beps/bep_0005.html

[6] Chan-Tin, E.; Heorhiadi, V.; Hopper, N.; aj.: Hijacking the Vuze BitTorrent network: all your hop are belong to us. *IET Information Security*, ročník 9, č. 4, 2014: s. 203–208.

[7] Chroboczek, J.: BEP 32. BitTorrent DHT Extensions for IPv6. 2009, [Online; navštevené 28.3.2018].
URL http://www.bittorrent.org/beps/bep_0032.html

[8] Dabek, F.; Cox, R.; Kaashoek, F.; aj.: Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM Computer Communication Review*, ročník 34, ACM, 2004, s. 15–26.

[9] Docker: What is container. 2018, [Online; navštevené 12.11.2017].
URL https://www.docker.com/what-container

[10] G. H. Arvid Norberg, L. S.: Bep 10: Extension protocol. 2008, [Online; navštevené 6.10.2017].
URL http://www.bittorrent.org/beps/bep_0010.html

[11] Hazel, A. N. G.: Bep 9: Extension for peers to send metadata files. 2008, [Online; navštevené 6.10.2017].
URL http://www.bittorrent.org/beps/bep_0009.html

[12] Hughes, R.: The Producer Consumer Pattern. 2013, [Online; navštevené 29.4.2018].
URL https://dzone.com/articles/producer-consumer-pattern

[13] Huitema, C.: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). 2006, [Online; navštevené 21.3.2018].
URL https://tools.ietf.org/html/rfc4380

[14] J. Falkner, J. P. a. o., M. Piatek: Profiling a million user DHT. In *Proceeding of ACM conference on Internet measurement*, 2007.

[15] K. Junemann, P. A.: Bitmon: A tool for automated monitoring of the bittorent dht. In *Conference on Peer-toPeer Computing*, ISBN 978-1-4244-7139-3, s. 1–2.

[16] Liang, W.; Jussi, K.: *Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT*. University of Helskinki, Finland, 2013, ISBN 978-1-4799-0521-8.

[17] Liu, X.; Meng, T.; Cai, K.; aj.: Rainbow: A Robust and Versatile Measurement Tool for Kademlia-Based DHT Networks. In *2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec 2010, ISSN 2379-5352, s. 222–228.

[18] Loewenstern, A.; Norberg, A.: Bep 5: DHT Protocol. 2017, [Online; navštevené 6.10.2017].
URL http://www.bittorrent.org/beps/bep_0005.html

[19] Maymounkov, P.; Mazières, D.: Kademlia: A Peer-to-peer Information System Based on the XOR Metric. New York, Mar. 2002, ISBN 3-540-44179-4, s. 53–65.

[20] Moritz Steiner, T. E.-N.; Biersack, E. W.: Long Term Study of Peer Behavior in the KAD DHT. 2009.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4926137

[21] Norberg, A.: BEP 42. DHT Security extensions. 2014, [Online; navštevené 3.4.2018].
URL http://www.bittorrent.org/beps/bep_0042.html

[22] P. Srisuresh, D., B.Ford: State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). 2008, [Online; navštevené 24.4.2018].
URL https://tools.ietf.org/html/rfc5128

[23] Prakash, A.; Satish, M.; Bhargav, T. S. S.; aj.: Detection and Mitigation of Denial of Service Attacks Using Stratified Architecture. *Procedia Computer Science*, ročník 87, 2016: s. 275–280.

[24] Rosen, A.: Towards a Framework for DHT Distributed Computing. 2016.

[25] Guido van Rossum, N. C., Barry Warsaw: PEP 8 – Style Guide for Python Code. 2013.
URL https://www.python.org/dev/peps/pep-0008/

[26] Steiner, M.; Carra, D.; Biersack, E. W.: Faster Content Access in KAD. In *2008 Eighth International Conference on Peer-to-Peer Computing*, 2008, ISBN 978-0-7695-3318-6, s. 195–204.

[27] Stutzbach, D.; Rejaie, R.: Understanding churn in peer-to-peer networks. [Online; navštevené 12.12.2017].
URL http://conferences.sigcomm.org/imc/2006/papers/p19-stutzbach2.pdf

[28] Wang, L.; Kangasharju, J.: Real-world sybil attacks in BitTorrent mainline DHT. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, IEEE, 2012, s. 826–832.

[29] Wolchok, S.; Halderman, J. A.: Crawling BitTorrent DHTs for Fun and Profit. In *Proceedings of the 4th USENIX conference on Offensive technologies*, USENIX Association, 2010, s. 1–8.

Zoznam príloh

Príloha A

Obsah CD nosiča

Na priloženom médiu sa v priečinku *dht_crawler* nachádzajú zdrojové súbory pre monitorovanie peerov pomocou DHT. Priečinok *doc* obsahuje dokumentáciu vygenerovanú zo zdrojových súborov v docs podpriečinu a v *code_coverage* je možné dohľadať % úspešnosť prechodu cez jednotlivé riadky kódu. V *examples* nájdeme vstupné torrent súbory, infoheše a manget-odkazy. Logs obsahuje iba prázdne adresáre a skript na vyčistenie všetkých logov a slúži na naplnenie logov pri automatických testoch. Na generovanie grafov som vytvoril automatický skript *get_results*, ktorý sa nachádza v priečinku *results*. Testy sú rozdelené na jednotkové a integračné (experimenty).

```
DHT-crawler/
├── dht_crawler
├── doc
│   ├── code_coverage
│   │   └── htmlcov
│   └── docs
├── examples
├── logs
│   ├── client_send
│   ├── error
│   ├── fifo_less
│   ├── fifo_queue
│   ├── less_peers
│   ├── lifo_less
│   ├── lifo_queue
│   ├── nodes_timer
│   ├── respond
│   ├── test
│   └── unique_ip_port
├── results
│   └── graphs
│       └── avg
└── tests
    ├── integration
    └── unit
```

Obr. A.1: Aresárová štruktúra projektu