Računalništvo in informatika

Pridobivanje podatkov iz omrežja DHT

Analiza pretočnega prometa skozi vozlišča protokola BitTorrent in prenos metapodatkov

Anton Luka Šijanec <anton@šijanec.eu>

4. letnik

□

Mentor: Andrej Šuštaršič, univ. dipl. ing. elektr.

2023
Gimnazija Bežigrad

Kazalo

Povzetek

Porazdeljene razpršilne tabele (angl. distributed hash table) so razpršilne tabele, ki podatke, ponavadi so to dokumenti, strukturirani kot vrednost in njen pripadajoč ključ, hranijo distribuirano na več vozliščih, kjer se podatki shranjujejo. V računalniških sistemih se DHT uporablja za hrambo podatkov v omrežjih P2P (angl. peer to peer), kjer se podatki vseh uporabnikov enakomerno porazdelijo med vozlišča in so tako decentralizirani in preprosto dostopni članom omrežja. Ker se podatki izmenjujejo znotraj omrežja na vozliščih, ki z izvorom in destinacijo podatkov niso povezani, jih lahko vozlišča v velikih količinah shranjujejo.

V raziskovalni nalogi je preverjena praktična zmožnost pridobivanja velike količine podatkov v omrežju BitTorrent za P2P izmenjavo datotek, pridobljeni podatki pa so analizirani. Vsaka poizvedba po seznamu imetnikov datotek vsebuje ključ podatka v DHT in se prenese preko okoli $\log_2 n$ vozlišč, kjer je n število vseh uporabnikov v omrežju. Ker vsaka poizvedba obišče tako veliko število vozlišč, lahko eno vozlišče prejme veliko obstoječih ključev v omrežju, s katerimi si lahko prenese metapodatke v omrežju BitTorrent.

Naloga se osredotoči na pridobivanje metapodatkov v omrežju BitTorrent, glede prenosa datotek, ki jih ponujajo računalniki, pa se vsled njihove velikosti ne opredeli. Metapodatki konceptualno sicer niso shranjeni v DHT (namesto metapodatkov o datotekah so v omrežju shranjeni seznami računalnikov, od katerih si metapodatke lahko prenesemo), vendar odkrivanje njihovega obstoja omogoči DHT.

Ključne besede porazdeljena razpršilna tabela, distribuirani sistemi, P2P omrežje, podatkovno rudarjenje, BitTorrent

1 Uvod

1.1 Peer-to-peer omrežja za distribucijo datotek

Izmenjava in distribucija velikih datotek na internetnih omrežjih veliki količini odjemalcev predstavlja težavo, saj je v osnovi TCP/IP sklada protokolov isto datoteko poslati tolikokrat, kolikor odjemalcev imamo. Distributorji večjih količin podatkov na internetu se morajo zaradi centraliziranega modela infrastrukture strežnikov, kjer centraliziran strežnik posreduje identične informacije večkrat večim odjemalcem, ki med seboj ne komunicirajo, posluževati dragih metod kolokacije strežnikov.

Koncept P2P (angl. peer-to-peer) predstavlja alternativen način distribucije identičnih datotek večim odjemalcem. Namesto enega strežnika, ki iste podatke pošlje vsakič znova odjemalcem, v omrežjih P2P za distribucijo datotek ni razlike med strežnikom in odjemalcem. Vsak odjemalec podatke tako prejema kot tudi pošilja. Takoj ko odjemalec prejme vsebino od drugega odjemalca, jo bo tudi sam začel deliti naprej drugim odjemalcem, ki to vsebino tudi sami iščejo. S svojim sodelovanjem v distribuciji vsebine razbremeni ostale odjemalce, ki datoteke distribuirajo prosilcem, saj so P2P omrežja izdelana tako, da lahko odjemalci vsebino prejemajo od večih odjemalcev hkrati. Čim več odjemalcev razpolaga z neko vsebino, tem manj podatkov mora poslati posamezen odjemalec novemu odjemalcu, ki si to vsebino želi prenesti. Tako se zmanjša obremenitev omrežja, saj je količina prenesenih podatkov po omrežni topologiji čedalje bolj razporejena.

Sistem pa ni povsem brezhiben, saj je še vedno treba na nek zunanji način med seboj povezati odjemalce, ki so zainteresirani za določeno temo (recimo za določeno datoteko). Druga očitna slabost pa je, da je možno ugotoviti, kdo prenaša kakšno vsebino, ker odjemalci (neke datoteke) vedo za internetne naslove drugih odjemalcev, saj lahko le tako neposredno čim bolj učinkovito komunicirajo z njimi.

Koncept P2P ni namenjen le distribuciji datotek, temveč se zaradi svoje prednosti razbremenitve strežnikov dandanes vse pogosteje uporablja, na primer pri spletnih videokonferencah, anonimizacijskih omrežjih, kriptovalutah, internetu stvari in drugje.

1.2 Protokol BitTorrent

Za distribucijo datotek morajo odjemalci za medsebojno komunikacijo uporabljati standardiziran protokol za signalizacijo prenosov. Eden izmed najbolj razvitih (ci-

tiraj) in uporabljenih protokolov je BitTorrent. Prvo implementacijo je idejni avtor protokola izdelal leta 2001(citiraj), od leta 2008 pa lahko z objavo dodatkov pri razvoju skupaj sodeluje širša javnost(citiraj). Zaradi razširljive zasnove je protokol namreč moč dopolnjevati in mu s tem dodajati nove funkcije. Sprva je na primer protokol omogočal le pospešeno distribucijo datotek iz enega strežnika k več odjemalcem (citiraj), saj so si odjemalci koščke vsebine delili med seboj, vendar je še vedno temeljil na centralnih strežnikih, ki stalno gostijo datoteke in koordinirajo skupek odjemalcev, danes pa omogoča (citiraj) od centraliziranih strežnikov povsem neodvisno delovanje, prav z uporabo protokola DHT.

Za nadaljnji opis je potrebno poznavanje pojmov, ki jih uvede BitTorrent:

Pojem	Izvirno angleško ime	Razlaga
soležnik (citraj)	peer	odjemni program na računalniku ali računalnik, za povezavo nanj potrebujemo njegov IP naslov in vrata
roj (citiraj)	swarm	več soležnikov, ki prenašajo datoteke torrenta
torrent/metainfo (ni ustaljenega prevoda, neposredni prevod bi bil *hudournik*)	torrent ali metainfo	strukturirana datoteka v obliki bencoding, ki vsebuje metapodatke o datotekah, torej imena datotek, njihove velikosti, razporeditev po imenikih, zgoščene vrednosti za preverjanje istovetnosti ob prenosu in drugo
sledilnik (citiraj)	tracker	centraliziran strežnik, ki hrani podatke o tem, kateri soležniki so v roju določenega torrenta
košček (citiraj)	piece	del vsebine torrenta konstantne dolžine
infohash (ni ustaljenega prevoda)	infohash	zgoščena vrednost serializiranih podatkov pod ključem info v torrentu, ki unikatno opišejo ključne metapodatke o torrentu
announce (ni ustaljenega prevoda, neposredni prevod bi bil *obvestilo*)	announce ali ~ment	obvestilo ali obveščanje o obstoju soležnika za torrent, ki ga pošlje soležnik bodisi sledilniku bodisi v DHT in s tem zagotovi, da bodo ostali soležniki izvedeli za njegov obstoj in se potencialno povezali nanj

Tabela 1.1: Nepopoln seznam pojmov BitTorrenta, potrebnih za razumevanje naloge

BitTorrent protokol ne omogoča iskanja po datotekah, ki se prenašajo po omrežju. Za prenos datoteke je najprej treba poznati metapodatke o obstoječih datotekah. Ti metapodatki so shranjeni v t. i. obliki torrent, strojno berljivi datoteki, serializirani

s preprosto serializacijsko metodo bencoding. Vsebujejo imena in poti datotek ter njihove zgoščene vrednosti, ime torrenta, lastnosti prenosa: velikost koščka, ime, zasebnost (angl. private torrent).

V nalogi se ne osredotočam na klasičen način iskanja soležnikov s sledilniki, prav tako ne govorim o prenosu datotek od soležnikov ter o signalizaciji za omejevanje pasovne širine prenosa (choking), temveč samo o prenosu metapodatkov.

1.3 Protokol BitTorrent DHT

DHT je kot koncept definiran zelo splošno, za BitTorrent je uporabljen sistem DHT, imenovan Kademila. Uporablja se odpravo odvisnosti od sledilnika, saj lahko v njej hranimo seznam soležnikov v roju.

Pojem	Izvirno angleško ime	Razlaga
vozlišče (citiraj)	node	odjemni program na računalniku ali računalnik
usmerjevalna tabela (citiraj)	routing table	seznam vozlišč, ki ga hrani posamezno vozlišče
ID vozlišča	node ID	160 bitov dolga naključno generirana številka, ki pripada vsakemu vozlišču
merilo za razdaljo	distance metric	funkcija (XOR), ki izrazi konceptualno razdaljo kot 160 bitov dolgo številko med dvema vozliščema
koš (citiraj)	bucket	na posamezno vozlišče relativna množica drugih vozlišč, ki so si glede na merilo za razdaljo blizu, shranjena v usmerjevalni tabeli

Tabela 1.2: Nepopoln seznam pojmov Kademile, potrebnih za razumevanje naloge. Za noben pojem nisem našel ustaljenih slovenskih prevodov.(citiraj)

Kademilo, kot se uporablja v BitTorrentu, si lahko za začetek predstavljamo kot abstraktno razpršilno tabelo, ki je shranjena porazdeljeno na velikem omrežju vozlišč/računalnikov in podpira naslednji operaciji (citiraj):

Pridobi soležnike Vrne seznam soležnikov (IP naslov in vrata) za torrent, opisan z njegovim infohashom.

Announce V seznam soležnikov za torrent, opisan z njegovim infohashom, vstavi IP naslov in vrata pošiljatelja zahteve.

Cilj raziskovalne naloge je s sodelovanjem v DHT omrežju pridobiti čim več obstoječih ključev v razpršilni tabeli, da lahko z operacijo **pridobi soležnike** pridobimo sezname soležnikov, na katere se lahko povežemo in od njih prenesemo metapodatke o torrentih, da lahko te podatke kot izvleček celotnega omrežja kasneje uporabimo za analiziranje.

1.4 Obstoječe implementacije

Da je to pridobivanje mogoče, se ve že od vpeljave protokola DHT, saj obstaja mnogo implementacij koncepta pridobivanja podatkov iz omrežja DHT za prenos metapodatkov torrentov:

- Spletna stran in istoimenski program **Btdigg** (citiraj)
- Spletna stran v kitajščini pod več imeni: **clzhizhu**.com, cilizhizhu, clzz1020.buzz, clzz1025.buzz, clzz1026.buzz idr. Za obstoj te strani sem ugotovil med implementacijo programa, saj je njeno iskanje invazivno in moti obstoječe delovanje DHT. Več o tem v sledečih poglavjih.
- Spletna stran **I know what you download** (citiraj), ki hrani najdene podatke o rojih in s tem razkrije identiteto prenašalcev.

2 Teoretični del

2.1 `bencoding` serializacija (bkodiranje)

V BEP-0003 (citiraj) je opisan pojem bencoding serializacije, s katero je serializirana večina paketov, ki se pošiljajo med vozlišči DHT in soležniki. Strukturo, ki opisuje JSONu (citiraj) podobno strukturirane podatke, vsebuje štiri podatkovne tipe:

- **niz** ali **string** je serializiran tako, da ASCII (citiraj) številki dolžine niza sledi dvopičje in za njim niz bajtov. Primer: `18:pozdravljen, svet!`
- **število** ali **int** je serializirano tako, da ASCII znaku `i` sledi ASCII številka (lahko tudi negativna) in nato znak `e`, ki označuje konec podatka. Primer: `i-1337e`
- **seznam** ali **list** je serializiran tako, da ASCII znaku `l` sledi poljubno število podatkov lahko tudi različnih tipov, nato pa znak `e`. Primer: `li-1337e18:pozdravljen, svet!lee`
- **slovar** ali **dict** vsebuje povezave (asociacije) med ključi in vrednosti. Ključi so nizi, vrednosti pa so poljubni tipi. Serializiran je podobno kot seznam, le da se začne z znakom `d`. Ključi in vrednosti so prepleteni; prvi in nato vsak drugi element predstavlja ključe, vsakemu ključu sledeči podatek pa predstavlja vrednost pod tem ključem. Primer: `d4:testli-1337e18:pozdravljen, svet!lee6:zzzzzzd7:podpira9:gnezdenjeee`, ki bi ga v JSONu predstavili kot `{"test": [-1337, "pozdravljen, svet!", []], "zzzzzz": {"podpira": "gnezdenje"}}`. Za hitrejše iskanje morajo biti vrednosti sortirane glede na ključ.

2.2 Protokol BitTorrent

2.2.1 Datoteka torrent/metainfo

Ko neke datoteke avtor želi deliti s protokolom BitTorrent, ustvari torrent datoteko, ki je bkodiran slovar. S to datoteko drugim omogoči prenos, zato jim jo na nedefiniran. Glavni ključi v slovarju so (citiraj BEP):

- `announce`: URL sledilnika (za to nalogo brezpredmeten)

- `info`: informacije o datotekah v torrentu
  - `private`: za soležnike se sme spraševati le sledilnik in ne DHT (citiraj BEP 27)
  - `name`: ime torrenta, v primeru, da torrent vsebuje le eno datoteko, pa ime datoteke
  - `piece length`: velikost koščka. Datoteke so razdeljene na več enako velikih koščkov, da jih je moč neodvisno nalagati od drugih soležnikov. Če je datotek več, so zaporedno spojene skupaj in razdeljene na koščke, zato ena datoteka v prvi različici protokola ni vedno na mejah koščkov.
  - `pieces`: niz dolžine $20n$, kjer je n število koščkov. Za vsak košček je tu zapisana njegova zgoščena vrednost tipa SHA-1 (citiraj SHA-1)
  - `length`: dolžina torrenta, prisotna le, če torrent vsebuje eno datoteko
  - `files`: seznam datotek v torrentu, če je torrent večdatotečni. Vsaka datoteka je predstavljena kot slovar:
    - `length`: dolžina datoteke
    - `path`: pot do datoteke kot seznam imen direktorijev in na koncu ime datoteke, recimo `[„programi", „travnik", „src", „dht.c"]` predstavlja datoteko `programi/travnik/src/dht.c`

Namesto pošiljanja torrent datoteke lahko potencialnim soležnikom prenos omogočimo tudi tako, da jih o njenem obstoju obvestimo samo z zgoščeno vrednostjo slovarja `info` (infohash). Odjemalci s tem ključem napravijo poizvedbo po soležnikih v DHT in od njih prenesejo slovar `info`, ne pa tudi celotne datoteke, vendar slovar `info vsebuje vse potrebno za prenos datotek` (citiraj metadata transfer BEP 9). Zgoščena vrednost se običajno pošilja kot magnetna povezava, torej shematski zapis URI:

`magnet:?dn=ime torrenta&xt=urn:btih:infohash`

BitTorrent različica 2 ima drugačno strukturo, ki poda podobne podatke, vendar na malce spremenjen način. Uporablja recimo zgoščeno vrednost SHA256 in namesto ključa `pieces` hrani samo eno zgoščeno vrednost, po sistemu `merkle hash tree` (citiraj) pa pridobi še ostale med prejemom datotek, s čimer se korenito zmanjša velikost torrenta za velike datoteke.

2.2.2 Povezava na soležnike za prevzem metapodatkov

Če odjemalec želi od soležnika prejeti info slovar, se nanj poveže bodisi po µTP (citiraj µTP) bodisi po TCP. V eksperimentalnem delu se na soležnike povezujem po TCP, saj je to bolj preprosto. µTP sicer prinaša nove funkcije za bolj učinkovito rabo pasovne širine ob prenosu datotek, vendar to za prenos slovarjev info ni kritično, saj so sami po sebi relativno majhni.

Povezava po TCP za prevzem metapodatkov se začne z rokovanjem:

- Bajt 19, ki mu sledi niz `BitTorrent protocol`
- Osem rezerviranih bajtov 0, ki so na voljo za razširjanje protokola
- dvajsetbajtni infohash
- dvajsetbajtna unikatna identifikacijska številka odjemalca

Za rokovanjem sledi neskončno dolg pretok paketov. Pred sporočilom paketa je štiribajtna neoznačena velikoendianska številka, ki predstavlja dolžino sporočila. Sporočila dolžine 0 so t. i. *keepalive* sporočila, ki jih prejemnik ignorira. Paketi s sporočilom pa se začnejo z enobajtnim tipom sporočila, ki mu sledi vsebina sporočila, vezana na ta tip.

2.2.2.1 Razširitveni protokol

Prenos metapodatkov je opisan v standardu BEP-0009, vendar sam po sebi ne predstavlja številke tipa. Za uporabo prenosa metapodatkov je najprej treba vzpostaviti razširitveni protokol, ki odjemalcem omogoča dodajanje poljubnih protokolov v komunikacijo, ne da bi med njimi prišlo do nekompatibilnosti.

Paketi razširitvenega protokola (citiraj BEP 10) imajo številko tipa 20. Da sogovornika vesta, da lahko pošiljata razširitvene pakete, oba med rokovanjem nastavita 19. bit z desne v polju osmih rezerviranih bajtov. Drugi bajt sporočila (šesti bajt celega paketa) predstavlja podtip. Če je podtip 0, gre za razširitveno rokovanje — sogovornik pove, katere razširitve podpira — v tem primeru bo preostanek sporočila bkodirana struktura:

```
{„m": {„ut_metadata": 1}, „v": „program odjemalca", „metadata_size": 69420}
```

Slovar `m` poda prevod z nizi poimenovanih dodatkov v številke. Soležnik ob prejemu tega paketa ve, da lahko pakete tipa `ut_metadata` pošilja sogovorniku tako, da podtip razširitvenega paketa nastavi na 1 in v preostanek sporočila vstavi telo protokola `ut_metadata`.

2.2.2.2 `Prevzem metapodatkov`

Slovar metadata se konceptualno razdeli na delčke velikosti 16384 bajtov (zadnji delček je lahko manjši), soležnik posamezen delček zahteva s paketom (citiraj BEP 9):

- 4 bajtna dolžina sledečih polj
- bajt 20
- bajt vrednosti, kakršno je dobil v `m` slovarju od soležnika pod ključem `ut_metadata`

- bkodiran slovar `{„msg_type": 0, „piece": 5}`, kjer 5 predstavlja številko delčka, ki ga zahteva, tip 0 pa predstavlja zahtevo

Sogovornik lahko bodisi odgovori z zavrnitvijo oblike `{„msg_type": 2, „piece": 5}`, če nima vseh delčkov (za preverjanje zgoščene vrednosti slovarja info je potrebno poznavanje vseh koščkov), bodisi odgovori s paketom

- 4 bajtna dolžina sledečih polj
- bajt 20
- bajt vrednosti, kakršno je dobil v `m` slovarju od soležnika pod ključem `ut_metadata`
- bkodiran slovar `{„msg_type": 1, „piece": 5, „total_size": 69420}`, kjer 5 predstavlja številko delčka, ki ga pošilja, tip 1 predstavlja podatke, 69420 pa je celotna dolžina slovarja info.
- bajti delčka bkodiranega slovarja info

Preden lahko odjemalec metapodatke uporabi (torej pošilja naprej ali začne s prenosom torrenta), mora prenesti vse delčke in preveriti veljavnost zgoščene vrednosti. Če gre za BitTorrent različice 1, je ta zgoščena vrednost SHA-1, če pa gre za BitTorrent različice 2, je zgoščena vrednost SHA-256 (citiraj BEP bittorrent v2).

2.3 Protokol BitTorrent DHT

Naloga protokola DHT, standardiziranega 31. januarja 2008 v standardu BEP-0005, je vzdrževanje seznama soležnikov v roju vseh obstoječih torrentov, ki obstajajo in niso zasebni (več o tem v uvodu).

Komunikacija med vozlišči poteka izključno po protokolu UDP v obliki bkodiranih slovarjev.

2.3.1 Sestava grafa

Povezave med vozlišči si predstavljajmo kot velik usmerjen graf. Vsako vozlišče ima približno $K \log_2 n$ (konstanta $K = 8$, n je število vseh vozlišč na svetu) povezav na druga vozlišča, ki jih hrani v svoji lastni usmerjevalni tabeli, ki vsebuje IP naslov in vrata vozlišč ter njihove IDje. ID vozlišča si vsako vozlišče ob prvem zagonu izmisli naključno. S tem je zagotovljena homogena porazdelitev vozlišč po spektru možnih IDjev.

Ko vozlišče izve za novo vozlišče, s katerim lahko komunicira[1], ga zapiše v svojo usmerjevalno tabelo, če je v košu, v katerega to vozlišče spada, dovolj prostora. Za vsako vozlišče implementacije hranijo tudi čas zadnjega odgovora na paket. Vozlišča, ki se nekaj minut ne oglasijo na poizvedbe, se iz tabele odstrani.

[1] torej mu na poizvedbe odgovarja, kar zaradi obstoja NAT in požarnih zidov ni samoumevno

Koši so definirani kot skupki največ osmih vozlišč. Ko program želi vstaviti novo vozlišče v usmerjevalno tabelo, preveri, če ima koš, v katerega to vozlišče spada, prostor. V kolikor je v košu prostor, shrani vozlišče, v nasprotnem primeru pa preveri, če tako ID novega vozlišča kot tudi ID sebe pripadata v isti koš[2]; v tem primeru ta koš razpolovi na dva dela, da lahko vanj vstavi novo vozlišče. Če noben izmed teh dveh pogojev ne vstavi najdenega vozlišča v usmerjevalno tabelo, je vozlišče bodisi zavrženo bodisi vstavljeno v predpomnilnik, da bo vstavljeno v prihodnje.

Program začne z enim košem, ki bo hranil vozlišča z identifikacijskimi številkami od 00...00 do ff...ff. Razpolovitev koša v takem stanju bi iz začetnega koša izdelala dva koša s ključi od 00...00 do 7f...ff ter od 80...00 do ff...ff. Druga razpolovitev se lahko izvede le na enem izmed teh dveh košev, na tistem namreč, katerega naslovno območje zavzema ID vozlišča tega programa. Nadaljnje razpolovitve vodijo v stanje, kjer je $\log_2 n$ košev, vsak koš pa predstavlja podmnožico vseh možnih IDjev z močjo 2^{160-i}, kjer je i indeks koša od 1 do $\log_2 n$. Ker vsak koš vsebuje le K vozlišč in ker je zaradi algoritma razpolavljanja košev največ košev okoli IDja trenutnega vozlišča, so v usmerjevalni tabeli tega vozlišča najbolj reprezentirana vozlišča, katerih ID je podoben IDju trenutnega vozlišča.

2.3.2 Komunikacija in izvajanje poizvedb

Da se program prvič poveže v omrežje, mora najprej najti vsaj enega člana omrežja. Algoritem za povezavo v omrežje ni definiran. Implementacije ob izhodu iz programa usmerjevalno tabelo shranijo na disk, da ob ponovnem zagonu vsaj nekaj vozlišč iz prejšnega zagona še deluje. Če se nobeno vozlišče ne odzove, vpraša centraliziran strežnik, t. i *bootstrap node*, ki hrani podatke o veliki količini vozlišč.

Za pridobivanje seznama soležnikov odjemalec v usmerjevalni tabeli poišče t infohashu najbližjih vozlišč (lahko tudi cel koš, v katerega spada infohash). Razdaljo definiramo kot operacijo XOR med infohashom in IDjem. Tem vozliščem pošlje paket tipa `get_peers`. Odgovor na ta paket je seznam soležnikov. V kolikor pa kontaktirano vozlišče ne pozna soležnikov, pa vrne seznam vozlišč iz njegove usmerjevalne tabele, ki so temu infohashu najbližje. Program za vsak torrent hrani v najbližjih vozlišč, ki jih vsake toliko časa kontaktira za nove soležnike in bližja vozlišča. Ob prejetju seznama vozlišč se torrent odjemalec vpiše kot soležnika in s tem doda v roj tako, da vozlišču pošlje paket tipa `announce_peer`.

Iskanje po DHT se torej obnaša kot iskanje po binarnem drevesu in ima kompleksnost $O(\log n)$.

2.3.2.1 Sestava paketa in osnovni tipi paketov

Paketi se pošiljajo po UDP. Celotna vsebina UDP paketa je bkodiran slovar (citiraj BEP 5). Paketi se delijo na zahteve in na odgovore, da pa vozlišče prejeto zahtevo

[2]Vozlišče sebe sicer nikoli ne shrani v usmerjevalno tabelo.

lahko poveže s poslanim odgovorom, pa vsi paketi vsebujejo ključ `t` s kratkim nizom bajtov, ki bo prepisan v odgovor. Ključ `y` v paketu predstavlja tip paketa, torej niz `q` za zahtevo, niz `r` za odgovor ali niz `e` za poročilo o napaki, slednje vsebuje standardizirano kodo napake in tekstovno sporočilo. Vozlišče lahko ime programa in različico predstavi s štiribajtnim nizom pod ključem `v`. Vsaka poizvedba ima pod ključem `a/id` (odgovor pa pod ključem `r/id`) zapisan ID pošiljatelja — tako je en `ping` paket dovolj, da vozlišče izve za novo vozlišče in ga potencialno vstavi v usmerjevalno tabelo.

Parametri zahteve so zapisani v slovarju pod ključem `a`, parametri odziva so pod ključem `r`, tip zahteve pa je kot niz naveden pod ključem `q`. Obstajajo štirje:

find_node Zahteva vsebuje ključ `target`, v katerem je dvajsetbajtni niz zgoščene vrednosti iskanega vozlišča. Odgovor pod ključem `nodes` vsebuje niz K vozlišč iz usmerjevalne tabele, katerih ID je najbližji iskanemu. Vozlišča si en za drugim sledijo v nizu, vsako pa je dolgo 26 znakov, 20 za ID, 4 za IP naslov in 2 za vrata. V primeru IPv6 je seveda dolžina enega vozlišča 38, ključ pa se imenuje `nodes6`.

get_peers Sistem je podoben ukazu `find_node`, le da je namesto parametra `target` podan parameter `infohash` z dvajsetbajtnim infohashom iskanega torrenta. Odgovor na paket lahko poleg `nodes` in `nodes6` vsebuje tudi `values`, seznam nizov, kjer vsak niz predstavlja IP naslov in vrata soležnika v roju, ter ključ `token`, pod katerim je zapisan niz, ki ga mora vozlišče, ki v seznam želi zapisati svoj naslov, napisati pod ključem `token` v paketu `announce`_peer.

announce_peer Vozlišče ga pošlje vozlišču, katerega ID je blizu infohasha torrenta, da se bodo nanj z BitTorrent protokolom povezali ostali soležniki v roju in prenašali koščke torrenta. Parametri zahteve so `port`, `info_hash` in token iz prej prejetega odgovora na `get_peers`. Žeton `token` poskrbi, da s ponarejanjem izvora UDP/IP paketov ne moremo v seznam vnesti drugih računalnikov, temveč le tistega, ki je prejel odgovor na `get_peers`. Vozlišče, ki paket prejme, podatke shrani v seznam soležnikov.

ping Poleg `id` zahteva in odgovor nimata dodatnih parametrov. Namenjen je preizkusu delovanja vozlišča s čim manjšo procesorsko obremenitvijo.

3 Eksperimentalni del

Namen raziskovalne naloge je prenesti čim več info slovarjev iz metainfo slovarjev/torrent datotek. V ta namen sem po standardih implementiral odjemalec BitTorrent, vendar nepopolno, le do te mere, da zna sodelovati v DHT in prenašati metapodatke.

3.1 Program travnik

Program travnik je spisan v programskem jeziku C in sestoji iz več komponent, ki se med seboj povezujejo kot t. i. *single-header* knjižnice, na koncu pa se povežejo v programsko datoteko, ki se ob zagonu poveže v DHT mrežo in v njej prenese en torrent ter prestreže vse infohashe torrentov, za katere dobi poizvedbe `get_peers`. Najdene infohashe doda v seznam torrentov, za katere bo poizkušal prejeti soležnike, ko soležnike prejme, pa enega za drugim sprašuje za metapodatke. Ko metapodatke enkrat prenese, jih ne za torrent ne bo več prenašal.

Izdelani program ne implementira možnosti oddajanja metapodatkov, omogoča pa shranjevanje in še vedno deluje kot veljavno DHT vozlišče.

Izvorna koda programa je dostopna na http://ni.šijanec.eu./sijanec/travnik/.

3.1.1 Implementacija bkodiranja (`src/bencoding.c`)

Za dekodiranje in enkodiranje bkodiranih objektov sem spisal v C spisal knjižnico, ki bencoding objekte dekodira v objektno strukturo, na kateri omogoči osnovne operacije, kot so iskanje ključev, zanka preko celotnega seznama ali slovarja, vstavljanje novih elementov, brisanje elementov ter dupliciranje elementov. Deserializirana oblika je drevo elementov strukture bencoding:

```
struct bencoding {
        struct bencoding * next;
        struct bencoding * prev;
        struct bencoding * child;
        struct bencoding * parent;
        enum benc type;
        struct bencoding * key;
        char * value;
```

```
        size_t valuelen;
        long int intvalue;
        int index;
        unsigned seqnr;
        const char * after; /**< zaseben atribut */
}
```

Za izdelavo in prevajanje med oblikami so med drugim na voljo sledeče funkcije:

- za deserializacijo v drevo elementov je implementirana funkcija `struct bencoding * bdecode (const char * vir, int len, enum benc opts)`, ki sezname in slovarje bere z rekurzivnim klicem
- za serializacijo v bencoding funkcija `char * bencode (char * dest, struct bencoding * b)`
- `char * b2json (char * dest, struct bencding * b)` za serializacijo v JSON za namene razhroščevanja in obdelave podatkov.
  JSON sicer ne more popolnoma reprezentirati podatkov, ki jih reprezentira bkodiranje, saj morajo biti vsi nizi v obliki UTF-8, česar bencoding ne zagotavlja (tam so lahko v nizih poljubni bajti). Kljub temu pa obstajajo JSON bralniki, ki podpirajo poljubne bajte v nizih.

Za urejanje in branje obstoječih bencoding dreves so med drugim na voljo sledeče funkcije:

- `struct bencoding * bstr (char * str)`, ki izdela bencoding niz iz Cjevskega
- `struct bencoding * bnum (long nr)`, ki izdela bencoding število iz Cjevskega
- `void binsert (struct bencoding * benc, struct bencoding * elem)`, ki vstavi nov element v slovar/seznam
- `void bdetach (struct bencoding * elem)`, ki brez uničenja odstrani element iz slovarja/seznama
- `struct bencoding * bpath (const struct bencoding * benc, const char * key)`, ki vrne bencoding element na ključu, ki je podan kot niz (recimo `r/nodes6`)
- `bforeach(list, elem) {}` kontrolna struktura (makro), ki izvede blok kode za vsak element seznama/slovarja
- `struct bencoding * bval (struct bencoding * benc, struct bencoding * val)`, ki najde vrednost v slovarju/seznamu glede na njeno vrednost
- `struct bencoding * bclone (struct bencoding * b)`, ki duplicira bencoding drevo

3.1.2 Implementacija DHT (`src/dht.c`)

Celotno povezovanje z vozlišči je spisano v knjižnici za DHT. Ta opiše več struktur in operacij z njimi. Ureja povezovanje na DHT vozlišča in tudi TCP za prenos metapodatkov. Vzpostavi eno UDP vtičnico, preko katere komunicira s svetom. Z bkodiranim seznamom, ki ga uporabnik knjižnice shrani na disk, je omogočena tudi obstojna shramba podatkov, da lahko od zagona do zagona DHT ohranja usmerjevalno tabelo, številko vrat in ID vozlišča.

Mišljeno je, da program deluje z eno nitjo, zato je knjižnica izdelana tako, da se koda izvaja periodično in da knjižnica nikoli ne ustavi izvajanja s sistemskim klicem, temveč se poslužuje zunanjega `poll(2)` klica v dogodkovni zanki.

3.1.2.1 Podatkovne strukture

V tej rubriki so navedene le podatkovne strukture, ki so namenjene uporabniku, ne strukture interne implementacije knjižnice, ker bi jih bilo preveč.

Za razliko od tradicionalne implementacije `dht.c`, ki jo uporablja velik del obstoječih BitTorrent odjemalcev, je ta knjižnica popolnoma samostojna/brez stanja, v smislu da ne uporablja globalnih spremenljivk in lahko v enem procesu obstaja večkrat. Oprimek (angl. *handle*) knjižnice je kazalec na Cjevsko strukturo:

```
struct dht {
        unsigned char id[20]; // ID vozlišča
        int socket; // vtičnica za UDP komunikacijo
        unsigned char secret[16]; // AES ključ za announce žeton
        FILE * log; // stdio za dnevnik
        struct bucket * buckets; // shramba košev
        struct bucket * buckets6; // shramba košev IPv6
        struct torrent * torrents; // shramba torrentov
        void (* possible_torrent)(struct dht *,
              const unsigned char *, struct torrent *);
        void * userdata;
        unsigned torrents_num;
        unsigned peers_num;
        unsigned peers_max;
        struct torrent * last_torrent;
        unsigned peers_per_torrent_max;
        unsigned time; // čas zagona
        unsigned rxp; // prejetih paketov
        unsigned txp; // poslanih paketov
        unsigned rxb; // prejetih bajtov
        unsigned txb; // poslanih bajtov
        unsigned tcp_max; // omejitev TCP povezav
```

```
        void (* possible_torrent)(struct dht *,
                const unsigned char *, struct torrent *);
        unsigned tt; // poslanih bajtov po TCP
        unsigned tr; // prejetih bajtov po TCP
        unsigned p; // število poslanih pingov
        struct sockaddr_in6 pings[PINGS_CAP];
        unsigned periods; // število klicev periodic()
        unsigned rxqp; // prejetih zahtev
        unsigned txqp; // prejetih zahtev
        unsigned rxrp; // prejetih odzivov
        unsigned txrp; // poslanih odzivov
        unsigned removed_torrents;
};
```

Torrent je reprezentiran v strukturi `torrent`. Ker je vsak torrent lahko povezan na enega soležnika hkrati, struktura vsebuje tudi atribute soležnika:

```
struct torrent {
        unsigned char ut_metadata; // če soležnik podpira
        unsigned char ut_pex; // če soležnik podpira
        enum state state;
        int socket; // TCP vtičnica do soležnika oz. -1
        void * userdata;
        void (* disconnection)(struct torrent *);
        struct peer * dl; // povezani soležnik oz. NULL
        time_t time; // začetek prenosa metapodatkov
        enum interested type; // announce, peers, info
        unsigned char hash[20]; // infohash
        struct peer * peers;
        struct node * nodes;
        struct torrent * next;
        struct torrent * prev;
        int progress; // število prenesenih delčkov
        int size; // velikost metapodatkov v bajtih
        unsigned char * metadata; // metapodatki, ki se nalagajo
        void (* intentions)(struct torrent *);
        unsigned char * packet; // paket, ki se še sestavlja
        int recvd; // število pridobljenih bajtov paketa
        char * software; // ime programa, ki teče na soležniku
        time_t ttl; // čas, po katerem naj obupam s prenosom
        unsigned canary; // interni atribut za razhroščevanje
};
```

3.1.2.2 Funkcije

Uporabniku knjižnice so med drugim namenjene slednje funkcije:

- `struct torrent * add_torrent (struct dht * d, struct torrent * t)`, ki doda torrent v shrambo torrentov. Praviloma uporabnik torrentu nastavi type na `peers|info`.
- `struct bencoding * persistent (const struct dht * d)`, ki vrne bkodiran slovar, ki naj ga uporabnik ob naslednjem zagonu posreduje knjižnici
- `void work (struct dht * d)`, ki naj jo uporabnik pokliče, ko `poll(2)` pove, da je možno brati na UDP vtičnici
- `void tcp_work (struct dht * d)`, ki naj jo uporabnik pokliče, ko `poll(2)` pove, da je možno brati na TCP vtičnici
- `void periodic (struct dht * d)`, ki naj jo uporabnik pokliče vsakih 10 sekund, da se v DHT pošljejo poizvedbe o torrentih in začnejo povezave za prenos metapodatkov.

Poleg tega mora uporabnik skrbeti še za povratne klice (angl *callback*):

- `void possible_torrent (struct dht *, const unsigned char *, struct torrent *)`, ki uporabnika obvesti o najdenem infohashu v dohodni `get_peers` zahtevi. Uporabnik v tej funkciji nov torrent doda med torrente in zahteva prenos metapodatkov.
- `void connection (struct dht *, struct torrent *)`, ki uporabnika obvesti, da naj v `poll(2)` klicu gleda novo vtičnico `torrent->socket`.
- `void disconnection (struct torrent *)`, ki uporabnika obvesti, da mora prenehati opazovati vtičnico `torrent->socket` v `poll(2)` klicu. Uporabnik v tej funkciji tudi shrani metapodatke na disk, saj niso dostopni ne prej ne kasneje.
- `void intentions (struct torrent *)`, ki uporabnika obvesti o željeni dejavnosti na TCP vtičnici (branje/pisanje), ki jo prebere iz `torrent->state`

3.1.3 Servisni programi

Za razhroščevanje in uporabo travnika sta priložena še dva samostoječa programa. `utils/bencoding.c` omogoča pretvorbo med bkodiranjem in JSONom ter omejeno urejanje bencoding struktur iz ukazne vrstice preko standardnega vhoda in izhoda, `utils/info.c` pa omogoča prenos metapodatkov torrenta s podanim infohashom iz specifičnega naslova IP in vrat.

3.2 Algoritem prestrezanja podatkov

Vedno, ko program zazna novo infohash, ga doda med torrente, katerih metapodatke želi prenesti. Vsak cikel desetih sekund (klic funkcije `periodic`), bo program poiskal soležnike in vozlišča, ki so blizu temu infohashu. Nato se bo za vsak torrent poizkusil povezati na enega izmed soležnikov, na katega se še ni povezal, ter izvedel protokol, opisan v razdelku 2.2.2. Medtem knjižnica konstantno izvaja povratne klice in spreminja stanje vtičnice v klicu poll, ker za komunikacijo pričakuje bodisi zmožnost branja bodisi zmožnost pisanja v vtičnico.

Ko je slovar info prenesen in se infohash torrenta ujema z bodisi prvimi dvajsetimi bajti zgoščene vrednosti SHA-256 bodisi zgoščeno vrednostjo SHA-1, se torrent shrani v datoteko v trenutni direktorij ter odstrani zahteva po nadaljnjem pridobivanju soležnikov in prenosu metapodatkov.

Tako se bo v trenutnem direktoriju sproti nabralo veliko `infohash.torrent` datotek.

Da program prvič začne sodelovati z omrežjem, torej da ga sosednja vozlišča vpišejo v svoje usmerjevalne tabele, prenese metapodatke vgrajenega torrenta `Big Buck Bunny`.

3.3 Obdelava podatkov

Podatke sem sprva mislil obdelati tako, da bi jih shranjeval v relacijski podatkovni zbirki tipa MySQL, zato sem spisal PHP program, ki s knjižnico (citiraj Rhilip/Bencode) za razčlenjevanje metainfo datotek odpre vsako datoteko in jo vstavi v podatkovno zbirko s tabelama `torrenti` in `datoteke`. Z naraščajočim številom vrstic v podatkovni zbirki pa postanejo `JOIN` operacije med tabelo s torrenti in tabelo z datotekami prepočasno. Relacijske podatkovne zbirke namreč niso narejene za ta namen. Alternativa njim so podatkovne zbirke, ki delajo na nivoju t. i dokumentov.

Sprva sem mislil uporabiti podatkovno orodje MongoDB (citiraj), vendar mi je zaradi restriktivne licence in komercialno orientirane narave neprivlačen.

Za izvajanje preprostih iskanj z regularnimi izrazi (angl. *regular expression/RegEx*) po celotnem seznamu datotek/torrentov, ki imajo tako ali tako O(n) kompleksnost, je v mojem primeru dovolj, če vse torrente hranim kot Pythonski `dict` (slovar). Za ohranitev tega slovarja v delovnem spominu za daljše časovno obdobje in za uporaben uporabniški vmesnik sem izbral programsko orodje Jupyter Notebook (citiraj), ki uporablja `ipython` REPL in lahko znotraj dokumenta izrisuje grafe in ostale diagrame.

Python uporabne knjižnice, ki podpira obe obliki torrent datotek, nima, zato sem preprost vmesnik za razčlenjevanje datotek v objektne strukture spisal sam (Priloga: 7.1).

4 Rezultati

4.1 Analiza podatkov

5 Razprava

5.1 Težave pri pridobivanju podatkov

5.1.1 Napad Sybil

Napad Sybil je pogosto možen v DHT omrežjih, ki za identifikacijske številke vozlišč ne izvajajo asimetrične kriptografije — izrazito je prisoten pri Kademili oz. BitTorrent Mainline DHT. Napad učinkovito omrtviči vozlišča — onemogoči vzpostavljanje povezav in zapolni usmerjevalno tabelo tako, da so v njej večinoma napadalčeva vozlišča. Napadalec iz enega ali več IP naslovov izdela veliko število virtualnih vozlišč, katerih IDji so zelo blizu vozlišča žrtve napada. Tako bo žrtev vedno vstavila napadalčeva vozlišča v usmerjevalno tabelo, saj bo vedno lahko razpolovila koš.

Usmerjevalna tabela žrtve ob uspešnem napadu izgleda takole:

```
BUCKET id=449a918e2f5d0eaffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb7ffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb8ffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb97ffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9bffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c1ffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c2ffffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c31ffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c32ffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c337ffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33bffffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33cffffffffffffffffffffff
449a918e2f5d0eb9c33d269d398f2b2b181f35ed :: ffff:82.156.184.234/6881 unans=0 good
BUCKET id=449a918e2f5d0eb9c33d3ffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d4ffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d53ffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d54ffffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d553fffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d555fffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d556fffffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5573fffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5575fffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d55767fffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576bfffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576cfffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d1ffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d2ffffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d31fffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d321fffffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d3227fffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322bfffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322cfffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322d3ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322d5ffffffffff
449a918e2f5d0eb9c33d5576d322d6050db58f35 :: ffff:184.72.202.243/6881 unans=0 good
449a918e2f5d0eb9c33d5576d322d63a050f460b :: ffff:192.241.151.29/6882 unans=0 questionable
449a918e2f5d0eb9c33d5576d322d63e82e28652 :: ffff:34.89.135.129/6881 unans=0 good
449a918e2f5d0eb9c33d5576d322d652d63f0de1 :: ffff:85.10.202.14/6881 unans=0 questionable
449a918e2f5d0eb9c33d5576d322d67444e8eb27 :: ffff:64.235.252.215/6881 unans=0 good
BUCKET id=449a918e2f5d0eb9c33d5576d322d67fffffffff
449a918e2f5d0eb9c33d5576d322d688a2f0eaab :: ffff:192.241.151.29/6883 unans=0 questionable
449a918e2f5d0eb9c33d5576d322d6c69af7ebe7 :: ffff:220.191.18.238/6881 unans=0 good
449a918e2f5d0eb9c33d5576d322d6cdfede2698 :: ffff:65.108.201.176/56881 unans=0 good
449a918e2f5d0eb9c33d5576d322d6cdfede5aa4 :: ffff:106.255.239.227/6688 unans=0 questionable
449a918e2f5d0eb9c33d5576d322d6cdfede7776 :: ffff:123.173.71.216/6688 unans=0 good
449a918e2f5d0eb9c33d5576d322d6cdfedeabdc :: ffff:52.214.147.108/6971 unans=0 good
```

```
449a918e2f5d0eb9c33d5576d322d6cdfeded216 :: ffff:136.243.96.42/1688 unans=0 good
449a918e2f5d0eb9c33d5576d322d6cfc5eba6c7 :: ffff:35.232.31.198/6881 unans=0 good
BUCKET id=449a918e2f5d0eb9c33d5576d322d6fffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322d7fffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322dfffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d322ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d323ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d327ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d32ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d33ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d37ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d3ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576d7ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576dffffffffff
BUCKET id=449a918e2f5d0eb9c33d5576ffffffffff
BUCKET id=449a918e2f5d0eb9c33d5577ffffffffff
BUCKET id=449a918e2f5d0eb9c33d557fffffffffff
BUCKET id=449a918e2f5d0eb9c33d55fffffffffff
BUCKET id=449a918e2f5d0eb9c33d57fffffffffff
BUCKET id=449a918e2f5d0eb9c33d5fffffffffff
BUCKET id=449a918e2f5d0eb9c33d7fffffffffff
449a918e2f5d0eb9c33d9b1f68e7c9c199c75672 :: ffff:167.172.226.132/6060 unans=0 questionable
BUCKET id=449a918e2f5d0eb9c33dffffffffffffffff
BUCKET id=449a918e2f5d0eb9c33fffffffffffffffff
BUCKET id=449a918e2f5d0eb9c37fffffffffffffffff
BUCKET id=449a918e2f5d0eb9c3fffffffffffffffff
BUCKET id=449a918e2f5d0eb9c7fffffffffffffffff
BUCKET id=449a918e2f5d0eb9cfffffffffffffffffff
BUCKET id=449a918e2f5d0eb9dfffffffffffffffffff
```

5.1.1.1 Preventivni omilitveni ukrepi

- Vozlišče v usmerjevalno tabelo sprejme samo eno vozlišče iz enega IP naslova. Težava nastane pri IPv6, ko je dolžina predpone omrežja lahko zelo različna. Napadalci imajo lahko na voljo velik spekter naslovov, celo večji od /48, legitimni uporabniki pa imajo velikokrat naslovne prostore velikosti /128 (samo en naslov).
- Uporaba fiksnih prefiksov IDjev (BEP 0042) (citiraj), kjer morajo vozlišča uporabljati ID, ki se začne z IP naslovom, transformiranim skozi CRC32c funkcijo. Težava nastane, ko imajo napadalci spet dovolj velik naslovni prostor, da lahko pokrijejo vse predpone IDja. Poleg tega je to zgolj razširitev osnovnega DHT protokola, na katero se odjemalci ne morejo zanašati. Nepravilen ID je sicer lahko napadalec, lahko pa je tudi vozlišče, ki razširitve ni implementiral.
- Vsakemu vozlišču lahko pred vstavljanjem v usmerjevalno tabelo pošljemo ping paket, ki vsebuje drugačen ID, kot ga odjemalec sicer uporablja. Če v odgovoru na ping ID vozlišča ni enak, kot smo ga videli prej, pomeni, da je vozlišče zagotovo napadalec. Težava nastane, ko lahko sogovornik nas smatra kot napadalca, saj smo mu poslali pakete iz dveh različnih node IDjev (čeprav v teoriji ne smemo zaupati izvornemu naslovu prejetih UDP paketov).

5.1.1.2 Ublažitev posledic napada

travnik ima poleg omejitve največ enega vozlišča z enim IP naslovom v usmerjevalni tabeli tudi protiukrep, ki prepreči zavrnitev storitve kot posledico napada Sybil. Protiukrep deluje tako, da v primeru, ko zazna, da ima shranjenih več kot 64 košev, izbriše skoraj celotno usmerjevalno tabelo in se še enkrat sinhronizira z omrežjem z novim IDjem, v upanju, da napadalec ne bo napadel še enkrat.

5.1.2 Slaba zmogljivost mrežne opreme

Ker se ob normalnem delovanju travnika prenese do 2000 paketov z različnimi IP naslovi na sekundo, slaba omrežna oprema kljub majhni porabljeni pasovni širini (okoli 4 megabite na sekundo) začne delovati slabo. Ključen primer je bil domači optični modem, ki med delovanjem travnika burno izgublja pakete do te mere, da prihaja do izpadov razreševanja internetnih imen (DNS). Problem sem omilil (na 2000 paketov/s) tako, da nov najden infohash dodam med željene torrente največ enkrat na dve sekundi in omejim čas življenja torrenta (koliko časa za torrent aktivno iščem soležnike, preden ga izbrišem) na 256 sekund. Seveda to zaradi velike količine torrentov, za katere nikoli ne dobim metapodatkov, precej zmanjša število prejetih torrentov.

5.2 Uporabna vrednost korpusa prenesenih podatkov

Podatki predstavljajo vzorec populacije torrentov, ki se pretakajo po internetu. Vsak prenesen torrent je poleg metapodatkov o datotekah označen še s časom prejema, programsko opremo in različico odjemalca, ki je torrent poslal, ter IP naslovom pošiljatelja. Glede na te informacije je možno analizirati stanje BitTorrent omrežja skozi čas, ugotoviti, za kakšne namene se uporablja (kakšne vsebine se pretakajo z njim), kateri programi/države prevladujejo, kakšni podatkovni tipi datotek so najbolj pogosti itd.

5.3 Etičnost in legitimnost rudarjenja podatkov

Čeprav gre za izrazito osebne podatke, se morajo uporabniki BitTorrent omrežja zavedati, da so njihovi prenosi *a priori* javni, tudi če jih nihče aktivno ne prenaša. Nekateri BitTorrent odjemalci uporabnike ob prvem zagonu o tem celo obvestijo, med delovanjem pa celo prikazujejo IP naslove soležnikov, na katere se povezujejo. Uporabniki se zato zavedajo, da je njihova identiteta drugim članom roja znana. Pogosto pa se ne zavedajo, da se obstoječe roje da odkriti in se jim pridružiti. (vstavi sliko)

5.4 Invazivnost v omrežje

Implementacija za to raziskavo je delovala neinvazivno, saj je implementirana tako, kot bi bil implementiran navaden torrent odjemalec, le da zahteve pošilja hitreje. Ne poslužuje se bolj invazivnih taktik, ki posegajo v omrežje, kot je npr. napad Sybil (citiraj), in dosledno shranjuje in daje drugim na razpolago informacije o soležnikih.

Program bi bil lahko manj invaziven, če bi namesto `find_nodes` pošiljal `ping` zahteve, ko bi bilo to ustrezno. `find_nodes` se vseeno uporablja, da se z enim paketom pridobi čim več informacij o vozliščih.

5.5 Vzorčenje ključev

Vzorčenje ključev, opisano v protokolu BEP-xxxx (citiraj), ni bilo uporabljeno, ker ga ne podpirajo vse implementacije BitTorrent DHT protokola. S pošiljanjem teh zahtev bi kljub temu vzorec pridobljenih torrentov obsegal enako reprezentativen delež prenesenih torrentov na internetu, saj so vozlišča, ki podpirajo ta protokol, zaradi naključnih IDjev homogeno razpršena po naslovnem prostoru.

6 Zaključek

Raziskovana naloga predstavi kako je praktično mogoče preprosto implementirati učinkovito metodo za pridobivanje izvlečka metapodatkov iz omrežja BitTorrent. Prav tako je prikazana uporabna vrednost korpusa prenesenih podatkov za nadaljne raziskave in osnovne metode analize takih podatkov ter preprost iskalnik po metapodatkih.

6.1 Načrti za prihodnost

- Implementirati travnik na večji količini strežnikov, ki nimajo težav z mrežno opremo in lahko pošiljajo več paketov na sekundo.
- Optimizirati travnik in ga prepisati v programski jezik z vgrajeno podporo za bolj učinkovite podatkovne strukture ter načrtovalne sposobnosti dogodkov.
- Izdelati program, ki stalno prenaša člane rojev, s čimer se odpre več analitičnih možnosti, med drugim:
  - popularnost torrentov skozi čas (glede na velikost roja) z implementacijo dodatka PEX
  - obstoj soležnikov v omrežju
  - boljša sposobnost relacije med IP naslovi odjemalcev in torrenti, ki jih prenašajo, za klasifikacijo interesnih skupin
- Izdelati učinkovit iskalnik ki z indeksiranjem besednih simbolov/žetonov omogoča hitro iskanje torrentov

Zahvala

Za pomoč pri obdelavi podatkov se zahvaljujem Oliverju Wagnerju (oliwerix.com.) in Adrianu Sebastianu Šiški (ass.si.).

7 Izvorna koda uporabljenih programov

Izvorna koda za nalogo spisanega programa `travnik` je objavljena na internetu na naslovu http://ni.sijanec.eu./sijanec/travnik. Vključitev vseh programov v prilogo zaradi obširnosti ni mogoča.

7.1 `travnik.py`: razčlenjevalnik `.torrent` datotek

```
from bencodepy import decode
from enum import Enum
from hashlib import sha1, sha256
from os import scandir
from re import search, IGNORECASE
class Type(Enum):
        UNDEF = 0,
        V1 = 1,
        V2 = 2,
        HYBRID = 3
class Torrent():
        def __init__(self):
                self.sha1 = b''
                self.files = {}
                self.type = Type.UNDEF
                self.cache = None
        def file(self, f):
                self.parse(open(f, "rb").read())
        def parse(self, b):
                infodict = b[b.find(b'4:info')+6:b.rfind(b'6:sourced2:ip')]
                self.sha1 = sha1(infodict).digest()
                self.sha256 = sha256(infodict).digest()
                self.dict = decode(b)
                if b'pieces' in self.dict.get(b'info'):
                        self.dict.get(b'info').pop(b'pieces')
                if b'files' in self.dict.get(b'info').keys():
                        self.type = Type.V1
                        for file in self.dict.get(b'info').get(b'files'):
                                if file.get(b'attr') is not None and b'p' in file.get(b'attr')
                                     or b'padding.file' in b'/'.join(file.get(b'path')) or b'.pad
                                     ' in file.get(b'path') or b'_____padding_file_' in b'/'.join
                                     (file.get(b'path')):
                                        continue
                                def insert_file(d, path, length, self):
                                        name = path.pop()
                                        if not len(path):
                                                d[name] = length
                                                return
                                        if name not in d.keys():
                                                d[name] = {}
                                        insert_file(d[name], path, length, self)
                                file.get(b'path').reverse()
                                insert_file(self.files, file.get(b'path'), file.get(b'length'),
                                     self)
                        self.dict.get(b'info').pop(b'files')
                if b'file tree' in self.dict.get(b'info').keys(): # some torrents have broken
                     file trees so we use files first
                        if self.type is Type.V1:
                                self.type = Type.HYBRID
                        else:
                                def filetree(names):
                                        r = {}
                                        for key in names.keys():
                                                if key == b'':
                                                        return names.get(key).get(b'length')
```

```
                                        r[key] = filetree(names.get(key))
                                return r
                        self.files = filetree(self.dict.get(b'info').get(b'file tree'))
                        self.dict.get(b'info').pop(b'file tree')
                if not len(self.files):
                        self.type = Type.V1
                        self.files[self.dict.get(b'info').get(b'name')] = self.dict.get(b'info')
                            .get(b'length')
                first_filename = [i for i in self.files.keys()][0]
                if len(self.files) == 1 and self.files[first_filename] == {}:
                        print("fixed bad single file torrent", self.sha1.hex())
                        self.files[first_filename] = self.dict.get(b'info').get(b'length')
        def paths(self):
                def paths_r(d, path=None):
                        if path is None:
                                path = []
                        for f in d.keys():
                                if type(d[f]) is int:
                                        z = path.copy()
                                        z.append(f)
                                        yield z, d[f]
                                else:
                                        z = path.copy()
                                        z.append(f)
                                        for z, v in paths_r(d[f], z):
                                                yield z, v
                for z, v in paths_r(self.files):
                        yield z, v
        def matches(self, r, cache=False):
                does = False
                if cache and self.cache:
                        return search(r, self.cache, IGNORECASE)
                try:
                        decoded = self.dict.get(b'info').get(b'name').decode()
                except UnicodeDecodeError:
                        decoded = self.dict.get(b'info').get(b'name').decode("iso-8859-2")
                except AttributeError:
                        decoded = str(self.dict.get(b'info').get(b'name'))
                if search(r, self.dict.get(b'source').get(b'ip').decode(), IGNORECASE):
                        does = True
                        if not cache:
                                return True
                if search(r, decoded, IGNORECASE):
                        does = True
                        if not cache:
                                return True
                if cache:
                        self.cache = self.dict.get(b'source').get(b'ip').decode() + "|" +
                            decoded + "|"
                for path, size in self.paths():
                        try:
                                decd = b'/'.join(path).decode()
                        except UnicodeDecodeError:
                                decd = b'/'.join(path).decode("iso-8859-2")
                        self.cache += decd + "|"
                        if search(r, decd, IGNORECASE):
                                does = True
                                if not cache:
                                        return True
                return does
        def matching_files(self, r, decode=False):
                def matching_files_r(dirc, r, decode):
                        files = {}
                        for name, content in dirc.items():
                                try:
                                        decoded = name.decode()
                                except UnicodeDecodeError:
                                        decoded = name.decode("iso-8859-2") # TODO we could try
                                            detecting the encoding
                                if search(r, decoded, IGNORECASE):
                                        files[decoded if decode else name] = content if type(
                                            content) is int else {}
                                if type(content) is dict:
                                        inhalt = matching_files_r(content, r, decode)
                                        if inhalt:
                                                files[decoded if decode else name] = inhalt
                        return files
                return matching_files_r(self.files, r, decode)
        def __repr__(self):
                return str(self.__dict__)
        def __hash__(self):
                if len(self.sha1):
                        return int.from_bytes(self.sha1, byteorder="big")
                return id(self)
def glob(d):
        r = {}
        for f in scandir(d):
                if f.name.endswith(".torrent") and f.is_file():
                        t = Torrent()
                        t.file(f.path)
```

```
                r[t.sha1] = t
    return r
```

Literatura