UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Ozren Demonja

REALIZACIJA P2P PROTOKOLA ZA DOSTAVU SINHRONIZOVANOG SADRŽAJA

master rad

Beograd, 2018.

Mentor:

dr Aleksandar KARTELJ, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Saša MALKOV, profesor
Univerzitet u Beogradu, Matematički fakultet

dr Miroslav MARIĆ, profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami i tati

Naslov master rada: Realizacija P2P protokola za dostavu sinhronizovanog sadržaja

Rezime: Popularnost deljenja video sadržaja putem Interneta beleži značajan rast. Sadržaj generisan na ovakav način je obiman. Tako obiman sadržaj emiter može efikasnije distribuirati upotrebom mreže ravnopravnih računara. Upravo zbog kvaliteta mreže ravnopravnih računara pri razmeni video sadržaja implementiran je veliki broj protokola. Kako je veliki broj protokola baziran na brzoj distribuciji sadržaja i malom dodatnom sadržaju, ne postoji ni jedan opšte prihvaćeni protokol za dostavu sinhronizovanog sadržaja. U ovom radu predstavljen je potpuno novi protokol Kikkar. Protokol koristi hibridnu topologiju i omogućava dostavu sinhronizovanog sadržaja upotrebom mreže ravnopravnih računara. Eksperimentalni rezultati pokazali su da je dodatni sabraćaj koji protokol unosi kao i brzina propagacije sadržaja kroz mrežu slična sa drugim protokolima koji ne podržavaju sinhronizaciju.

Ključne reči: P2P mreža, deljenje video sadržaja, deljenje sinhronizovanog sadržaja, P2P protokol, deljenje u realnom vremenu, decentralizovana razmena podataka

Sadržaj

Glava 1

Uvod

Svakodnevno se povećava broj dostupnih radio stanica i TV kanala koji se emituju putem Interneta. Tradicionalno se za njihov prenos koristi centralizovana klijent-server arhitektura. Ograničenja klijent-server arhitekture pri dostavi video sadržaja u realnom vremenu mogu se prevazići korišćenjem mreže ravnopravnih članova. Pored prevazilaženja ograničenja, mreža ravnopravnih članova omogućava emiteru da poveća kvalitet sadržaja i smanji cenu prenosa. Na taj način mreža ravnopravnih članova omogućava emitovanje ličnog medijskog sadržaja. Ovako široka dostupnost je dovela do revolucije pri distribuciji informacija na sličan način kao što je veb (eng. *World Wide Web, WWW*) pre više više od dve decenije.

Dobre karakteristike mreže ravnopravnih računara dugo su poznate i koriste se za dostavu video sadržaja u realnom vremenu. I pored široke primene, većina do sada poznatih sistema nema nikakav vid sinhronizacije. Kod ovakvog pristupa prikazani video u nekom vremenskom trenutku se može značajno razlikovati u zavisnosti od pozicije korisnika u mreži. Različitost u prikazu video sadržaja može izazvati širok spektar posledica, od gubljenja korisnika do nemogućnosti primene protokola u određenim uslovima u kojima je sinhronizacija bitan faktor.

U radu je predstavljen protokol za dostavu sinhronizovanog sadržaja u realnom vremenu. Predstavljeni protokol koristi topologiju koja je prvi put predstavljena Screamer protokolom i dodaje joj mogućnost dostave sinhronizovanog sadržaja u realnom vremenu sa malom zadrškom. Pored samog protokola rad obuhvata i implementaciju predloženog rešenja. Implementacija rešenja obuhvata dve aplikacije pisane u programskom jeziku Java. Radi kompletne slike predloženog protokola predstavljena je evaluacija dobijenog rešenja.

Glava 2 opisuje osnovne koncepte mreže ravnopravnih računara. Opisani su

ukratko različiti koncepti i diskutovano je o njihovim manama i prednostima uzimajući u obzir deljenje sadržaja. U glavi 3 predstavljen je Kikkar protokol. U glavi 4 detaljno je opisana implementacija predloženog Kikkar protokola. Implementacija protokola obuhvata dva Java programa, tragača i parnjaka. Glava 5 predstavlja eksperimentalne rezultate i evaluira dobijena rešenja. U glavi 6 izveden je zaključak i date su neke ideje za buduća unapređenja protokola.

1.1 Pregled literature

Tokom proteklih desetak godina mehanizmi za dostavu sadržaja u realnom vremenu upotrebom mreže ravnopravnih računara su detaljno izučavani [50]. Arhitektura koja se predlaže za sisteme za dostavu sadržaja u realnom vremenu obično se može svrstati u dve kategorije. To su topologija mreže sa povlačenjem sadržaja ili topologija stabla sa guranjem sadržaja. Prva predložena bila je topologija stabla sa guranjem sadržaja [43]. Topologija mreže sa povlačenjem sadržaja predstavljena je znatno kasnije [49]. Nakon uspeha opisanih topologija predstavljen je i njihov spoj, hibridna topologija. Početni radovi koji su upotrebljavali hibridnu topologiju imali su dobre performanse po ceni dosta više razmenjenih poruka [19].

U poslednjih par godina istražuju se tehnologije koje bi ubrzale razmenu sadržaja. Jedna od takvih tehnologija koja se često pominje u novijim radovima je slučajno mrežno kodiranje (eng. *random network coding*) [31]. Ova tehnologija je interesantna u naučnim krugovima. Sa druge strane, u industriji još uvek nije zaživela jer njena implementacija unosi značajan dodatni posao svakom parnjaku. Pored dodatnog posla povećava se i kompleksnost samog slanja podatka. Još jedna tehnika koja izaziva dosta polemike u naučnim krugovima je upotreba nekog vida identifikacije parnjaka koji mogu doprineti mreži i njihovo isticanje. Jedno od takvih rešenja predstavljeno je u radu [15]. Nažalost, ni ovo rešenje nije zaživelo zbog nepredvidivosti parnjaka koje se ogleda u čestom napuštanju mreže i dinamičkom menjanju resursa.

Kada se govori o sistemima za dostavu sinhronizovanog sadržaja u realnom vremenu nema značajnog pomaka u industriji i nauci. I pored značajnog interesovanja za ovakve sisteme autor ovog teksta nije uspeo pronaći ni jedan sistem za sinhronizovanu dostavu sadržaja sa korišćenjem mreže ravnopravnih članova za deljenje sadržaja. Svi takvi sistemi nikad nisu implementirani ili su implementirani samo konceptualno tako da njihova rešenja nisu proverena. Ipak, neke od ideja kao što su

optimalne veličine paketa i način sinhronizacije jata su uzete iz rada [48].

Prva analiza mreže ravnopravnih računara prezentovana je u radu [39]. Analizirane su performanse propagacije kašnjenja primenom topologije drveta. Prvi matematički model koji je opisivao topologiju mreže sa povlačenjem sadržaja opisan je u radu [9].

Glava 2

P2P mreža

Osnovna svrha interneta kao globalne mreže u današnje vreme je razmena digitalnog sadržaja. Arhitektura koja se koristi za podršku razmeni često je zasnovana na modelu klijent-server (eng. *Client-Server architecture*). Model klijent-server deli zadatke između servera koji obezbeđuju resurse ili usluge i klijenata koji zahtevaju usluge [12].

Kako svaki pristup sistemu od strane klijenta zahteva veću upotrebu resursa i bolje performanse, jasno se uočava da je server usko grlo sistema. Tokom 1990-ih godina internet se najvećim delom zasnivao na klijent-server modelu. Poslednjih desetak godina dolazi do velikog proboja novih tehnologija usled želje korisnika za interaktivnijim sadržajem, porastom brzine pristupa internetu (eng. *broadband*), rasprostranjenosti pristupa, boljom stabilnosti veze i jačim računarima korisnika.

Navedene činjenice dovode do promene načina korišćenja interneta i pravljenja mreže ravnopravnih članova (eng. *peer-to-peer (P2P) network*) u kojoj korisnici dele svoje resurse [46]. Primena ovakvog pristupa obećavajuće je rešenje za mnoge oblasti.

U poglavlju 2.1 opisani su osnovni koncepti mreže ravnopravnih članova. U poglavlju 2.2 opisane su struktuirane i nestruktuirane mreže ravnopravnih članova. U poglavlju 2.3 opisane su topologija stabla (eng. *tree topology*), mrežna (eng. *mesh topology*) i hibridna (eng. *hybrid topology*) topologija. Način pripreme i slanja video sadržaja opisan je u poglavlju 2.4. U poglavlju 2.5 opisani su osnovni problemi prilikom pravljenja mreže ravnopravnih članova.

2.1 Osnovni koncepti

Korišćenje aplikativnog sloja u formiranju pokrivača (eng. *overlays*) za pristup internet servisima ima dugu istoriju. I pored duge tradicije ovakav pristup se do nedavno koristio samo za dizajn specifičnih protokola i interno povezivanje servisa infrastrukture, bez mogućnosti korišćenja između dva računara koja nisu susedi. Popularizacija ideje za formiranje pokrivača na aplikativnom sloju počinje 1999. godine sa Napster-om [1] i nastavlja se kroz druge sisteme za razmenu digitalnog sadržaja kao što su Gnutella [8], FastTrack [16], KaZaa [27] i BitTorrent [6].

Arhitektura ravnopravnih članova (eng. *peer-to-peer, P2P*) definiše se kao distribuirani vid arhitekture u kojoj svaki parnjak (eng. *peer*) može samostalno da obavlja poslove, ali omogućava drugim parnjacima da koriste njegove resurse kao što i sam može po potrebi koristiti resurse drugih parnjaka [38]. Drugim rečima, svaki parnjak komunicira sa ostalim parnjacima u mreži koristeći softver koji se ponaša i kao klijent i kao server. Na osnovu navedenih činjenica zaključuje se da arhitektura ravnopravnih članova omogućava veću autonomiju članova mreže koji sami definišu pravila o deljenju resursa. Glavna primena arhitekture ravnopravnih članova prisutna je u sistemima sa većom tolerancijom greške kod distribuirane obrade. Pored toga, ovaj vid arhitekture koristi se prilikom razmene fajlova, direktne komunikacije, obrade velikih količina podataka i softvera za zabavu.

Prilikom deljenja resursa svaki parnjak doprinosi resursima mreže ravnopravnih članova. Idealno, deljenje resursa je proporcionalno broju parnjaka koji koriste mrežu ravnopravnih članova. Međutim, usled različitih problema u praksi navedeno svojstvo većinom nije ispunjeno.

P2P sistem čini skup međusobno povezanih parnjaka, direktno ili preko drugih parnjaka formirajući povezani graf. Kako u grafu ne postoji centralna tačka kontrole i kolektivno se obavlja posao, P2P sistemi su decentralizovani. U nekim situacijama definiše se centralni server za poslove kao što su identifikacija, autentikacija i različite sigurnosne provere. U ovakvim uslovima, po pravilu server obavlja što je moguće manji deo posla. Pored toga, ponekad se u P2P sistemima iz različitih razloga uvode parnjaci zaduženi za dodatni deo posla. Ovakvi parnjaci se nazivaju super parnjaci (eng. *super peers*).

Učešće parnjaka u obavljanju posla u P2P sistemima određeno je lokalno. Iz ove činjenice proizilazi bolja organizacija P2P sistema vremenom usled korišćenja lokalnih znanja i operacija na svakom parnjaku. Ovim se takođe sprečava neplanirana

dominacija nekog parnjaka sistemom.

Dva važna svojstva P2P sistema su samoskalabilnost i stabilnost. Samoskalabilnost, kao glavna prednost P2P sistema, ispoljava se povećanjem ukupne snage sistema prilikom pristupanja parnjaka. Stabilnost mreže se obično definiše kao njena otpornost na veliku stopu promene strukture mreže usled čestog dolaska i odlaska parnjaka. U literaturi ove promene se nazivaju talasanja (eng. *churn*). P2P sistem bi trebao biti stabilan do određenog nivoa talasanja, u kontekstu održavanja grafa konekcije i upućivanja (eng. *route*) sadržaja deterministički sa prihvatljivim granicama broja skokova (eng. *hop-count bounds*).

P2P sistem, kao tehnologija u zamahu, privlači veliku pažnju običnih i korporativnih korisnika. Takođe, pored pažnje korisnika, značajno interesovanje beleži se i u istraživačkim krugovima [12]. U poslednjih nekoliko godina, veliki deo istraživanja bavi se P2P deljenjem resursa i prenosom podataka.

2.2 Klasifikacija P2P mreža

Usled velikog broja kriterijuma dizajniranja P2P mreža, ne postoji jedinstven način klasifikacije [38]. Na primer, sistemi za deljenje fajlova često se klasifikuju po generacijama. Prvu generaciju karakteriše hibridni dizajn. Glavnu karakteristiku ove generacije predstavlja kombinacija servera sa P2P upućivanjem sadržaja. Druga generacija je zasnovana na decentralizovanoj arhitekturi. Trećom generacijom smatraju se anonimni P2P sistemi kao što su Freenet [10] i I2P [21]. Ovakav način kategorizacije prema generacijama ima značajnih nedostataka. Glavni nedostatak ogleda se u nedefinisanim važnim dimenzijama problema i nejasnoći dodatnih mogućnosti sledeće generacije. Takođe, sistemi sve tri generacije su korišćeni u isto vreme što se ne može zaključiti na osnovu njihovih imena. Zbog navedenih nedostataka, u literaturi se češće sreće podela prema P2P pokrivaču. Prema ovoj podeli, razlikujemo struktuiranu i nestruktuiranu P2P mrežu [17].

U struktuiranim P2P mrežama topologija je precizno definisana. Takođe, neophodna je striktna kontrola protoka sadržaja. Sadržaj se ne sme nalaziti kod proizvoljnih parnjaka već mora biti na strogo definisanim lokacijama koje će omogućiti efikasniju pretragu. Većina struktuiranih P2P mreža je bazirana na distribuiranim heš tabelama (eng. *Distributed Hash Table (DHT)*). Kao primere struktuiranih P2P mreža navodimo Tapestry [51], Chord [41], Pastry [35] i Kademlia [28].

Ne struktuirani P2P sistem je sačinjen od parnjaka koji se pridružuju mreži sa

nekim labavim skupom pravila, bez ikakvog predznanja o njenoj topologiji. Freenet, Gnutella, FastTrack, KaZaA i BitTorrent su primeri ovakvih sistema. Ovakve mreže se zbog svojih svojstava tipično koriste za udruživanje snage parnjaka nebi li se neki posao brže obavio ili razmenu fajlova kod kojih nije od presudnog značaja brzina pronalaska fajla.

U nestruktuiranim P2P mrežama, pretraga je bazirana na prosleđivanju upita. Svaki parnjak upit prosleđuje komšijama, osim u slučaju kada parnjak može odgovoriti na traženi upit ili se brojač skokova upita smanji na 0 [38]. Sa ovakvim načinom prosleđivanja postoje različita pravila kontrole prosleđivanja upita. Način prosleđivanja može se kontrolisati tako što se upit neće proslediti svim komšijama nego samo određenim. Izbor kome će upit biti prosleđen se vrši prema informacijama koje parnjak čuva o komšijama. Te informacije predstavljene su istorijom ili indeksima sadržaja koji komšija može obezbediti. Takođe, često se prosleđivanje vrši slučajno bez ikakvog znanja. Naravno, ovo svojstvo smanjuje šansu da se pronađe sadržaj. Opravdanje nestruktuiranog pristupa zasniva se na promenljivosti broja članova mreže i ispravnosti prvog izbora komšije.

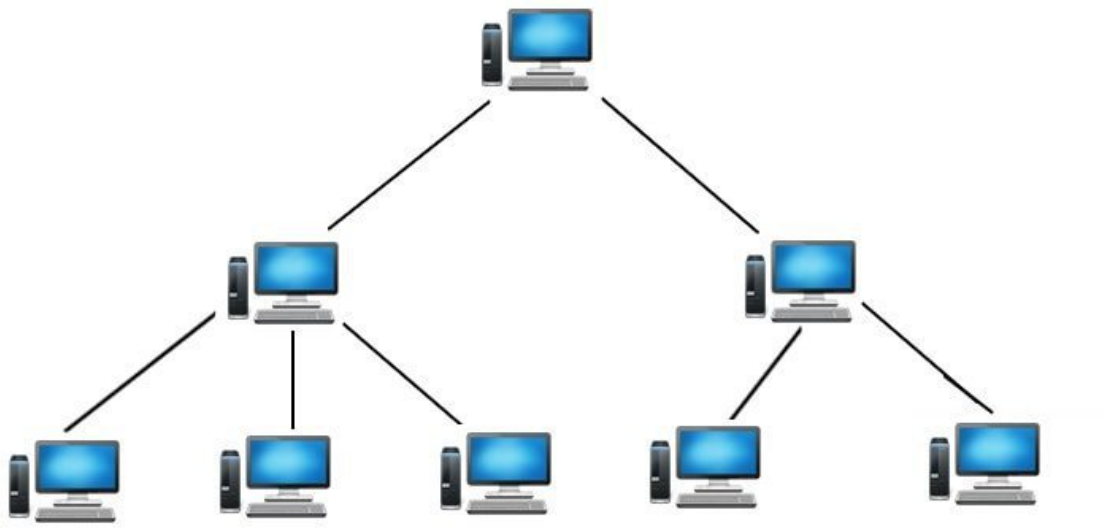
Nestruktuirane mreže ravnopravnih računara uglavnom su sastavljene od parnjaka koji su kućni PC računari. Oni su najčešće povezani sa mrežom slabim propusnim opsegom (eng. *bandwidth*). Takođe, većinom je propusni opseg nesrazmeran u odnosu na dolazni i odlazni saobraćaj u korist dolaznog saobraćaja. Navedena činjenica blago narušava samoskalabilnost mreže. Postoje različiti načini da se spreči narušavanje samoskalabilnosti. Neki od njih su zasnovani na kažnjavanju parnjaka koji ne doprinose ili jako malo doprinose resursima sistema.

2.3 Topologija

P2P sistem je sačinjen od dva podsistema koji se izvršavaju paralelno. To su konstrukcija i održavanje izabrane topologije kao i prosleđivanje bloka (eng. *chunk*) podataka. Algoritmi za konstrukciju i prosleđivanje zajedno formiraju P2P protokol. U zavisnosti od odluka donešenih pri dizajnu sistema tj. od izabrane topologije i algoritma prosleđivanja može se napraviti P2P sistem sa različitim karakteristikama.

Topologija drveta

Među prvim predloženim topologijama P2P sistema za razmenu sadržaja predstavljena je topologija drveta. U okviru ove topologije, svi parnjaci su organizovani tako da čine strukturu drveta. Primer topologije drveta prikazan je na slici 2.1.



Slika 2.1: Primer topologije drveta

U korenu drveta nalazi se parnjak koji deli sadržaj-izvor. Svaki parnjak u sistemu može imati tačno određeni broj dece koji je ili unapred definisan ili određen kapacitetom parnjaka. Deca su komšije koje preuzimaju sadržaj od roditelja. Konstrukcija drveta i relacija roditelj-dete može biti određena različitim faktorima kao što su ukupno vreme potrebno da podatak stigne od izvora do svih parnjaka u mreži, dostupan propusni opseg i fizička topologija same mreže. Podaci koji se šalju na ovakav način se najpre podele na manje delove koji se nazivaju paketi. Paket se zatim šalje od izvora do parnjaka koji se nalaze na prvom nivou drveta. Nakon toga parnjaci sa prvog nivoa drveta šalju podatke parnjacima na drugom nivou drveta i tako redom sve dok paket ne dođe do svih parnjaka u mreži [12].

Iako jednostavna za konstrukciju, topologija drveta ima dosta nedostataka. Najveći nedostatak je neravnomerna raspodela tereta prosleđivanja koja se javlja kao posledica činjenice da veliki deo parnjaka čine listovi koji ne daju nikakav doprinos sistemu. Dodatno, parnjak koji nema odlazni (eng. *upload*) kapacitet proporcionalan proizvodu broja dece i veličine bloka koji prenosi, konstantno će nagomilavati

blokove podataka i usporavati ceo sistem. Ovakav parnjak treba da bude lociran jedino u listovima stabla ili veoma blizu listova. Pored navedenih osobina, važno je napomenuti da je ovakva struktura ranjiva na napuštanje parnjaka. Prilikom napuštanja parnjaka na višem nivou, njegova podstabla privremeno ostaju odsečena od ostatka mreže.

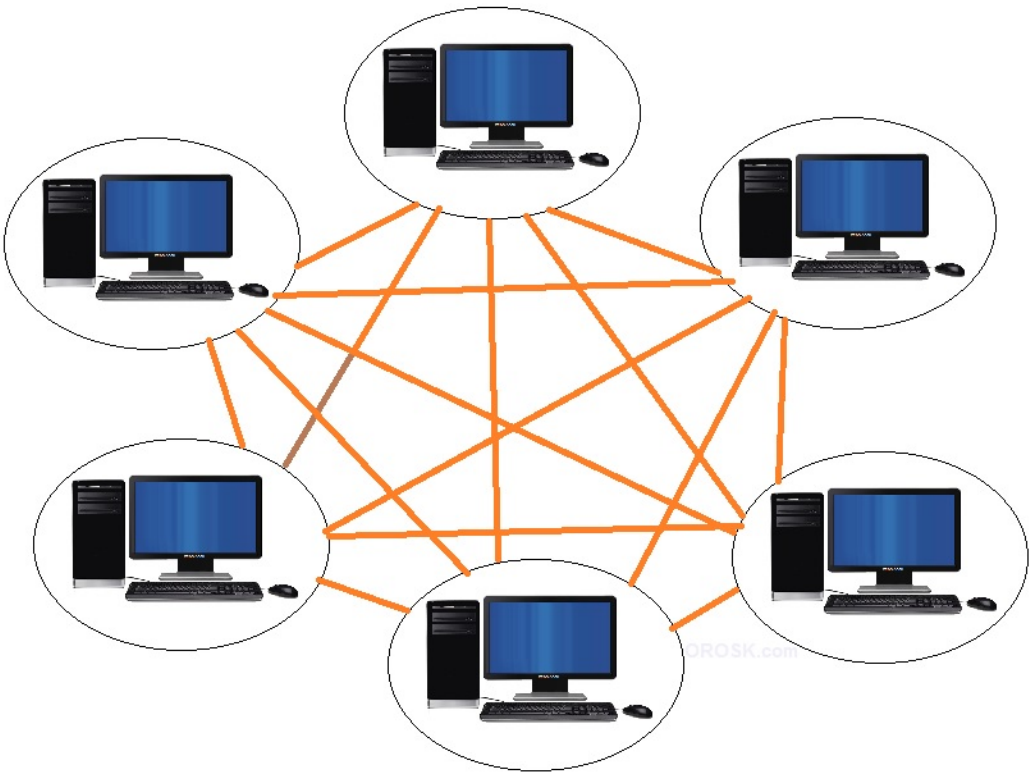
Da bi se povećala otpornost na napuštanja parnjaka i povećao kapacitet mreže koristi se rešenje bazirano na višestrukim stablima. U ovakvim sistemima parnjaci se nalaze u jednom ili više stabala. Svaki parnjak prima sadržaj od svih stabala čiji je član. Izvorni čvor je izvor za sva stabla. Dobijen sadržaj parnjak prosleđuje unapred određenim stablima. Sistem se na ovaj način deli u podsisteme u kojima svaki od podsistema predstavlja jedno stablo. Radi bržeg distribuiranja sadržaja svakom podsistemu se dodeljuje jedinstven paket.

Topologija zasnovana na višestrukim stablima rešava neke od problema upotrebe samo jednog stabla, ali uvodi dodatne probleme [12]. Prvo, na ovakav način postoji više putanja od izvora do lista pa je potreban značajno složeniji algoritam prosleđivanja. Drugo, kako stabla imaju različite podatke potrebno je obezbediti da svi podaci dođu do svakog parnjaka. Treće, prilikom konstrukcije mreže treba kreirati stabla koja su što je moguće manje dubine čime bi se obezbedio brži protok podataka. Dodatno, javlja se povremena potreba prebacivanja parnjaka iz jednog stabla u drugo uz dodatni zadatak balansiranja dubina svih stabala. Četvrti problem leži u balansiranju kapaciteta stabala.

Topologija mreže

Sistem zasnovan na mreži se fokusira na kreiranje topologije koja se brzo adaptira na promene. Glavna ideja pri pravljenju ove topologije bila je da parnjaci provode što je moguće manje vremena za održavanje pokrivača i brzi oporavak mreže pri njihovom napuštanju. Prednost ove topologije je brža i jednostavnija konstrukcija pokrivača. Sa druge strane, u ovakvoj konstrukciji potrebno je uložiti značajno vreme za implementaciju algoritma za prosleđivanje sadržaja između parnjaka. Primer topologije mreže nalazi se na slici 2.2.

Značajan kontrast se može primetiti ukoliko se posmatraju razlike između topologije mreže i topologije drveta 2.3. Kod topologije drveta kreiranje same topologije je složenije i zahteva više vremena, ali jednom kad se postigne željeni dizajn mreže prosleđivanje podataka je jednostavno. Kod topologije mreže situacija je potpuno



Slika 2.2: Primer topologije mreže

drugačija. Sama izrada pokrivača je krajnje jednostavna, jeftina i brza, ali je prosleđivanje podataka značajno kompleksnije.

U mrežnom pokrivaču glavni cilj parnjaka je da održavaju veliki broj dolaznih konekcija [38]. U slučaju izlaska komšije iz mreže, njegov uticaj na parnjake je minimalan. Pokrivač se obično gradi na distribuirani način i svaki parnjak je svestan samo malog dela učesnika, njegovih komšija. Način biranja komšija se razlikuje od politike (eng. *policy*) do politike, pri čemu je u svim slučajevima ovaj proces brz i jednostavan. Primer brze i jednostavne politike je slučajno biranje komšija.

U odnosu na topologiju drveta, prosleđivanje podataka u mreži ne odvija se pravolinijski. Nedostatak konkretne strukture koja bi definisala put kojim paket treba ići čini nemogućim određivanje unapred putanje kojom paket treba proći. Zbog navedenog svojstva prosleđivanje paketa je bazirano na lokalnim odlukama koje svaki parnjak donese. Parnjaci donose odluke prema informacijama koje imaju o komšijama. Paket se može razmeniti primenom guranja (eng. *push*) ili povlačenja (eng. *pull*). Kod guranja parnjak određuje koji paket šalje svakom komšiji, a kod povlačenja određuje koji paket traži od svakog komšije. Oba pristupa imaju dobre

i loše strane.

Pristup sa guranjem je efikasniji u mreži sa ograničenim odlaznim kapacitetom jer izbegava višestruke zahteve za pakete. Pristup sa povlačenjem je dobar izbor za mreže sa ograničenim dolaznim kapacitetom jer parnjak može da kontroliše koliko brzo će primati pakete od komšija. Kod obe šeme zbog lokalne koordinacije propagacije paketa dolazi do blage neefikasnosti u prosleđivanju. Kod sistema guranja ova činjenica se ogleda u višestrukim kopijama koje parnjak može da dobije ako mu više komšija šalje isti paket. Kod sistema povlačenja može se desiti da komšije zaguše parnjaka sa previše zahteva za paketima. Takođe sistemi povlačenja unose dodatan sadržaj u mrežu (eng. *overhead*) s obzirom da se paket mora tražiti.

Hibridna topologija

Hibridna topologija kombinuje prednosti topologija drveta i mreže. Preciznije, predstavlja kombinaciju robusnosti mrežne topologije sa jednostavnošću i efikasnošću prosleđivanja paketa koju nudi topologija drveta.

U hibridnoj topologiji, mreža je podeljena na više podmreža slično kao kod topologije drveta [38]. Da bi parnjak mogao da prima pakete, mora pronaći roditelje koji će mu prosleđivati pakete iz svih podmreža. Kada se novi parnjak pridružuje mreži, on dobija listu svih parnjaka na koje može da se poveže. Zatim parnjak među njima pokušava da odabere komšije tako da pokrije sve podmreže. Nakon što je pronašao takve parnjake povezuje se na njih. Glavna mana ovakvog sistema je ta što je dobijanje pravilne topologije težak zadatak. Često se može desiti da mala greška dovodi do situacije da parnjak ili grupa parnjaka bude izolovana od ostatka mreže.

2.4 Protok video sadržaja

Od samog nastanka, video je važan mediji za komunikaciju i zabavu. Sačinjen je od serije slika koje se smenjuju odgovarajućom brzinom dajući privid neprekidnog kretanja. Ovaj trik bio je poznat još u drugom veku u Kini, ali je ostao nepoznat ostatku sveta sve do 19. veka. Otkrićem kamere 1888. godine omogućeno je „automatsko" hvatanje i čuvanje pojedinačnih komponenti slike na filmskoj traci. Emitovanje televizijskog signala, nakon izuma 1928. godine, omogućilo je bilionima ljudi širom sveta uživo praćenje snimljenog sadržaja. Zahvaljujući širokoj dostup-

nosti signala, primarni vid zabave i dobijanja informacija umesto novina i radija postala je televizija.

Tokom većeg dela 20. veka, jedini način dobijanja televizijskog signala bio je preko vazduha ili kabla. Ranih 2000-tih internet doživljava veliki rast propusnog opsega. Pored rasta propusnog opsega svakodnevno se razvijaju bolji algoritmi kompresije video snimaka čime se omogućava da se sa značajno manje podataka predstavi isti snimak bez velikih gubitaka na kvalitetu. Takođe, značajnu ulogu u svemu igra i Murov zakon [37] činjenicom da eksponencijalni porast snage računara omogućava rešavanje raznovrsnijih problema. Navedena svojstva omogućavaju da sistem za dostavu video sadržaja u realnom vremenu (eng. *streaming*) putem interneta postane moguć. Dostava video sadržaja u realnom vremenu omogućava emitovanje pre preuzimanja celokupnog sadržaja. Video se neprekidno šalje i emitovanje se omogućava odmah po pristizanju ili sa određenom zadrškom [42].

Internet je dizajniran za slanje paketa bez mogućnosti kontrole toka. Ovakav način transfera ne pogoduje slanju vremenski neprekidno baziranog saobraćaja kao što su video i audio sadržaji [46]. Glavni razlog je što dostava u realnom vremenu ima određeni poredak kojim se mora emitovati kao i vremenska ograničenja. Na primer, video sadržaj se mora emitovati neprekidno slika za slikom. U slučaju da podatak ne stigne na vreme, proces emitovanja će biti prekinut.

Danas se internet sve više koristi za razmenu multimedijalnog sadržaja umesto statičnog teksta i grafika [5]. Aplikacije koje zahtevaju dostavu u realnom vremenu velikom broju korisnika su, između ostalog, Internet TV, prenos sportskih događaja, online igrice i edukacija preko daljine. Usled ovako širokog spektra primena istraživači već skoro trideset godina istražuju odgovarajuću podršku aplikacijama u obliku IP višesmernog emitovanja (eng. *multicast*). IP višesmerno emitovanje je vrsta slanja paketa u kome se dodaje posebna zastavica (eng. *flag*) u IP pakete kojom se signalizira ruterima da je paket potrebno poslati grupi računara paralelno [13]. Međutim, usled ozbiljnih problema sa skaliranjem i podrškom na višim nivoima funkcionalnosti, višesmerno emitovanje nije zaživelo šire. Visoki troškovi mreža za isporuku sadržaja (eng. *Content Delivery Networks (CDN)*) i obezbeđivanja odgovarajućeg propusnog opsega su dva glavna faktora koji ograničavaju ovakav vid slanja samo na mali deo izdavača Internet usluga, koji višesmerno emitovanje prevashodno koriste za dostavu kvalitetnog video sadržaja IPTV. Za dostavu sadržaja pretplatnicima izdavači koriste usmerivače paketa (eng. *packet switching*).

U nadolazećim godinama, postoji značajni interes za korišćenje mreže ravno-

pravnih računara za dostavu sadržaja u realnom vremenu. Dve ključne stavke čine pomenuti pristup primamljivim. Prvo, ovakva tehnologija ne zahteva podršku mrežne infrastrukture što je čini jeftinom i jednostavnom za izradu. Drugo, u takvoj tehnologiji, svi koji primaju sadržaj isti taj sadržaj dele dalje, što obezbeđuje visok nivo skalabilnosti. Pored velikog interesa i uloženog truda u pronalaženje odgovarajućeg P2P sistema, ova tema je još uvek otvorena. Glavna tačka spoticanja P2P mreže za dostavu sadržaja u realnom vremenu je susret sa drugačijim poteškoćama u odnosu na uobičajne probleme mreže. Kod dostave sadržaja u realnom vremenu posebno se mora voditi računa o kašnjenju signala, dok u slučaju preuzimanja sadržaja kašnjenje nije kritično. Zapravo, prihvatljivo je preuzimanje sadržaja u trajanju od nekoliko sati pa čak i dana. Različitost i striktniji zahtevi prenosa video sadržaja u realnom vremenu zahteva fundamentalno drugačiji dizajn i pristup.

2.5 Modelovanje mreže

U idealnom slučaju, P2P sistem za dostavu sinhronzovanog sadržaja bi trebao postići jednako dobre performanse uzimajući u obzir sve bitne metrike. Pri postizanju ovog cilja neophodno je prevazići različite izazove povezane sa okruženjem u kojem P2P sistem radi i korisničkim ponašanjem. Izazovi su mnogobrojni i različiti pri čemu u nastavku navodimo četiri glavna.

Prvo, P2P sistem treba biti što je moguće otporniji na talasanje. Talasanje može da utiče na kontinualnost signala koji se prikazuje korisniku uzimajući u obzir činjenicu da napuštanje parnjaka može usporiti ili potpuno onemogućiti distribuciju podataka u mreži. Kada parnjak napusti mrežu, svi parnjaci kojima je on bio dobavljač sadržaja moraju pronaći novog dobavljača u što kraćem roku.

Drugo, ovakav sistem treba imati dobro skaliranje. Kako mreža raste, povećava se broj parnjaka kao i ukupno vreme potrebno da se sadržaj pošalje svim parnjacima. Za nesmetanu reprodukciju ovakvog sadržaja, kašnjenje reprodukcije takođe treba da raste. Kada se u obzir uzme i skalabilnost, izazov je dvostruki. Dodatno, porast prosečnog kašnjenja reprodukcije sadržaja treba biti takav da nema veliki uticaj na ukupno iskustvo korisnika (eng. *user experience*). Drugo, sistem treba održavati što manje kašnjenje čime se obezbeđuje bolji privid prenosa uživo.

Treći izazov sa kojim se susreću P2P sistemi su različiti propusni opsezi parnjaka. Uspešnost samog sistema zavisi od prosečnog odlaznog kapaciteta parnjaka i jačine enkodovanja video snimka. Danas, propusni opsezi parnjaka mogu biti ve-

oma različiti. Dodatnu poteškoću predstavlja asimetrična priroda pristupa internetu koja dovodi do toga da parnjaci značajno više sadržaja mogu preuzeti nego poslati. Zbog ovih razloga, P2P sistem treba biti izgrađen tako da se heterogenost propusnog opsega koristi efektivno.

Četvrti izazov je da prosečna brzina slanja parnjaka može biti smanjena usled nevoljnosti parnjaka za doprinos mreži. Parnjaci koji žele da prime podatke ali ne i da ih dalje dele sa ostatkom mreže se nazivaju grebatori (eng. *free-riders*). Grebatori su opasnost za sistem jer koriste resurse bez daljeg deljenja pa samim tim smanjuju prosečnu brzinu deljenja u mreži. Stoga, važno je da se P2P sistem dizajnira tako da podstiče doprinos mreži.

Glava 3

Predloženi Kikkar P2P protokol

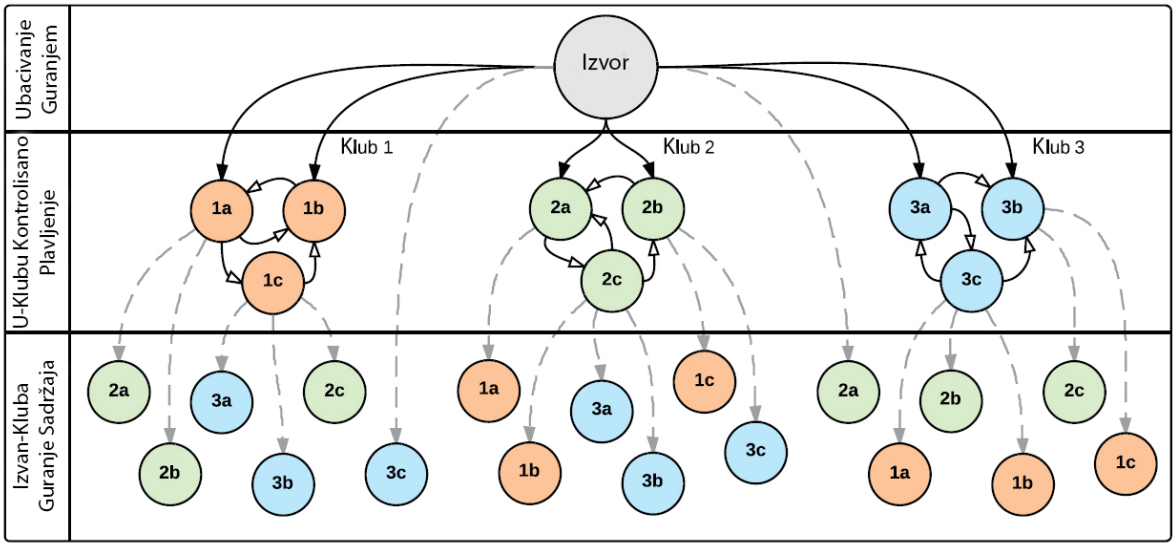
U poglavlju 2 navedeno je da distribucija medijskog sadržaja preko interneta privlači veliku pažnju u poslednjoj deceniji. Navedena činjenica dovela je do detaljnijeg istraživanja teme od strane naučnika i industrije. Veliki broj različitih P2P protokola za dostavu video sadržaja u realnom vremenu predstavljen je u prethodnim godinama [50]. Pored velikog broja predloženih protokola, samo pojedini imaju neki vid sinhronizacije. Ostali protokoli vode se logikom da će sadržaj biti relativno sinhronizovan. Relativna sinhronizacija uzrokuje da se sadržaj koji se prikazuje na parnjacima može značajno razlikovati.

Protokol Kikkar [1] je baziran na Screamer protokolu [11]. Screamer koristi hibridnu topologiju koja omogućava dostavu video sadržaja u realnom vremenu sa malim zastojem (eng. *delays*) i malim dodatnim saobraćajem u mreži (eng. *overhead*). Zbog ovakvih karakteristika posebno je pogodan za dostavu video sadržaja u realnom vremenu. Važno je napomenuti da sam Screamer protokol nema nikakav mehanizam kojim bi se kontrolisala sinhronizacija sadržaja među parnjacima, te Kikkar menja neke delove i donosi nove kako bi omogućio sinhronizaciju.

3.1 Arhitektura mreže

Protokol Kikkar je baziran na protokolu Screamer. Odavde zaključujemo da se Kikkar klasifikuje kao hibridni pokrivač koji koristi brojne mehanizme kako bi postigao što manji zastoj i dodatni saobraćaj u mreži. Na slici 3.1 prikazana je struktura mreže za dostavu video sadržaja u realnom vremenu za jednu konfiguraciju protokola. Uočavamo da su parnjaci podeljeni u klubove. Jedan parnjak može da

[1] Kikkar (hebrejski kružni okrug, kompletna težina)



Slika 3.1: Screamer topologija (slika je preuzeta iz rada [36])

pripada najviše jednom klubu. Ukupan broj klubova je 6. Proces dostave video sadržaja se može apstrahovati na tri faze: (1) ubacivanje guranjem video blokova od izvora (eng. *source*) u ostale klubove, (2) kontrolisano plavljenje (eng. *flooding*) u klubu, i (3) guranje sadržaja u listove mreže, tj. guranje sadržaja izvan kluba.

Kikkar protokol razlikuje tri glavne uloge u sistemu: tragač (eng. *tracker*), parnjak koji proizvodi i distribuira video snimak (eng. *source*) i parnjaci koji traže video snimak. Tragač služi da pomogne parnjacima u bržem i lakšem pronalaženju ostalih parnjaka. Bitno je napomenuti da se tragač koristi samo za otkrivanje novih parnjaka i sinhronizaciju časovnika u mreži.

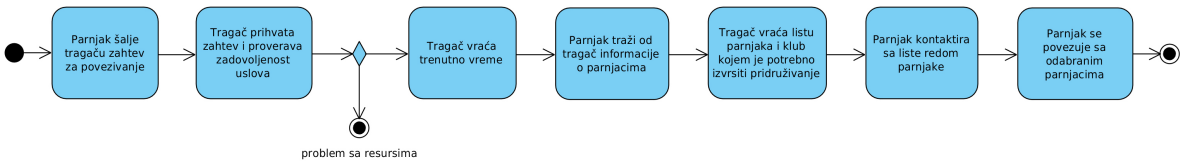
Kod Kikkar protokola, svi parnjaci su podeljeni u klubove, tj. svaki parnjak pripada jednom klubu. Dodeljivanje veze parnjak-klub vrši tragač prilikom pristupanja parnjaka sistemu. Parnjak doprinosi klubu kome pripada svojim resursima. Ostalim klubovima parnjak je list tj. ništa ne doprinosi već samo dobija sadržaj. Parnjak koji šalje video igra posebnu ulogu u sistemu, tj. pripada svim klubovima. Takođe, on je zadužen za podelu video snimaka na manje delove i dodelu delova određenim klubovima. Ovakav način slanja je čest kod topologije višestrukih stabala.

Parnjak komunicira sa tragačem upotrebom HTTP protokola. Parnjaci komuniciraju međusobom koristeći protokol korisničkih datagrama (eng. *User Datagram Protocol, UDP*). Veze između parnjaka se formiraju tako što se povezuju parnjak

koji šalje i parnjak koji prima. Važi pravilo da parnjak koji je na početku veze odabran da šalje video ne prima sadržaj od parnjaka primaoca. Drugim rečima veze su jednosmerne.

Svaki parnjak u klubu teži da ima bar dve i ne više od tri odlazne i dolazne konekcije sa parnjacima koji pripadaju njegovom klubu. Ovim svojstvom se postiže eksponencijalna brzina slanja video snimka. Takođe, pored parnjaka iz njegovog kluba teži i da postigne po jednu dolaznu konekciju sa parnjacima koji ne pripadaju njegovom klubu. Ova konekcija je istog značaja kao i konekcija u klubu. Bez njenog postojanja, ne mogu se obezbediti svi potrebni podaci. Broj odlaznih konekcija sa parnjacima koji ne pripadaju njegovom klubu nije bitan. Konekcije će detaljnije biti opisane u pogavlju 3.4.

3.2 Pridruživanje parnjaka



Slika 3.2: Pridruživanje novog parnjaka

Prilikom pridruživanja parnjaka jatu (eng. *swarm*) kako bi preuzeo sadržaj u mreži, neophodno je slanje zahteva za povezivanje tragaču. Navedeno svojstvo ilustrovano je dijagramom aktivnosti 3.2. Kako bi se jato zaštitilo od pridruživanja parnjaka koji ne može obezbediti odgovarajuće resurse, tragač proverava brzinu dolaznog i odlaznog saobraćaja novog parnjaka. Ukoliko je brzina dolaznog i odlaznog saobraćaja ispod granice, tragač prekida kontakt sa parnjakom. Ukoliko je brzina dolaznog i odlaznog saobraćaja zadovoljavajuća tragač vraća trenutno vreme parnjaku korišćenjem protokola mrežnog vremena (eng. *Network Time Protocol, NTP*). Nakon uspešne sinhronizacije vremena, parnjak od tragača zahteva informaciju o parnjacima koji pripadaju jatu. Tragač izdvaja iz svakog kluba određeni broj parnjaka i prosleđuje njihove informacije. Informacija sadrži IP adresu parnjaka, port na kome parnjak osluškuje i pripadnost parnjaka klubu. Dodatno, pored informacije o parnjacima tragač prosleđuje parnjaku klub u koji je obavezan da uđe. Kako bi se obezbedio jednak broj parnjaka po klubovima, novi parnjak se uvek upućuje na prvi sledeći klub na koji je poslednji parnjak prosleđen.

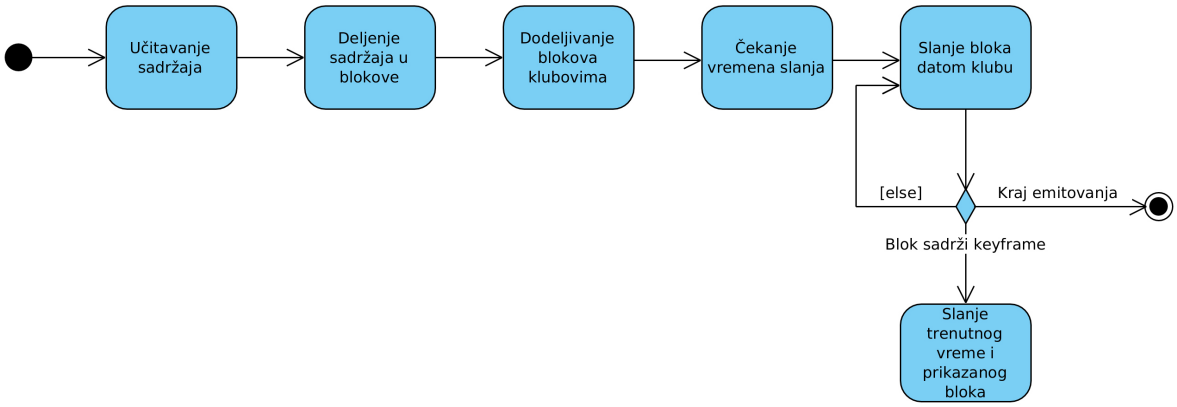
Parnjak zatim koristi dobijenu listu parnjaka kako bi odlučio sa kojima od njih će se povezati. Najpre se šalje `ping` poruka za povezivanje svim parnjacima sa liste. Ping poruka sadrži broj porta na kojem parnjak osluškuje, broj kluba u kome se parnjak nalazi i tip konekcije koju parnjak želi da ostvari tj. odlazna ili dolazna konekcija. Zatim se čekaju odgovori određeno vreme. Nakon pristizanja ping poruke, parnjak odgovara `pong` porukom koja obavezno sadrži informacije o broju veza koje ima ping-ovani parnjak i količini uskladištenih podataka. Nakon određenog vremena, parnjak pregleda primljene pong poruk i sortira parnjake prema poželjnosti. Uslov koji se koristi da bi se poredili parnjaci prema poželjnosti je količina uskladištenih podataka. U slučaju postojanja više parnjaka sa istom količinom uskladištenih podataka bira se parnjak sa što manjim brojem dolaznih ili odlaznih veza kako bi se što manje remetilo jato.

Zatim se odgovarajućim parnjacima šalje `request` poruka. Ova poruka sadrži informaciju o klubu u kome se parnjak trenutno nalazi kao i tip konekcije. Trenutni klub i tip konekcije su prosleđeni ranije u okviru ping poruke ali kako je moguće da parnjak u međuvremenu dobije neki drugi klub ili port ponovo se šalju. Na request poruku odgovara se `response` porukom. Response poruka predstavlja potvrdu o povezivanju. Ukoliko izostane response poruka povezivanje se i dalje smatram uspešnim pri čemu sistem za kontrolu zagušenja ima zadatak da neaktivne parnjake izbaci iz mreže.

Svi parnjaci koji su se javili pong porukom i nisu izabrani se čuvaju u specijalnoj strukturi podataka. Glavni razlog čuvanja je brže kasnije povezivanje i manje opterećivanje servera ukoliko se kasnije ukaže potreba za povezivanjem. Parnjaci koji nisu poslali pong poruku se brišu jer se smatraju neaktivnim ili zagušenim.

3.3 Distribucija sadržaja

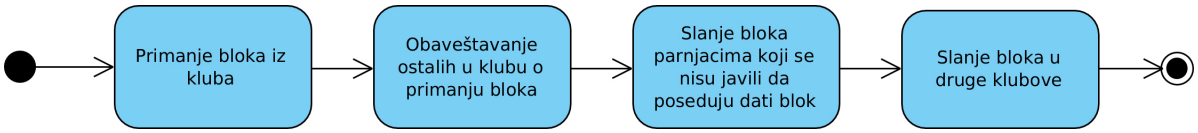
U delu 3.1 izvor se definiše kao parnjak koji generiše i šalje video sadržaj. Video sadržaj može biti prethodno snimljem pa samo emitovan ili uživo snimljen i emitovan. Na slici 3.4 predstavljen je dijagram aktivnosti izvora. Najpre je neophodno da izvor učita sadržaj. Zatim se učitani sadržaj deli na manje blokove koji su pogodniji za prenošenje putem interneta. Svaki od blokova sadrži svoj redni broj i prema tom broju se dodeljuje jednom od klubova. Dodeljivanje se vrši tako što se redni broj podeli po modulu sa brojem klubova. Dobijeni broj odgovara broju kluba kome dati blok pripada. Dodatno, blok sadrži redni broj dela video snimka koji se šalje,



Slika 3.3: Distribucija sadržaja od izvora ka korisnicima

informaciju da li je blok keyframe kao i sam video snimak.

Nakon dodeljivanja bloka klubu čeka se vreme slanja. Vreme čekanja odgovara ukupnom broju blokova koje je potrebno poslati podeljeno sa vremenom potrebnim za slanje blokova. Po isticanju vremena, blok se šalje u klub. Ukoliko poslati blok sadrži keyframe, pre njega se šalje `kontrolna` poruka. Kontrolna poruka sadrži broj video snimka koji je prikazan, dužinu do sada prikazanog video snimka izvora kao i vreme kada je kontrolna poruka napravljena. Osnovna svrha kontrolne poruke je da obavesti ostale parnjake u jatu o trenutno prikazanom sadržaju, tj. da omogući globalnu sinhronizaciju jata.



Slika 3.4: Distribucija sadržaja od korisnika ka korisnicima

Kod distribucije sadržaja među korisnicima razlikujemo dva načina distribucije. Podela je izvršena u zavisnosti da li dobijeni sadržaj pripada klubu ili ne. Kada parnjak primi blok podataka koji je dodeljen njegovom klubu, on obaveštava o isporuci sve parnjake u klubu sa kojima ima odlaznu vezu a zatim snima dati blok u skladište. Obaveštenje se vrši preko `have` poruke koja sadrži broj primljenog bloka. Obaveštavanje ostalih parnjaka ima za cilj da smanji mogućnost višestruke isporuke istog sadržaja pojedinim parnjacima. Posle određenog vremena parnjak šalje sadržaj svim parnjacima od kojih nije dobio nikakvu poruku koja bi ga obavestila da već ima taj sadržaj. U poruci koja služi za obaveštenje o postojanju sadržaja nalazi se

broj prisutnog bloka. Nakon što je sadržaj poslat u klub, sadržaj se šalje ostalim klubovima. Kako parnjak ima po jednu vezu sa parnjacima iz drugih klubova, nema potrebe za razmenom kontrolne poruke. Kada parnjak dobije blok koji ne pripada njegovom klubu, snima dati blok u skladište i ne distribuira blok dalje.

Prethodno opisan metod distribucije blokova u jatu funkcioniše bez greške u idealnim uslovima. Kako se u okviru mreže javljaju zagušenja i gube paketi, postoji mogućnost da deo snimka ne stigne do parnjaka. Da bismo obezbedili što bolji kvalitet snimka koristi se mehanizam traženja izgubljenih blokova. Pri primanju kontrolne poruke, parnjak proverava stanje prethodne grupe blokova. Za svaki nedostajući blok parnjak šalje zahtev za slanje svim komšijama u klubu. Zahtev sadrži listu brojeva nedostajućih blokova. Kako bi se izbeglo zatrpavanje komšija parnjak može tražiti najviše 20 nedostajućih blokova.

3.4 Kontrola zagušenja

U Kikkar protokolu, radi kontrolisanja zagušenja može doći do raskidanja konekcije pri određenim uslovima. Za raskidanje veze se koristi `terminirajuća` poruka. Ova poruka sadrži razlog terminacije (na primer dolazak novog parnjaka, prespora veza, izlazak iz jata). Jedna od okolnosti pri kojoj može doći do raskidanja konekcije je preveliko kašnjenje paketa. Ukoliko se preko dolazne konekcije ne dobije ni jedna poruka za unapred definisano vreme veza se smatra mrtvom i prekida se. Tačno vreme čekanja zavisi od same mreže i postavlja ga administrator mreže. Merenje vremena počinje dolaskom poslednje poruke. Pri svakoj dobijenoj poruci proveravaju se svi parnjaci. U slučaju da neki od parnjaka nije poslao poruku određeno vreme, veza se raskida. Ukoliko parnjak nema sadržaj za slanje a želi da spreči raskidanje veze, on šalje `keep-alive` poruku kojom obaveštava dobavljača sadržaja da je sa vezom sve u redu. Sa ciljem da bude što manja, keep-alive poruka sadrži samo redni broj koji je jedinstveno identifikuje.

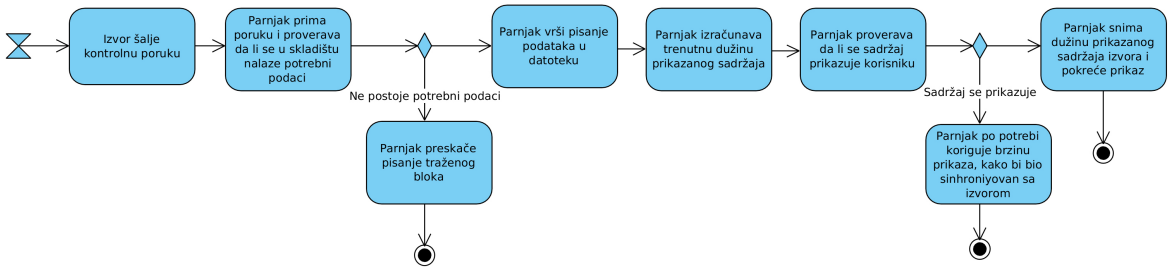
Druga okolnost pod kojom je dozvoljeno raskidanje veze je dolazak paketa u pogrešnom redosledu. Svaki paket ima svoj broj. Ukoliko paketi pristižu u pogrešnom redosledu to može ukazivati na postojanje zagušenja pa će paketi ići alternativnim putevima. U tom slučaju, radi rasterećenja mreže veza se raskida. Dodatno, mali broj paketa koji pristižu u pogrešnom redosledu za veći vremenski period ne mora ukazivati na kritično zagušenje. Treći način rešavanja problema sa zagušenjem je odbacivanje svih veza u kojima se blok podataka predugo čeka. Predugo čekanje

obično ukazuje na usko grlo u sistemu. Radi određivanja trajanja pauze u pristizanju paketa, čuvamo vreme kada je poslednji paket primljen.

Očigledno je da odbacivanjem neke veze možemo pokvariti topologiju i da je potrebno vršiti nova povezivanja. U zavisnosti od veze koja je odbačena, zavisi da li će doći do ponovnog povezivanja. Ukoliko je parnjak izgubio dolaznu vezu van kluba, onda od tragača traži spisak parnjaka i povezuje se sa parnjakom iz istog kluba kome je pripadao parnjak čija je veza izgubljena. Povezivanje se vrši postupkom koji je opisan u delu 3.2. Ukoliko je parnjak izgubio odlaznu vezu van kluba, ne vrši se nikakvo povezivanja. U slučaju da su parnjaci u istom klub, stvari su komplikovanije. U slučaju da parnjak izgubi dolaznu vezu u klubu, proverava da li su mu ostale najmanje još dve dolazne veze. Ukoliko nisu pronalazi se novi parnjak i vrši se povezivanje koje je opisano u delu 3.2. Isti princip važi i u slučaju gubljenja dolazne veze.

Pored problema sa manjkom veza, može se javiti i višak veza parnjaka. Kada parnjak dobije zahtev za povezivanje, povezivanje se ostvaruje ne uzimajući u obzir prethodne veze. Ukoliko parnjak ima više od tri dolazne ili tri odlazne veze u klubu potrebno je neke veze odbaciti. Odbacuju se one veze koje najduže nisu prenele paket.

3.5 Sinhronizacija parnjaka



Slika 3.5: Sinhronizacija sadržaja

U Kikkar protokolu izvor je pored slanja video sadržaja zadužen i za slanje kontrolnih poruka. Na slici 3.5 ilustrovan je način obrade kontrolne poruke. Kontrolna poruka se šalje u pravilnim razmacima pre svakog keyframe-a. Za razliku od ostalih poruka, kontrolna poruka se zbog svoje važnosti šalje u sve klubove. Nakon primanja kontrolne poruke, parnjak je prosleđuje svim komšijama sa kojima ima odlaznu

vezu, bez obzira da li su je oni već primili i da li su u njegovom klubu. Na ovaj način se osigurava da poruka stigne do svakog parnjaka. Nakon primanja kontrolne poruke koja ima isti sadržaj kao i neka od prethodnih poruka parnjak prekida njeno dalje prosleđivanje. Na ovaj način se sprečava beskonačno slanje poruke kroz jato. Nakon primanja kontrolne poruke, parnjak proverava da li skladište sadrži odgovarajuće video blokove koji su porukom obuhvaćeni. Ukoliko skladištu nedostaje prvi obuhvaćeni blok, preskače se ceo opseg. Kako se sve značajne informacije za prikazivanje video snimka nalaze u prvom bloku, prethodna činjenica je potpuno opravdana. U slučaju postojanja prvog bloka, obuhvaćeni blokovi se šalju video plejeru. Kako bi se sprečilo prekidanje video snimka, u slučaju preskakanja nekog video bloka prikazuje se prethodni video blok.

Ukoliko je uspešno izvršeno slanje sadržaja video plejeru, prelazi se na korak sinhronizacije. U koraku sinhronizacije se proverava da li video plejer prikazuje sadržaj. U slučaju neaktivnosti multimedijalnog plejera, snima se vrednost prikazanog sadržaja izvora sabrana sa kašnjenjem paketa. Ova vrednost će se koristiti kasnije kao reper za eventualne korekcije. Nakon što je vrednost snimljena pokreće se multimedijalni plejer. Ukoliko je plejer već aktivan, izračunava se dužina do sada prikazanog sadržaja. Ukoliko je ta dužina manja od zbira dužine emitovanja sadržaja izvora i kašnjenja paketa, potrebno je povećati brzinu snimka određeni vremenski interval dok se ne sinhronizuje sa izvorom. Slično, u slučaju da je pomenuta dužina veća od zbira dužine emitovanja sadržaja izvora i kašnjenja paketa, brzina se smanjuje određeni vremenski interval dok se ne postigne sinhronizacija sa izvorom.

Glava 4

Implementacija Kikkar protokola

Za demonstraciju Kikkar protokola opisanog u poglavlju 3 razvijene su veb i desktop aplikacija. Za razvoj je korišćen programski jezik Java 8 [23] i razvojno okruženje Eclipse [14]. Veb aplikacija implementira funkcionalnosti tragača Kikkar protokola, a desktop aplikacija u zavisnosti od podešenih parametara predstavlja parnjaka ili izvor.

Za uspešno pokretanje veb aplikacije neophodan je Java servlet kontejnera (eng. *Java Servlet Container*), dok je za desktop aplikaciju potrebna samo Java virtuelna mašina (eng. *Java virtual machine, JVM*). Prilikom inicijalizacije desktop aplikacije bira se mod pokretanja aplikacije. Dostupni su mod izvora ili mod parnjaka. Kada je mod izabran, aplikacija do kraja rada ostaje u odabranom modu. I veb i desktop aplikacija su otvorenog koda i mogu se pronaći na github repozitorijumu [1]. Detaljnije o veb aplikaciji može se pronaći u poglavlju 4.1. Više detalja o desktop aplikaciji nalazi se u poglavlju 4.2.

4.1 Implementacija tragača

Implementacija Kikkar protokola obuhvata aplikacije za veb i desktop. Veb aplikacija predstavlja tragača Kikkar protokola.

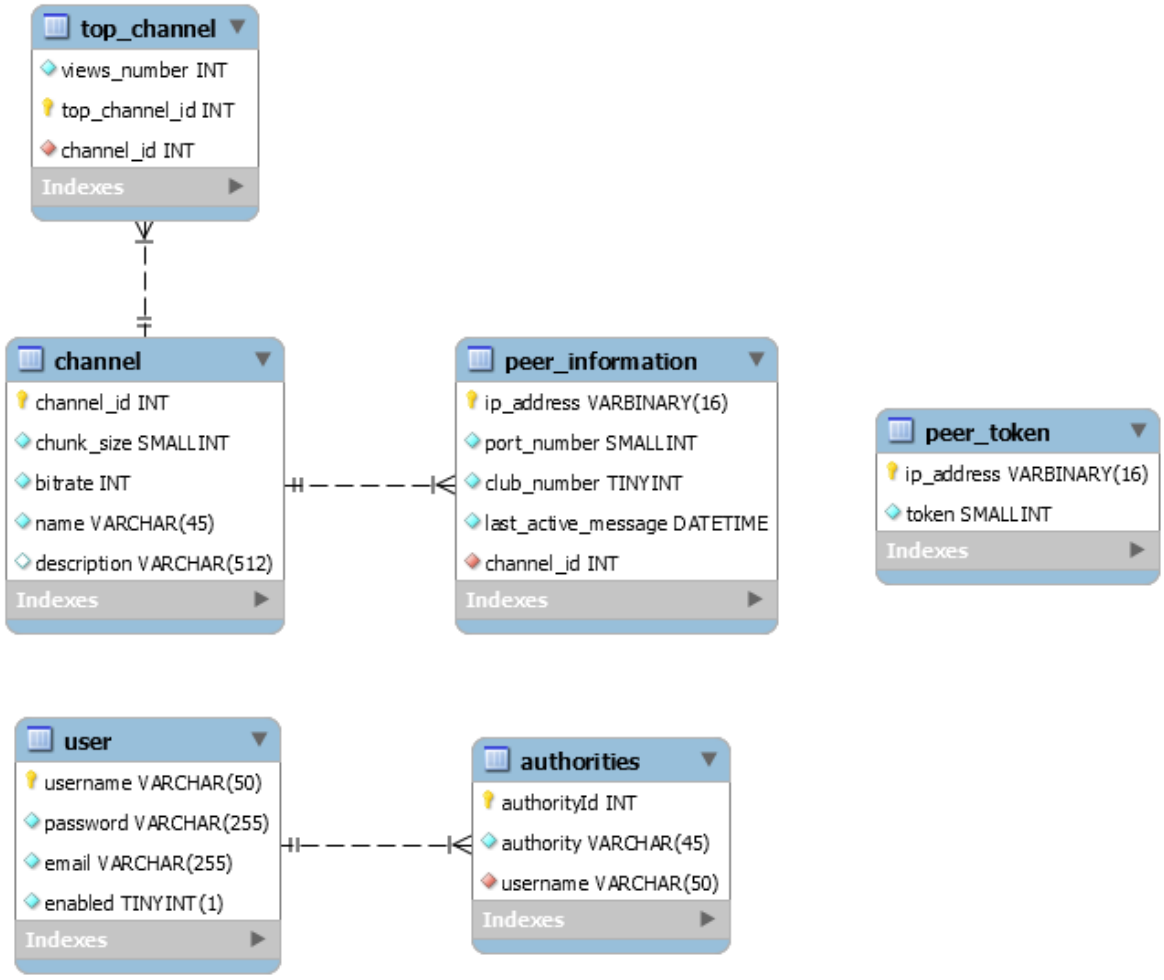
Pre implementacije samog tragača, neophodno je obavljanje nekoliko zadataka. Prvi zadatak je odabir odgovarajućeg Java servlet kontejnera za izvršavanje veb aplikacije. Za Java servlet kontejner odabran je Tomcat [4] verzije 9. Razlozi za odabir Tomcat kontejnera su široka primena u industriji, jednostavnost, odlična zajednica

[1] Repozitorijum se može naći na adresi `https://github.com/ozzy80/master-rad/tree/master/kod`

i dokumentacija. Drugi zadatak je odabir odgovarajuće baze podataka. Za bazu podataka izabrana je MySQL [30] baza. Detaljnije o samoj bazi i formiranim tabelama nalazi se u poglavlju 4.1. Konačno, potrebno je naći odgovarajće radne okvire (eng. *framework*). Detaljnije o korišćenim bibliotekama i radnim okvirima može se pronaći u poglavlju 4.1. U poglavlju 4.1 dat je opis funkcionalnosti aplikacije.

Baza podataka

Za potrebe aplikacije izabrana je MySQL [30] baza podataka. Razlog za odabir MySQL baze podataka je široka primena u industriji i nauci, dobra dokumentacija i jednostavna integracija. Na slici 4.1 predstavljen je EER dijagram baze podataka (eng. *EER database diagram*).



Slika 4.1: EER dijagram baze podataka

Tabela `channel` sadrži informacije o kanalima. Ove informacije pruža emiter signala pri registraciji kanala. Informacije koje se čuvaju u tabeli obuhvataju:

channel_id	- jedinstveni identifikacioni broj kanala, primarni ključ
chunk_size	- maksimalnu veličinu bloka podatka poslatog kroz mrežu
bitrate	- maksimalnu brzinu protoka video snimka
name	- ime kanala
description	- opis kanala

Tabela `peer_information` čuva informacije o aktivnim parnjacima za svaki kanal posebno. Za svakog parnjaka čuva se:

ip_address	- IP adresa, primarni ključ
port_number	- broj porta na kojem se osluškuje saobraćaj
club_number	- broj kluba kojem pripada
last_active_message	- vreme poslednje primljene aktivne poruke
channel_id	- jedinstven identifikacioni broj gledanog kanala, strani ključ ka tabeli `channel`

Tabela `user` čuva informacije o vlasnicima kanala, izvornim parnjacima. Tabela sadrži navedena polja:

username	- jedinstveno korisničko ime, primarni ključ
password	- šifru korisnika
email	- email adresa korisnika
enabled	- podatak o odobrenosti naloga

Tabela `authorities` služi za autorizaciju korisnika. Sastoji se od polja:

authority_id	- jedinstveni broj autorizacije, primarni ključ
authority	- ime autorizacije
username	- jedinstveno korisničko ime, strani ključ ka tabeli `user`

Tabela `top_channel` čuva listu najposećenijih kanala. Tabela se sastoji od polja:

views_number	- broj aktivnih parnjaka
top_channel_id	- jedinstveni identifikacioni broj, primarni ključ
channel_id	- jedinstven identifikacioni broj kanala, strani ključ ka tabeli `channel`

Tabela `peer_token` služi za čuvanje dodeljenog pristupnog tokena korisnicima. Tabela se sastoji od polja:

ip_address	- IP adresa parnjaka, primarni i strani ključ ka tabeli `user`
tokena	- pristupni token

Korišćene biblioteke i radni okviri

Pre opisa korišćenih biblioteka i radnih okvira, ukratko će biti predstavljen softverski alat *Apache Maven* [3]. Glavna namena alata je upravljanje i izgradnja projekta. Dva osnovna pravca upotrebe su priprema izvornog koda za distribuciju i jednostavno dodavanje Java biblioteka (eng. *jars*) u projekat. Kako su Java biblioteke obimne, većinom se nove biblioteke grade korišćenjem već postojećih. Postojeće biblioteke često koriste druge biblioteke pri čemu pomenuta zavisnost može biti proizvoljne dubine. U slučaju nekompatibilnosti različitih verzija biblioteka zadatak se dodatno komplikuje. Pri upotrebi Apache Maven alata pravila o zavisnostima navodimo u datoteci *pom.xml*. Ova datoteka mora da sadrži grupu, predmet za upotrebu, verziju i naziv projekta. U okviru pom.xml datoteke definiše se i način pakovanja, koji može biti jar (eng. *Java Application Archive*) ili war (eng. *Web Application Archive*). Kako je tragač veb aplikacija koristi se war. U okviru pom.xml nalazi se i deo sa zavisnostima (eng. *dependencies*). U okviru ovog dela se definišu biblioteke ili radni okviri koji su neophodni projektu. Bez upotrebe Maven-a navedena pravila morala bi se ručno postavljati.

U nastavku će biti opisane korišćene biblioteke i radni okviri. Prvi korišćeni je *Spring* radni okvir [40]. Spring je nastao 2002. godine sa idejom omogućavanja lake manipulacije velikih projekata na Java platformi (eng. *Java SE*). Pre Spring-a koristila se najčešće čista Java (eng. *pure Java*). Glavna mana ovakvog pristupa je težina učenja i održavanja koda. U početku Spring je predstavljao dopunu Java standarda za poslovne aplikacije (eng. *Java 2 Platform Enterprise Edition, J2EE*), ali danas se sve više koristi kao alternativa preobimnom i kompleksnom J2EE standardu, naročito u delu koji se tiče rada sa Java zrnima (eng. *Enterprise Java Beans*). Spring se može koristiti u bilo kojoj Java aplikaciji. Kao projekat, Spring okvir sastoji se od više modula koji su u velikoj meri nezavisni, tako da svaka aplikacija može da koristi samo deo Spring okvira koji joj je potreban. U projektu je korišćeno jezgro Springa, koje je obavezno za sve Spring okvire, modul podrške za veb MVC (eng. *Model–view–controller, MVC*) radni okvir, objektno relaciono mapiranje (eng. *object relational mapping*) i sigurnost same veb aplikacije.

Prilikom razvoja korisničke aplikacije jedna od ključnih stvari je razdvajanje sadržaja od prezentacije. Kako bismo ovo postigli na veb-u koristimo MVC arhitekturu. Glavni akcenat ovakve arhitekture je podela odgovornosti između različitih slojeva. Aplikacija je podeljena na tri glavne komponente i svaka od njih obavlja različite zadatke. Model je prva komponenta. On predstavlja internu reprezentaciju podataka. Sadrži glavne programske podatke kao što su informacije o objektima iz baze. Druga komponenta je pogled. Ova komponenta omogućava korisniku interfejs preko kog može da komunicira sa aplikacijom i prikazuje korisniku stanje modela. Treća komponenta je kontrolor. Glavna odgovornost kontrolora je upravljanje korisničkim zahtevima i iniciranje aktivnosti na nivou modela i promene na pogledu.

Komunikacija sa bazom podataka urađena je uz pomoć Hibernate [20]. Hibernate je programski okvir za objektno-relaciono mapiranje u Java programskom jeziku. Njegova glavna svrha je mapiranje objektno-orijentisanog modela u model relacione baze podataka radi lakše manipulacije podacima i održavanja koda. Mapiranje se postiže povezivanjem Java klasa i tabela baze podataka i mapiranjem Java tipova podataka u SQL tipove podataka. Povezivanje Java klasa i tabela baze podataka vrši se korišćenjem Java anotacija. Pretraga upita vrši se uz pomoć objektno orijentisane verzije SQL-a koja se zove HQL (eng. *Hibernate Query Language, HQL*). HQL nam omogućava pisanje upita nad Java objektima, a to znači da su upiti nezavisni od tipa baze podataka. Ukoliko dođe do zamene tipa baze podataka u okviru projekta, neće biti potrebno menjati upite. Za sam pristup MySQL bazi koristi se MySQL konektor.

Za kreiranje i čitanje iz JSON formata koristi se *jackson* [22]. Jackson je veoma popularna i izuzetno efikasna Java biblioteka za serijalizaciju i mapiranje Java objekata u JSON i obrnuto. Ostale biblioteke koje se koriste su *jstl* [25] kao tehnologija za kreiranje veb stranica sa dinamičkim sadržajem, biblioteka *bootstrap* [7] za kreiranje responzivnih veb sadržaja i *JQuery* [24] kao biblioteka za brži i lakši razvoj interaktivne veb stranice.

Neki detalji implementacije

Pri implementaciji tragača koristi se MVC arhitektura. Model čine klase `Channel`, `Authorities`, `PeerInformation`, `Token`, `TopChannel` i `User` koje redom odgovaraju istoimenim tabelama iz poglavlja 4.1. Sloj za pristup objektima (eng. *data access object, DAO*) služi za pisanje jednostavnih HQL naredbi za pristup mapiranim objektima. Klase koje se nalaze u ovom sloju imaju identična imena kao modeli

kojim pristupaju sa nastavkom DAO. Sloj servisa služi za biznis logiku. Sastoji se od klasa `ChannelManager`, `PeerConnectionManager`, `PeerInformationManager`, `TokenManager` i `UserManager`. Većina metoda ovih klasa vrši prosleđivanje upita nižim slojevima, jer ne postoji nikakva logika koju bi bilo neophodno implementirati. Jedini izuzetak je metod *void deleteDeadPeers()* koji je privatan u klasi `PeerInformationManager` i koristi se za periodično brisanje parnjaka koji se nisu javili u poslednjih 1000 sekundi. Kontroler se sastoji od klasa `FileUploadController`, `HomeController`, `LoginController` i `PeerConnectionController`. Za preuzimanje fajlova od korisnika koristi se klasa `FileUploadController`. `LoginController` se koristi za prijavljivanje i kreiranje novih naloga emiterima video sadržaja. Za opsluživanje veb stranica koristi se `HomeController`. Za pristupanje parnjacima koristi se `PeerConnectionController`.

Serverska aplikacija je napravljena po REST principima i prihvata zahteve ka sledećim adresama:

Metoda	Putanja	Opis
`POST`	`/channel/fileUpload`	Prihvata sliku od klijenta i čuva je u bazi podataka
`GET`	`/`	Preusmerava korisnika na stranicu za prijavljivanje
`GET`	`/channel`	Provera da li je korisnik prijavljen. Ukoliko jeste šalje ga na stranicu sa kanalima, ukoliko nije vraća ga na formu za prijavljivanje
`GET`	`/channel/channelId`	Vraća podatke o kanalu čiji je jedinstveni identifikacioni broj `channelId`
`GET`	`/login`	Preusmerava korisnika na stranu za prijavljivanje
`POST`	`/register`	Prihvata podatke od korisnika i čuva ih u bazi podataka, preusmerva korisnika na stranicu sa spiskom kanala
`GET`	`/initial/channelId`	Proverava se korisnikova verzija protokola. Ukoliko nije podržana vraća se statusni kod 505, ukoliko jeste proverava se brzina odlaznog i dolaznog saobraćaja. Ukoliko je brzina zadovoljavajuća prosleđuje se pristupni token, ukoliko nije vraća se statusni kod 509
`GET`	`/list/channelId`	Proverava se dobijeni token. Ukoliko token nije validan vraća se statusni kod 412, u suprotnom vraća se lista parnjaka
`GET`	`/leave`	Parnjak se briše iz liste parnjaka
`GET`	`/stayAlive`	Postavlja se vreme poslednjeg javljanja parnjaka

4.2 Implementacija parnjaka

U poglavlju 4.1 opisan je tragač kao jedan deo Kikkar protokola. U ovom poglavlju opisuje se desktop aplikacija. Desktop aplikacija odgovara parnjaku Kikkar protokola. U zavisnosti od podešenog moda parnjak može biti izvor ili obični parnjak. Za implementaciju parnjaka korišćeni su razni radni okviri i biblioteke. Detaljniji opisi radnih okvira i biblioteka nalaze se u poglavlju 4.2. Kako bi jato moglo da

funkcioniše razmenjuju se poruke između parnjaka. Detaljnije o tipu i nameni svake poruke može se naći u poglavlju 4.2. U poglavlju 4.2 dat je opis funkcionalnosti koje aplikacija pruža.

Korišćene biblioteke i radni okviri

Za upravljanje i izgradnju projekta koristi se *Apache Maven* [3] koji je opisan u poglavlju 4.1.

Biblioteka *Apache Commons Net* [2] se koristi za NTP protokol. Ona je skup različitih implementiranih internet protokola na klijentskoj strani. Glavna svrha biblioteke je da omogući korišćenje različitih protokola. Bitno je napomenuti da ona ne obezbeđuje viši nivo apstrakcije protokola već samo implementaciju.

Za pisanje jediničnih testova u projektu koriste se radni okviri *JUnit* [26] i *Mockito* [29]. JUnit dinamički proverava ispravnost softvera izvođenjem konačnog broja testova i upoređivanjem dobijenih rezultata sa očekivanim ponašanjem softvera. Koristi se za testiranje jedinica koda i integraciono testiranje. Testiranje jedinica koda je vrsta testiranja sa kojom se svaka jedinica koda testira kao zasebna celina. Jedinica koda može biti operacija, funkcija, metod, struktura podataka, klasa u zavisnosti od konteksta u kome se posmatra i programske paradigme. Integraciono testiranje je testiranje tokom kojeg se proverava saradnja između modula koji predstavljaju jednu celinu sistema. U okviru projekta je korišćeno samo testiranje jedinica koda. Mockito [29] je radni okvir koji omogućava kreiranje duplih test objekata (eng. *test double objects (mock objects)*) u automatskom jediničnom testiranju. Glavna korist od duplih test objekata je obmotavanje objekta tako da pri svakom pozivu vrši unapred definisanu akciju.

Gson [18] je Google-ova biblioteka koja se koristi za konverziju Java objekata u njihovu JSON reprezentaciju i obrnuto. Glavna prednost Gson-a nad sličnim bibliotekama je ta što radi sa proizvoljnim objektima. Navedena činjenica omogućava konvertovanje objekata u JSON reprezentaciju čak i ako nemamo njihov izvorni kod (eng. *source code*). Takođe, Gson potpuno podržava Java dženerike (eng. *Java Generics*), što često nije slučaj sa sličnim bibliotekama.

Za potrebe pozivanja metoda prema unapred određenom rasporedu koristi se biblioteka otvorenog koda *Quartz* [34]. Quartz je veoma moćna biblioteka koja se može koristiti kako za jednostavne tako i za komplikovane zadatke. Glavna prednost biblioteke Quartz nad drugim bibliotekama iste namene je što sa lakoćom može obraditi i održavati i desetine hiljada poslova.

Za formatiranje poruka koje se razmenjuju među parnjacima koristi se Google-ova biblioteka *Protocol buffers* [33]. Protocol buffers je biblioteka koja se koristi za binarnu serijalizaciju podataka. Biblioteka radi tako što se definiše struktura podataka koje želimo da serijalizujemo. Kada je struktura definisana vrši se prevođenje definisane strukture u odgovarajući programski jezik. Definisana struktura podataka omogućava da se razmenjuju podaci između različitih programskih jezika. Pošto se prenose binarni podaci dobijeni izlaz je značajno brži i manji od izlaza dobijenog primenom tekstualnih formata kao što su XML ili JSON format. Detaljnije o porukama koje se koriste u protokolu može se pronaći u poglavlju 4.2

Za video plejer izabran je *VLC* plejer [44] zbog dobre podrške, dokumentacije i velikog izbora kodeka. Da bi VLC plejer mogao koristiti iz Java aplikacije upotrebljava se programski okvir VLCj [45]. VLCj omogućava Java aplikaciji da pokrene instancu nativnog VLC medijskog plejera u okviru Java AWT prozora ili Swing JFrame-a.

Protocol buffers poruke

Kao što je navedeno u poglavlju 4.2 za razmenu poruka se koristi biblioteka Protocol buffers. Protocol buffers je izabran u mnoštvu sličnih biblioteka jer je izlaz dobijen primenom ove biblioteke mali, sama biblioteka je dobro dokumentovana i jednostavna za korišćenje i ima odličnu zajednicu. U nastavku sledi opis svake poruke koja se razmenjuje u okviru Kikkar protokola.

Listing 4.1: Enumeracija tipova konekcije

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
6  enum ConnectionType {
7      UPLOAD = 0;
8      DOWNLOAD = 1;
9      BOTH = 2;
10 }
```

Enumeracija tipova konekcije predstavljena kodom 4.1 sadrži informaciju o tipu konekcije koju parnjak želi da ostvari. Postoje tri tipa konekcije: dolazna (*DOWNLOAD*), odlazna (*UPLOAD*) i oba (*BOTH*). Ukoliko parnjak želi odlaznu konekciju

znači da traži parnjaka kome će prosleđivati sadržaj. Ukoliko parnjak traži dolaznu konekciju znači da traži parnjaka od koga će dobijati sadržaj. U slučaju da parnjak zahteva oba tipa konekcije znači da će kasnije odlučiti koji će tip konekcije koristiti.

Listing 4.2: Ping poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

import "ConnectionType.proto";

message PingMessage{
    int32 pingId = 1;
    int32 portNumber = 2;
    int32 clubNumber = 3;
    ConnectionType connectionType = 4;
}
```

Ping poruka predstavljena kodom 4.2 definiše ping poruku opisanu u poglavlju 3.2. Poruka se sastoji od:

pingId	- jedinstveni identifikacioni broj poruke
portNumber	- broj porta na kome se osluškuje saobraćaj
clubNumber	- broj pripadajućeg kluba
connectionType	- tip tražene konekcije

Listing 4.3: Pong poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message PongMessage{
    int32 responsePingId = 1;
    int32 uploadLinkNum = 2;
    int32 downloadLinkNum = 3;
    int32 bufferVideoNum = 4;
}
```

Pong poruka predstavljena kodom 4.3 definiše pong poruku opisanu u poglavlju 3.2. Poruka se sastoji od:

responsePingId	- jedinstveni identifikacioni broj ping poruke na koju se odgovara
uploadLinkNum	- broj odlaznih veza
downloadLinkNum	- broj dolaznih veza
bufferVideoNum	- broj uskladištenog video sadržaja

Listing 4.4: Request poruka

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
6  import "ConnectionType.proto";
7
8  message RequestMessage{
9      int32 requestId = 1;
10     int32 clubNumber = 2;
11     ConnectionType connectionType = 3;
12 }
```

Request poruka predstavljena kodom 4.4 definiše request poruku opisanu u poglavlju 3.2. Poruka se sastoji od:

requestId	- jedinstveni identifikacioni broj poruke
clubNumber	- broj pripadajućeg kluba
connectionType	- tip tražene konekcije

Listing 4.5: Response poruka

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
6  message ResponseMessage{
7      int32 responseRequestId = 1;
8  }
```

Response poruka 4.5 predstavlja response poruku opisanu u poglavlju 3.2. Poruka se sastoji od:

responseRequestId	- jedinstveni identifikacioni broj request poruke na koju se odgovara

Listing 4.6: VideoPacket poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message VideoPacket{
    int32 videoNum = 1;
    int32 chunkNum = 2;
    bool firstFrame = 3;
    bytes video = 4;
}
```

Video paket poruka predstavljena kodom 4.6 definiše video snimak koji se šalje kroz jato. Poruka se sastoji od:

videoNum	- jedinstveni identifikacioni broj video snimka
chunkNum	- broj dela snimka
firstFrame	- identifikator prvog frejma snimka
video	- video sadržaj

Listing 4.7: Kontrolna poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message ControlMessage{
    int32 messageId = 1;
    int32 currentChunkVideoNum = 2;
    int32 playerElapsedTime = 3;
    int64 timeInMilliseconds = 4;
}
```

Kontrolna poruka predstavljena kodom 4.7 je kontrolna poruka opisana u poglavlju 3.5. Poruka se sastoji od:

messageId	- jedinstveni identifikacioni broj poruke
currentChunkVideoNum	- trenutno prikazani video blok
playerElapsedTime	- prošlo vreme od početka emitovanja snimka
timeInMilliseconds	- vreme pravljenja poruke

Listing 4.8: Have poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message HaveMessage{
    int32 videoNum = 1;
}
```

Poruka *HaveMessage* predstavljena kodom 4.8 je have poruka opisana u poglavlju 3.3. Poruka se sastoji od:

videoNum	- broj primljenog video bloka

Listing 4.9: NotInterested poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message NotInterestedMessage{
    int32 videoNum = 1;
}
```

Poruka *NotInterestedMessage* predstavljena kodom 4.9 je notInterested poruka opisana u poglavlju 3.3. Poruka se sastoji od:

videoNum	- broj video bloka koji je primljen preko poruke HaveMessage ali je već uskladišten

Listing 4.10: KeepAlive poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message KeepAliveMessage{
    int32 messageId = 1;
}
```

Poruka *KeepAliveMessage* predstavljena kodom 4.10 je keep-alive poruka opisana u poglavlju 3.4. Poruka se sastoji od:

messageId - jedinstveni identifikacioni broj poruke

Listing 4.11: RequestVideo poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

message RequestVideoMessage{
    int32 messageId = 1;
    repeated int32 videoNum = 2 [packed=true];
}
```

Poruka *RequestVideoMessage* predstavljena kodom 4.11 je request video poruka opisana u poglavlju 3.3. Poruka se sastoji od:

messageId - jedinstveni identifikacioni broj poruke
videoNum - lista nedostajućih brojeva video snimaka

Listing 4.12: ResponseVideo poruka

```
syntax = "proto3";

package com.kikkar.packet;
option java_multiple_files = true;

```

```
6  message ResponseVideoMessage{
7      int32 videoNum = 1;
8      int32 chunkNum = 2;
9      bytes video = 3;
10 }
```

Poruka *ResponseVideoMessage* predstavljena kodom 4.12 je response video poruka opisana u poglavlju 3.3. Poruka je identična video poruci po sadržaju. Razlog za njeno uvođenje je taj što se logički razlikuje od video poruke.

Listing 4.13: TerminatedReason poruka

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
6  enum TerminatedReason {
7      LEAVE_PROGRAM = 0;
8      BLOCK_TIMEOUT = 1;
9      PACKET_NUMBER_DISORDER = 2;
10     DEAD_PEER = 3;
11     NEW_CONNECTION = 4;
12   }
```

Poruka *TerminatedReason* predstavljena kodom 4.13 sadrži informaciju o razlogu raskidanja konekcije. Razlog može biti jedan od enum tipova:

LEAVE_PROGRAM	- zatvaranje aplikacije
BLOCK_TIMEOUT	- predugo vreme čekanja bloka
PACKET_NUMBER_DISORDER	- dolazak paketa u krivom redosledu
DEAD_PEER	- predugo nejavljanje parnjaka
NEW_CONNECTION	- raskidanje usled nove konekcije

Listing 4.14: Terminated poruka

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
```

```
6  import "TerminatedReason.proto";
7
8  message TerminatedMessage{
9      int32 terminatedId = 1;
10     TerminatedReason terminatedReason = 2;
11 }
```

Poruka *TerminatedMessage* predstavljena kodom 4.14 je terminated poruka opisana u poglavlju 3.4. Poruka se sastoji od:

terminatedId	- jedinstveni identifikacioni broj poruke
terminatedReason	- razlog raskidanja veze

Listing 4.15: PacketWrapper poruka

```
1  syntax = "proto3";
2
3  package com.kikkar.packet;
4  option java_multiple_files = true;
5
6  import "ControlMessage.proto";
7  import "HaveMessage.proto";
8  import "KeepAliveMessage.proto";
9  import "NotInterestedMessage.proto";
10 import "PingMessage.proto";
11 import "PongMessage.proto";
12 import "RequestMessage.proto";
13 import "RequestVideoMessage.proto";
14 import "ResponseMessage.proto";
15 import "ResponseVideoMessage.proto";
16 import "TerminatedMessage.proto";
17 import "VideoPacket.proto";
18
19 message PacketWrapper{
20     int32 packetId = 1;
21     oneof message {
22         ControlMessage controlMessage = 2;
23         HaveMessage haveMessage = 3;
24         KeepAliveMessage keepAliveMessage = 4;
25         NotInterestedMessage notInterestedMessage = 5;
26         PingMessage pingMessage = 6;
27         PongMessage pongMessage = 7;
```

```
28          RequestMessage requestMessage = 8;
29          RequestVideoMessage requestVideoMessage = 9;
30          ResponseMessage responseMessage = 10;
31          ResponseVideoMessage responseVideoMessage = 11;
32          TerminatedMessage terminatedMessage = 12;
33          VideoPacket videoPacket = 13;
34      }
35  }
```

PacketWrapper predstavljen kodom 4.15 je omotač koji obmotava sve poruke. Njegova glavna svrha je da omogući proveru dolaska poruka u pravilnom redosledu kao i da eliminiše ponovljene ili prijavi izgubljene poruke. Omotač se sastoji od:

packetId	- jedinstveni identifikacioni broj omotača
oneof message	- poruka koja je umotana

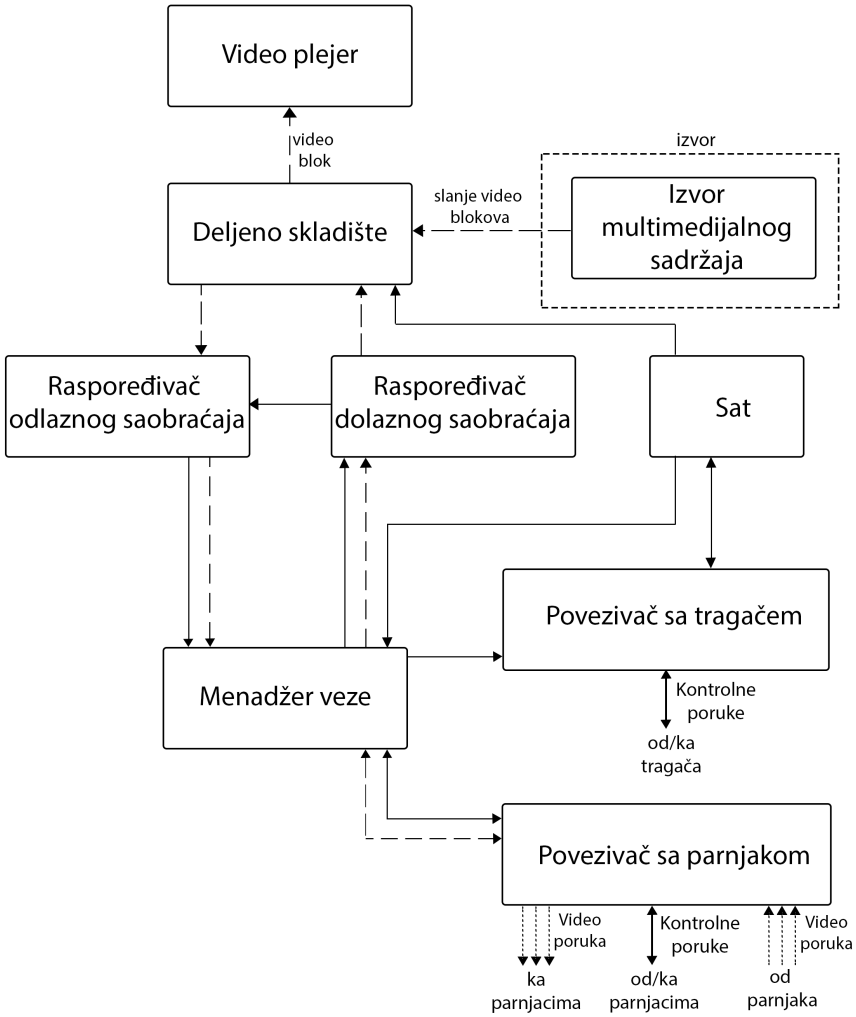
Implementacija parnjaka Kikkar protokola

Kako je Kikkar složen sistem bilo je neophodno izvršiti prepoznavanje odgovarajućih celina i razdvajanje sistema na delove. Pojednostavljen model softverske arhitekture parnjaka prikazan je na slici 4.2. Identičan model koristi i parnjak i izvor. Jedina razlika koja razdvaja parnjaka od izvora u modelu je ta što izvor sadrži dodatnu komponentu - izvor multimedijalnog sadržaja. Ova komponenta omogućava izvoru da uzima multimedijalni sadržaj koji je potrebno distribuirati kroz mrežu. Sadržaj može biti unapred pripremljen ili trenutno emitovan. U nastavku sledi opis svakog dela modela, njegovog zaduženja i opis podataka koje sadrži.

Povezivač sa tragačem

Komponenta *povezivač sa tragačem* (u programu klasa `ServerConnector`) predstavlja modul koji se koristi za komunikaciju parnjaka i tragača. Komunikacija se odvija uz pomoć HTTP i NTP protokola. Komponenta od podataka čuva informacije o željenom kanalu i datum poslednje izmene podataka. Informacije o željenom kanalu koriste se za prosleđivanje tačnog upita o određenom kanalu tragača. Informacija o poslednjoj modifikaciji služi za smanjenje opterećenja tragača. Ukoliko se jato nije promenilo od poslednje modifikacije, tragač ignoriše zahtev za novim parnjacima.

Funkcionalnosti koje ova klasa obezbeđuje su učitavanje JSON fajla koji sadrži podatke o kanalu u okviru metode `loadJson(String json)`. JSON fajl obez-



Slika 4.2: Arhitektura parnjaka. Isprekidane linije označavaju video podatke, pune linije označavaju kontrolne poruke.

beđuje tragač prilikom registrovanja kanala. Kako se većini metoda tragača pristupa GET zahtevima neophodno je kreirati URL upite. Metod koji se koristi za tu svrhu je `createURL(String baseURL, Map<String, String> parameters)`,

gde *baseURL* predstavlja url na koji želimo da kreiramo upit. Mapa parametara predstavlja parametre koje želimo da dodamo upitu. Kreiranje zaglavlja HTTP poruke se izvršava u metodi `getURLConnection(URL url)`, gde je *url* konkretan url za koji se kreira upit. Za inicijalnu konekciju sa tragačem koristi se metoda `connectToServer(URL url)`. Za sinhronizaciju vremena sa serverom koristi se metod `synchronizeTime(NTPUDPClient client, String ntpServer)`. Argument *client* predstavlja klijentsku implementaciju NTP protokola, a argument *ntpServer* predstavlja adresu NTP servera. U okviru klasnih metoda pristupa se klasi *Sat* o kojoj će biti više reči u poglavlju 4.2. U samom protokolu tragač obezbeđuje tačno vreme, ali se ovde zbog jednostavnosti koriste javno dostupni NTP serveri. Bitno je primetiti da se na ovaj način ne umanjuje opštost protokola. Lista parnjaka dobija se pozivanjem metoda `getPeerInfoList(URL url)` dok se obaveštavanje tragača o aktivnosti vrši se metodom `sendStayAliveMessage(URL url)`. Prilikom napuštanja kanala iz bilo kog razloga koristi se metod `sendLeaveMessage(URL url)`.

Povezivač sa parnjakom

Komponenta *povezivač sa parnjakom* (u programu klasa `PeerConnector`) predstavlja modul koji se koristi za komunikaciju parnjaka sa ostatkom jata. Komunikacija se odvija uz pomoću UDP protokola. Ova komponenta je najniža komponenta za komunikaciju sa parnjacima i kao takva ne donosi nikakve odluke već samo izvršava naredbe viših komponenti. Modul čuva sve pakete koje je parnjak dobio preko mreže u blokirajućem redu kao i podatke o samom sebi.

Komponenta sadrži veliki broj metoda. Većina ovih metoda se ipak ne koristi direktno. U ovom radu opisaćemo samo metode koje bi prosečni korisnik klase želeo da koristi. Za razumevanje ostalih metoda dovoljno je razumeti protokol opisan u poglavlju 3.

Za slanje paketa neophodno je pozivanje metoda `void send(DatagramPacket sendDataPacket, DatagramSocket socket)`. Argumenti su paket *sendDataPacket* koji želimo poslati i *soket* preko koga želimo izvršiti slanje. Za primanje podataka sa mreže koristi se metod `void startRecievePacketLoop(DatagramSocket socket)`. U okviru ovog metoda puni se blokirajuć red.

Prvi zadatak parnjaka kod pristupanja jatu je razmena ping-pong poruke. Ping poruka se šalje metodom `void sendPingMessages(List<PeerInformation> neighbourPeers, ConnectionType connectionType, DatagramSocket socket, Short clubNum)`. Lista parnjaka *neighbourPeers* predstavlja listu potencijalnih

komšija, *connectionType* predstavlja tip konekcije opisane u poglavlju 4.2, *socket* predstavlja apstrakciju soketa preko koga se vrši saobraćaj a *clubNum* broj kluba kome parnjak pripada. U okviru ove metode svakom od parnjaka iz potencijalne liste komšija šalje se ping poruka i postavlja se novo stanje parnjaka na čekanje pong poruke. Pong poruka se šalje metodom `void sendPongMessage(List<PeerInformation> neighbourPeers, PeerInformation peer, PingMessage pingMessage, DatagramSocket socket)`. Lista parnjaka *neighbourPeers* predstavlja listu već postojećih komšija, *peer* predstavlja parnjaka kome se odgovara na ping poruku *pingMessage* dok *socket* predstavlja apstrakciju soketa preko koga se vrši saobraćaj. U okviru ove metode proverava se koliko veza i uskladištenih podataka ima parnjak i šalje pong poruka.

Nakon što su razmenjene ping-pong poruke šalju se request-response poruke. Request poruka se šalje metodom `void sendRequestMessage(List<PeerInformation> neighbourPeers, DatagramSocket socket, ConnectionType connectionType)`. Argument *neighbourPeers* odgovara listi parnjaka sa kojima želimo da se povežemo, *socket* je apstrakcija soketa a *connectionType* tip konekcije koju želimo da uspostavimo. U okviru ove metode svakom parnjaku iz liste se šalje request poruka i postavlja stanje na čekanje response poruke. Response poruka se šalje metodom `void sendResponseMessage(PeerInformation peer, PacketWrapper packet, DatagramSocket socket)`, gde *peer* predstavlja parnjaka kome se šalje *packet* preko *socket*-a. Nakon što je response poruka poslata postavlja se status parnjaka na povezano.

Za slanje keep-alive poruke parnjaku opisane u poglavlju 3.4 koristi se metod `void sendKeepAliveMessage(PeerInformation peer, DatagramSocket socket)`. Argument *peer* odgovara parnjaku kojem se šalje poruka dok *socket* predstavlja apstrakciju soketa preko koga se odvija saobraćaj.

Menadžer veze

Komponenta *menadžer veze* (u programu klasa `ConnectionManager`) predstavlja modul koji se koristi za održavanje topologije parnjaka. Glavni zadatak ove komponente je da sakrije topologiju mreže od viših nivoa aplikacije i da koordiniše komunikaciju između tragača i parnjaka. U zavisnosti da li je aplikacija pokrenuta u režimu izvora ili parnjaka, menadžer veze istim metodama različito održava topologiju i procesira pakete u skladu sa protokolom. Modul od podataka čuva reference ka povezivaču sa tragačem i parnjakom, soket, podatke o trenutnom kanalu, listu

svih parnjaka, listu svih dobijenih pong poruka, sat, pakete za viši nivo aplikacije u blokirajućem redu kao i pristupni token tragača.

Za učitavanje JSON fajla koristi se metod `void loadJson(String rawJson)`. Ovaj metod samo prosleđuje parametre istom metodu povezivača sa tragačem. Da bi se pokrenulo osluškivanje i procesiranje paketa koristi se metod `void start()`. U okviru ovog metoda pokreće se u zasebnoj niti metod za primanje podatka klase povezivač sa parnjakom. Takođe, u okviru nje se radi kontrola zagušenja i eventualno slanje keep-alive poruke komšijama 3.4, kao i provera i održavanje topologije 3.2. Za svaki primljeni paket proverava se da li je u dobrom redosledu i prosleđuje metodu za obradu paketa. U okviru ove metode svi paketi koji ne predstavljaju video sadržaj već kontrolne poruke se obrađuju u skladu sa protokolom.

Kako bismo poslali pakete svim komšijama koristimo metod `void sendAll( PacketWrapper.Builder wrap, List<String> uninterestedPeerIp, PeerStatus peerStatus)`. Argument *wrap* predstavlja poruku koja se šalje, lista *uninterestedPeerIp* predstavlja listu komšija koji nisu zainteresovani za poruku dok *peerStatus* predstavlja status komšija kojima se šalje poruka. Svim komšijama koji imaju odgovarajući status i ne nalaze se na listi nezainteresovanih biće isporučena poruka. Najpre se poruka isporučuje komšijama u klubu pa tek onda ostalima da bi se što brže distribuirala. Ukoliko poruku želimo da pošaljemo jednom parnjaku koristi se metod `void sendOne(PacketWrapper.Builder wrap, String IpAddress)` gde *wrap* predstavlja poruku a *IpAddress* adresu parnjaka kome se poruka šalje. Ukoliko želimo da pošaljemo poruku svim komšijama u klubu koristimo metod `void sendToClub(PacketWrapper.Builder wrap, PeerStatus peerStatus, int clubNum)`. Kao i u prethodnim metodama argument *wrap* predstavlja poruku koja se šalje, *peerStatus* predstavlja status komšija kojima se šalje poruka a *clubNum* broj kluba.

Sat

Komponenta *sat* (u programu klasa `ClockSingleton`) predstavlja modul koji se koristi za ujednačavanje vremena svih parnjaka u mreži. Sat koristi unikat (eng. *singleton*) uzorak za projektovanje. Pri razvoju aplikacija često je potrebno omogućiti korišćenje samo jedne instance određene klase iz konceptualnih ili arhitekturalnih razloga. Treba biti oprezan pri korišćenju unikata jer sa njim uvodimo globalno stanje u aplikaciju čime se povećava njena kompleksnost. U većini slučajeva ubrizgavanje zavisnosti (eng. *dependency injection*) treba koristiti umesto

unikata. Korišćenjem ubrizgavanja zavisnosti ne postoji direktna zavisnost između klasa aplikacije jer objekat ne zna koja je klasa u pitanju već zna samo njen javni interfejs. Međutim, u aplikaciji se klasa sat koristi u različitim delovima programa te bi korišćenje ubrizgavanja zavisnosti previše zakomplikovalo kreiranje objekata željenih klasa pa se iz tog razloga koristi unikat.

Formiranje i dobijanje unikat objekta ostvaruje se korišćenjem metode `ClockSingleton getInstance()`. Takođe treba obratiti pažnju i na to da je konstruktor deklarisan sa privatnim pravima pristupa. Klasa koristi vremensku zonu Europe/Belgrade. Da bismo dobili vreme koje je isto svuda u mreži koristimo metod `Long getcurrentTimeMilliseconds()`.

Raspoređivač dolaznog saobraćaja

Komponenta *raspoređivač dolaznog saobraćaja* (u programu implementirana klasom `DownloadScheduler`) predstavlja modul koji se koristi za menadžerisanje video sadržaja i kontrolnih poruka koje su povezane sa video sadržajem. Podaci koje klasa čuva su deljeno skladište za video sadržaj, menadžer veze za komunikaciju sa mrežom, raspoređivač odlaznog saobraćaja kojem se prosleđuju potrebne akcije za slanje video sadržaja i video kontrolnih poruka kroz mrežu. Razlikuju se metodi za procesiranje paketa u zavisnosti da li je aplikacija pokrenuta u modu parnjaka ili izvora.

Da bi se pokrenulo preuzimanje odgovarajućih paketa od menadžera veze koristi se metod `void startDownload()`. Za obradu paketa u skladu sa opisanim protokolom koristi se metod `void processPacket(Pair<String,PacketWrapper> packetPair)`. U okviru ove metode poziva se i raspoređivač odlaznog saobraćaja ukoliko za to postoji potreba.

Raspoređivač odlaznog saobraćaja

Komponenta *raspoređivač odlaznog saobraćaja* (u programu implementirana klasom `DownloadScheduler`) predstavlja modul koji se koristi za menadžerisanje slanja video sadržaja i kontrolnih poruka koje imaju veze sa video sadržajem. Podaci koje klasa čuva su deljeno skladište za preuzimanje video sadržaja, menadžer veze za komunikaciju sa mrežom za dalje prosleđivanje poruka. Razlikuju se metodi za procesiranje paketa u zavisnosti da li je aplikacija pokrenuta u modu parnjaka ili izvora.

Za slanje kontrolne poruke, opisane u poglavlju 3.5 koristi se metod `void sendControlMessage(ControlMessage message)`. Za slanje nezainteresovane poruke, opisane u poglavlju 3.3, koristi se metod `void sendNotInterested(Pair<String, PacketWrapper> haveMessagePair)`. Za slanje video snimka koristi se metod `void sendVideo(int currentVideoNum, List<String> currentVideoNotInterestedIpAddresses)`, gde *currentVideoNum* odgovara jedinstvenom broju video snimka koji je potrebno poslati dok *currentVideoNotInterestedIpAddresses* IP adresama parnjaka koji nisu zainteresovani za sadržaj. Za odgovor na traženje video snimka koristi se metod `void sendVideo(void sendResponseMessage(Pair<String, PacketWrapper> packetPair, int[] videoNum)`. Za pronalaženje nedostajajućih video snimaka koristi se metod `void getMissingVideoNum()`.

Deljeno skladište

Komponenta *deljeno skladište* (u programu klasa `SharingBufferSingleton`) predstavlja modul koji se koristi za skladištenje multimedijalnog sadržaja i njegovo prosleđivanje video plejeru i sinhronizaciju video sadržaja. Koristi unikat obrazac za projektovanje na isti način i iz istog razloga kao i komponenta sat. Klasa čuva objekat klase sat, niz objekata video snimaka i objekat video plejer.

Pri korišćenju VLC plejera neophodno je zapisati podatak iz skladišta u fajl. Zapisivanje se ostvaruje metodom `void saveVideoPack(OutputStream os, int lastControlMessageVideoNum)`. U okviru ove metode osim snimanja sadržaja u fajl, briše se zastareli video sadržaj. Da bi se dobio broj trenutno uskladištenih podataka koristi se metod `int getNumberOfBufferedVideoContent()`. Za sinhronizaciju video sadržaja metodima opisanim u poglavlju 3.5 koriste se metodi `void synchronizeVideoPlayTime(int currentVideoNum, ControlMessage controlMessage)` i `void synchronizeVideoPlayTime(int currentVideoNum)` gde je *currentVideoNum* trenutni jedinstveni identifikator video sadržaja koji se prikazuje a *controlMessage* kontrolna poruka koja se prikazuje.

Video plejer

Komponenta *video plejer* (u programu klasa `VLCPlayer`) predstavlja modul koji se koristi za prikazivanje multimedijalnog sadržaja. Korsti se VLCj radni okvir za pokretanje instance nativnog VLC medijskog plejera i skrivanje kompleksnosti rada

sa LibVLC. VLCj radi na Linux, Windows i MacOSX operativnim sistemima. Od podataka klasa sadrži instancu plejera i putanju do video sadržaja.

Za prikazivanje video snimka koristi se metod `void playVideo()`. Dobijanje dužine trenutno prikazanog sadržaja vrši se metodom `int getCurrentPlayTime()`. Da bi se izvršila sinhronizacija video plejera sa izvorom koristi se metod `void synchronizeVideo(int sourceVideoTime)` gde *sourceVideoTime* predstavlja dužinu emitovanog snimka plejera.

Izvor multimedijalnog sadržaja

Komponenta *izvor multimedijalnog sadržaja* (u programu implementirana klasom `SourceVideoLoader`) predstavlja modul koji se koristi za prikupljanje multimedijalnog sadržaja. Multimedijalni sadržaj može biti snimljen ili emitovan. Ova komponenta se izdvaja od svih prethodno opisanih komponenti jer se povezuje sa ostalim komponentama samo u slučaju moda aplikacije izvor. Od podataka sadrži sat, deljeno skladište i raspoređivač odlaznog saobraćaja.

Učitavanje video sadržaja obezbeđeno je metodom `void loadVideo(String inputVideoPath, String outputVideoPath)` gde *inputVideoPath* odgovara putanji na kojoj se nalazi video sadržaj a *outputVideoPath* putanji na kojoj plejer očekuje video sadržaj. Vrši se učitavanje pojedinačnih blokova a zatim se čeka određeno vreme pre učitavanja sledećeg. Potrebno vreme čekanja zavisi od karakteristika video snimka. Vreme čekanja se uvodi kako bi se sprečilo zagušenje izvora. Za čitanje dela video bloka i njegovo slanje u jato koristi se metoda `void readChunk(InputStream is, int chunkNum)`. Vreme čekanja zavisi od brzine kojom se blokovi šalju i broja blokova. Najpre se svaki blok snimi u deljeno skladište a zatim distribuira u mrežu.

Glava 5

Eksperimentalna testiranja

U okviru ovog poglavlja analiziran je Kikkar protokol. Za ispitivanje protokola koristi se aplikacija opisana u poglavlju 4. Da bi davaoci usluga procenili kada treba primeniti predloženi protokol, neophodno je razumeti njegova svojstva i ograničenja.

Kikkar protokol se bazira na Screamer protokolu. Preciznije, Kikkar protokol nadograđuje Screamer protokol dajući mu mogućnost slanja sinhronizovanog sadržaja. Do danas postoji jako malo javno dostupnih studija koje analiziraju performanse Screamer protokola. U poglavlju 5.1 delimično će se koristiti rezultati dobijeni iz dostupne literature zajedno sa rezultatima koje je autor ovog rada dobio eksperimentalnim metodama. U datom poglavlju opisan je teorijski model mreže, način vršenog testiranja i količina dodatnog saobraćaja u mreži.

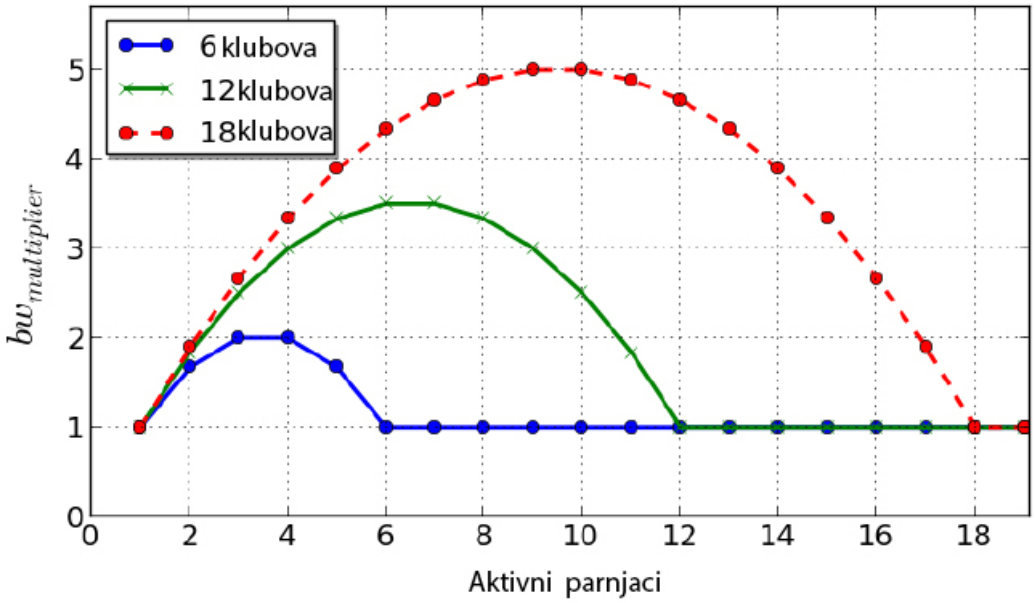
5.1 Dobijeni rezultati

Pre eksperimentalnih rezultata testiranja aplikacije biće opisan teorijski model predstavljen u radu [36]. Model je izgrađen na osnovu Screamer protokola a zatim pojednostavljen. Služi za opisivanje distribucije podataka i zahtevanog propusnog opsega izvora u najboljoj i najgoroj situaciji. U okviru modela određuje se trenutak u kome mreža postaje samodovoljna tj. trenutak kada novi parnjaci kompletan sadržaj dobijaju od drugih parnjaka i izvor ne učestvuje u deljenju.

Radi lakše analize pretpostavlja se da svi parnjaci imaju isti propusni opseg. U realnom slučaju kada parnjaci dolaze i odlaze iz mreže propusni opseg izvora se može izračunati formulom:

$bw_{dynamic} = b_{trans} X \sum_{i=0}^{\lceil c/2 \rceil - 1} (1 - \frac{2i}{c})$

gde je c ukupan broj klubova a b_{trans} bitrejt koji video snimak zahteva da bi se pravilno reprodukovao. U slučaju da bitrejt nije konstantna vrednost, uzima se prosečna vrednost.



Slika 5.1: Potrebni propusni opseg odlaznog saobraćaja izvora (slika je preuzeta iz rada [36])

Iz formule se jasno vidi da za 6 klubova izvor mora imati propusni opseg 2* bitrejt, dok za 12 klubova ova mera iznosi 3.5*bitrejt. Sa slike 5.1 se zaključuje da potrebna brzina odlaznog saobraćaja parnjaka neprekidno raste sve dok postoji jedan član u $\lceil c/2 \rceil$ klubova. Kada mreža sadrži više od $\lceil c/2 \rceil$ parnjaka, potrebna brzina odlaznog saobraćaja lagano opada. Kada u svakom klubu postoji bar jedan parnjak, mreža se stabilizuje i neophodni odlazni saobraćaj izvora je jedank bitrejtu video snimka koji šalje. Postojanje sve većeg broja klubova zahteva sve veći odlazni opseg izvora ali i manji propusni opseg parnjaka kako postoji manje ponovljenog sadržaja. Navedeno svojstvo važi i u obrnutom slučaju.

Mrežni saobraćaj je prikupljen korišćenjem alata Whireshark [47]. Wireshark je besplatan programski alat otvorenog koda koji služi za analizu mrežnih paketa. On prikuplja podatke koji se šalju u paketima kroz mrežu i prikazuje sve njihove detalje. Pored mogućnosti prikazivanja omogućava i sortiranje paketa u klase prema zadatom kriterijumu. Ovo nam omogućava da detaljno pratimo stanje mreže i saobraćaj koji

se na njoj odvija. Neke od najčešćih primena Wireshark-a su pomoć pri otklanjanju problema na mreži, razvoj i implementacija novih protokola i otkrivanje sigurnosnih propusta.

Svi eksperimenti su rađeni na video snimku koji ima prosečan bitrejt od 720 kbit/s. Da bi se dobio što veći broj parnjaka po klubu korišćena su 3 kluba sa ukupno 6 parnjaka. Parnjaci su raspoređeni na četiri računara tako da su dva računara pokretala po jednog parnjaka, a preostala dva računara po dva. Kao parnjak na kome su vršena merenja uzet je onaj koji se nalazio na računaru koji je pokretao samo jednog parnjaka. Usled binarne prirode biblioteke *protocol buffer* za razmenu sadržaja nije moguće lako odrediti koji paket predstavlja snimak a koji kontrolne poruke tako da je evaluacijom prikazana samo količina dodatnog saobraćaja koju protokol uvodi. Dodatni saobraćaj predstavlja sve podatke koji se šalju a ne predstavljaju sam video sadržaj.

Kako podaci dobijeni Whireshark alatom obuhvataju i pakete koji nemaju veze sa protokolom, za čišćenje dobijenog csv fajla koristi se python skripta obrada_csv.py. Kao biblioteka za obradu dobijenog csv fajla koristi se pandas [32]. Pandas je biblioteka otvorenog koda koja pruža visoke performanse za ceo proces analize i manipulacije podacima. U okviru ove skripte uzimaju se samo paketi koji su razmenjeni između parnjaka i servera. Iz dobijenih paketa se zatim izdvaja njihova veličina.

Rezultati pokazuju da je parnjaku bilo neophodno da primi 49 598 526 bajtova podataka da bi dobio video snimak veličine 45 771 008 bajtova odakle zaključujemo da je na dodatne podatke otišlo 4 827 518 bajtova. Drugim rečima, dodatni saobraćaj obuhvata 9,73% ukupnog saobraćaja. Dobijeni rezultat je u rangu sa ostalim protokolima slične namene. Sa druge strane, poslato je 16 199 180 bajtova podataka. Razlog za ovakav disbalans posledica je velikog talasanja mreže pa je parnjak neko vreme bio bez odlaznih veza. Takođe, veliki deo tereta preuzeo je izvor usled malog broja parnjaka i klubova.

Glava 6

Zaključak i budući radovi

U ovom radu predstavljen je protokol Kikkar koji omogućava sinhronizovano prikazivanje video sadržaja kroz mrežu upotrebom mreže ravnopravnih računara. Pokazano je da efekat mreže ravnopravnih računara važi kada se u jatu nalazi šest ili više parnjaka. Takođe, pokazano je da je udeo dodatnog saobraćaja manji od 10% što je u okviru standarda ovakvih mreža.

Glavne karakteristike Kikkar protokol preuzima od Screamer protokola. Najvažnije karakteristike su da održava veoma mali početni zastoj i malo vreme propagacije. Kikkar uvodi i sinhronizovan prikaz sadržaja kroz celu mrežu za razliku od Screamer protokola. Pored opisa protokola nalazi se i implementacija protokola u programskom jeziku Java, sa ciljem demonstracije mogućnosti protokola i evaluacije rešenja. Takođe, evaluacijom rezultata prikazano je da je postignuta jednaka efikasnost protokola u odnosu na protokole koji ne omogućavaju sinhronizaciju.

Buduća unapređenja mogla bi da obuhvate implementiranje nekog vida provere validnosti dobijenog sadržaja. Trenutna verzija protokola pretpostavlja da je sav sadržaj za reprodukciju koji parnjak dobije stigao od izvora što u slučaju zlonamernih korisnika ne mora biti ispunjeno. Može se desiti da je u međuvremenu paket zamenjen malicioznim paketom. Takođe, trenutna verzija protokola je primenljiva samo u okviru lokalne mreže zbog manjka jedinstvenih adresa kod IPv4 protokola i primene nestandardizovanih prevođenja mrežnih adresa (eng. *Network Address Translation, NAT*) za rešavanje problema. U narednim godinama se predviđa potpuni prelazak na IPv6 protokol pa će biti omogućeno korišćenje Kikkar protokola i na vebu. Takođe, trenutno postoji značajno vreme čekanja koje proizvodi video plejer. Ukoliko bi se omogućio niži pristup video plejeru i mogućnost finog podešavanja bafera, značajno bi se smanjilo vreme čekanja od trenutka pridruživanja mreži

do trenutka prvog reprodukovanog sadržaja.

Bibliography

[1] K. Aberer, A. Datta, and M. Hauswirth. „Efficient, Self-Contained Handling of Identity in Peer-to-Peer Systems". In: *IEEE Transactions on Knowledge and Data Engineering* 16.07 (July 2004), pp. 858–869. DOI: 10.1109/tkde.2004.1318567. URL: https://doi.org/10.1109%2Ftkde.2004.1318567.

[2] *Apache Commons Net Opis*. https://commons.apache.org/proper/commons-net/. 2018.

[3] *Apache Maven Opis*. https://maven.apache.org. 2018.

[4] *Apache Tomcat Opis*. http://tomcat.apache.org/. 2018.

[5] Josh Beggs and Dylan Thede. *Designing Web Audio*. Ed. by Richard Koman. 1st. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999. ISBN: 1565923537.

[6] *Bittorent Opis*. http://www.bittorrent.com/.

[7] *Bootstrap Opis*. https://getbootstrap.com/docs/4.1/getting-started/introduction/. 2018.

[8] Fernando Bordignon and Gabriel Tolosa. „Gnutella: Distributed System for Information Storage and Searching Model Description". In: (Aug. 2001).

[9] Damiano Carra, Renato Lo Cigno, and Ernst W Biersack. „Graph based analysis of mesh overlay streaming systems". In: *IEEE Journal of Selected Areas in Communications, Volume 25, December 2007* (Dec. 2007). DOI: http://dx.doi.org/10.1109/JSAC.2007.071206.

[10] Ian Clarke et al. „Freenet: A Distributed Anonymous Information Storage and Retrieval System". In: *Designing Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2001, pp. 46–66. DOI: 10.1007/3-540-44702-4_4. URL: https://doi.org/10.1007%2F3-540-44702-4_4.

[11] Bram Cohen. *Peer-to-Peer live streaming*. US Patent 4/801,778. May 2011. URL: https://patents.google.com/patent/US20150326657.

[12] Jorn De Boever. „Peer-to-Peer Networks as a Distribution and Publishing Model". In: (Jan. 2007).

[13] Stephen E. Deering and David R. Cheriton. „Multicast Routing in Datagram Internetworks and Extended LANs". In: *ACM Trans. Comput. Syst.* 8.2 (May 1990), pp. 85–110. ISSN: 0734-2071. DOI: 10.1145/78952.78953. URL: http://doi.acm.org/10.1145/78952.78953.

[14] *Eclipse Opis*. https://www.eclipse.org/. 2018.

[15] J. Esch. „Peer-to-peer media streaming: insights and new developments". In: *Proceedings of the IEEE* 99.12 (Dec. 2011), pp. 2087–2088. ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2170752.

[16] *FastTrack Opis*. http://ntrg.cs.tcd.ie/undergrad/4ba2.02/p2p/index.html.

[17] Prasanna Ganesan, Krishna Gummadi, and H. Garcia-Molina. „Canon in G major: designing DHTs with hierarchical structure". In: *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004. DOI: 10.1109/icdcs.2004.1281591. URL: https://doi.org/10.1109%2Ficdcs.2004.1281591.

[18] *Gson Opis*. https://github.com/google/gson. 2018.

[19] Chourouk Hammami et al. „Hybrid Live P2P Streaming Protocol". In: *Procedia Computer Science* 32 (2014), pp. 158–165.

[20] *Hibernate Opis*. https://hibernate.org/orm/documentation/5.3/. 2018.

[21] *I2P Dokumentacija*. https://geti2p.net/en/docs.

[22] *Jackson Opis*. https://github.com/FasterXML/jackson. 2018.

[23] *Java Opis*. http://www.java.com. 2018.

[24] *JQuery Opis*. https://api.jquery.com/. 2018.

[25] *JSTL Opis*. https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/. 2018.

[26] *Junit Opis*. https://junit.org/junit5/. 2018.

[27] Jian Liang, Rakesh Kumar, and Keith W Ross. „The kazaa overlay: A measurement study". In: (Oct. 2004).

[28] Petar Maymounkov and David Mazières. „Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems*. Springer Berlin Heidelberg, 2002, pp. 53–65. DOI: 10.1007/3-540-45748-8_5. URL: https://doi.org/10.1007%2F3-540-45748-8_5.

[29] *Mockito Opis*. https://site.mockito.org/. 2018.

[30] *Mysql Opis*. https://dev.mysql.com/. 2018.

[31] A. T. Nguyen, B. Li, and F. Eliassen. „Chameleon: Adaptive Peer-to-Peer Streaming with Network Coding". In: *2010 Proceedings IEEE INFOCOM*. Mar. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5462032.

[32] *Pandas Opis*. https://pandas.pydata.org/pandas-docs/stable/. 2018.

[33] *Protocol buffers Opis*. https://developers.google.com/protocol-buffers/. 2018.

[34] *Quartz Opis*. https://qz.com/. 2018.

[35] Antony Rowstron and Peter Druschel. „Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *Middleware 2001*. Springer Berlin Heidelberg, 2001, pp. 329–350. DOI: 10.1007/3-540-45518-3_18. URL: https://doi.org/10.1007%2F3-540-45518-3_18.

[36] Julius Rückert, Tamara Knierim, and David Hausheer. „Clubbing with the Peers:A Measurement Study of BitTorrent Live". In: Sept. 2014. DOI: 10.1109/P2P.2014.6934295.

[37] Robert R. Schaller. *Moore's law: Past, present, and future*. IEEE Spectr.,34(6):52–59, 1997.

[38] Xuemin Shen et al. *Handbook of Peer-to-Peer Networking*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 0387097503, 9780387097503.

[39] Tara Small, Ben Liang, and Baochun Li. „Scaling laws and tradeoffs in peer-to-peer live multimedia streaming". In: Jan. 2006, pp. 539–548. DOI: 10.1145/1180639.1180754.

[40] *Spring Opis*. https://spring.io/. 2018.

[41] I. Stoica et al. „Chord: a scalable peer-to-peer lookup protocol for internet applications". In: *IEEE/ACM Transactions on Networking* 11.1 (Feb. 2003), pp. 17–32. DOI: 10.1109/tnet.2002.808407. URL: https://doi.org/10.1109%2Ftnet.2002.808407.

[42] „Telecom, ATIS Telecom Glossary 2007". In: *American National Standard for Telecommunications* (2010). URL: http://www.atis.org.

[43] Philip A. Chou Venkata N. Padmanabhan Helen J. Wang. „Resilient Peer-to-Peer Streaming". In: *IEEE* 16 (2003).

[44] *VLC Opis*. https://www.videolan.org/vlc/. 2018.

[45] *VLCJ Opis*. https://github.com/caprica/vlcj. 2018.

[46] David J. Wetherall and Andrew S. Tanenbaum. *Computer Networks*. Pearson Education, 2013. ISBN: 1292024224, 9781292024226.

[47] *Whireshark Opis*. https://www.wireshark.org/docs/. 2018.

[48] M. Zhang et al. „iGridMedia: Providing Delay-Guaranteed Peer-to-Peer Live Streaming Service on Internet". In: *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*. Nov. 2008, pp. 1–5. DOI: 10.1109/GLOCOM.2008.ECP.337.

[49] X Zhang, J Liu, and Baochun Li. „CoolStreaming/DONet: A dData-driven overlay network for live media streaming". In: *Proceedings of IEEE INFOCOM* (Jan. 2005).

[50] Xiangyang Zhang and Hossam Hassanein. „A Survey of Peer-to-peer Live Video Streaming Schemes - An Algorithmic Perspective". In: *Comput. Netw.* 56.15 (Oct. 2012), pp. 3548–3579. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2012.06.013. URL: http://dx.doi.org/10.1016/j.comnet.2012.06.013.

[51] B.Y. Zhao et al. „Tapestry: A Resilient Global-Scale Overlay for Service Deployment". In: *IEEE Journal on Selected Areas in Communications* 22.1 (Jan. 2004), pp. 41–53. DOI: 10.1109/jsac.2003.818784. URL: https://doi.org/10.1109%2Fjsac.2003.818784.