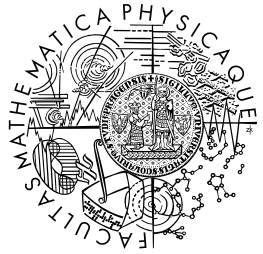

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁRSKA PRÁCA

Michal Medvecký

Peer-to-peer sieť na zdieľanie novinových článkov

Katedra softwarového inženýrství

Vedúci bakalárskej práce: RNDr. David Bednárek, Ph.D.

Študijný program: Informatika

Študijný obor: IPSS

Praha 2023

Prehlasujem, že som túto bakalársku prácu vypracoval(a) samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov. Táto práca nebola použitá k získaniu iného alebo rovnakého titulu.

Beriem na vedomie, že se na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona v platnom znení, najmä skutočnosť, že Univerzita Karlova má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školského diela podľa §60 odst. 1 autorského zákona.

V . . . . . dňa . . . .

Podpis autora

Moja vďaka patrí môjmu vedúcemu RNDr. Davidovi Bednárkovi, Ph.D. za prejavenú ochotu a čas počas písania tejto práce.

Ďalej by som rád poďakoval svojej rodine, kamarátom a susedom za podporu ako počas písania tejto práce, tak aj počas celého štúdia.

Názov práce: Peer-to-peer sieť na zdieľanie novinových článkov

Autor: Michal Medvecký

Katedra: Katedra softwarového inženýrství

Vedúci bakalárskej práce: RNDr. David Bednárek, Ph.D., Katedra softwarového inženýrství

Abstrakt: Jednoduchý, slobodný a bezpečný. Všetky tieto vlastnosti by mal spĺňovať každodenný prístup k informáciám v novinách. V tejto práci vytvoríme toto prostredie na princípe siete peer-to-peer. Navrhneme komunikačný protokol medzi peermi, zašifrujeme ich správy a overíme pravosť článkov, ktoré zo siete žiadajú. Zabezpečíme chod novín ako inštitúcie, spolu s publikáciou článkov a správy redaktorov. Čitatelia článkov sa ich stiahnutím zo siete stávajú aj ich sekundárnymi a spolu s novinami jedinými zdrojmi, zabezpečujúc ochranu informácií v sieti.

Kľúčové slová: peer-to-peer sieť internetové noviny P2P protokol

Title: Peer-to-peer network for newspaper publication

Author: Michal Medvecký

Department: Department of Software Engineering

Supervisor: RNDr. David Bednárek, Ph.D., Department of Software Engineering

Abstract: Easy, simple and free as in freedom. These are the requirements for everyday access to information in newspapers. In this thesis we will create such environment based on peer-to-peer network. We are designing a protocol for communication between peers, with messages being encrypted and with article verification upon download. Protocol ensures operation of newspaper as an institution with support for journalists and article publication. After an article is downloaded by the reader, it becomes available for further sharing inside the network. With readers and newspaper being the only sources of articles, we ensure protection of information inside the network.

Keywords: peer-to-peer network online newspaper P2P protocol

Obsah

Úvod

Slobodný prístup k informáciám sa v modernom svete stáva stále náročnejší. Veľké množstvo z nich je zavretých za tzv „paywallmi“, kde je ich obsah prístupný len po zaplatení istej peňažnej čiastky, alebo prekáža plynulému čítaniu veľké množstvo reklám. Pri pohľade na papierové noviny človek hneď rozozná jednotlivé články, avšak moderný web ide do značnej miery proti tomuto princípu.

Články internetových novín sú uložené na centrálnych serveroch a k ich prístupu je potrebné aktívne spojenie s týmto serverom. Po odpojení sa prístup k všetkým článkom stráca a bez použitia externých prostriedkov ich často nie je možné bez pripojenia prehliadať.

Problematické sú taktiež vysoké ceny za úložisko, webhosting a ďalšie služby, ktoré sú potrebné na prevádzku novín na webe. Vynútiť ochranu informácie môže byť až nemožné, najmä keď sú noviny umiestnené na dátových centrách v rôznych krajinách sveta s rôznorodou jurisdikciou. Zbieranie dát a informácií sa v poslednej dobe ukazuje ako každodenná rutina mnohých poskytovateľov týchto služieb, aj keď slúži len na ich interné účely.

Najmä menšie noviny s nízkymi príjmami s rádovo stovkami čitateľov trpia týmito problémami najviac a práve na ne je zameraná táto práca.

Výsledok práce taktiež poslúži aj na všeobecné zdieľanie informácií napríklad v rámci spoločnosti alebo inej uzavretej skupiny ľudí, čo by sme normálne za noviny nepovažovali.

Noviny ako platforma na šírenie informácií

Na šírenie informácií sa v digitálnom svete dá použiť značné množstvo rôznych technológií, najmä tých postavených na sociálnych sieťach. Zatiaľ čo však obsah na sociálnych sieťach buď to nie je moderovaný vôbec, alebo je moderovaný až po zverejnení, je možné do písať čokoľvek a z akéhokoľvek názorového spektra. Takáto moderácia môže však byť čisto amatérska a vykonávaná ľuďmi v ich voľnom čase. Ak však požadujeme, aby bol obsah moderovaný už od začiatku a teda aby aj bol tematicky jednotný, vytvoríme štruktúru, ktorej členmi budú len osoby, ktoré tvoria obsah týkajúci sa istej tematiky a podľa istých interných

pravidiel. Kvôli tomu nechceme, aby obsah tvoril ktokoľvek a teda bude mať spomínaná štruktúra istú hierarchiu. Na vyššej úrovni hierarchie sa nachádzajú osoby tvoriace obsah a na nižšej osoby, ktoré ho konzumujú. Vyšších členov hierarchie musí niekto určiť a teda zavedieme ešte jednu, najvyššiu vrstvu, ktorú však bude tvoriť len jedna osoba. Hierarchia má teda tri vrstvy, ktoré nazveme šéfredaktorskou, ktorú tvorí šéfredaktor, redaktorskou, ktorú tvoria redaktori. Tieto dve najvyššie vrstvy sú ako tvorcovia, tak aj konzumenti obsahu. Nakoniec je vrstva čitateľská, ktorú tvoria čitatelia, teda konzumenti obsahu. Celej tejto hierarchii dáme názov „noviny".

Pre noviny je teda typická ich uzavretosť, čo sa týka tematiky ich obsahovej stránky. Šéfredaktor, ktorý môže byť zároveň ich majiteľom, rozhoduje o osobách, ktoré môžu pre dané noviny tvoriť obsah, čím odpadá potreba jeho moderovania.

Nevýhody modelu klient-server a ich odstránenie

Sieť typu klient-server

Sieť typu klient-server tvorí jeden centrálny prvok, server, na ktorý sa pripájajú jednotliví klienti a získavajú z neho informácie. Takáto sieť má však niekoľko nevýhod.

Nevýhoda klient-server modelu spočíva najmä práve v jednom centrálnom prvku, serveri, okolo ktorého je celá sieť postavená. Bez tohto centrálneho servera je sieť v podstate nefunkčná, ak nie sú nasadené záložné zdroje, ktoré preberú rolu centrálneho prvku, ak by malo dôjsť k jeho výpadku. Problémom je však to, že aj tieto záložné zdroje často ovláda jedna entita, či už je to nejaká spoločnosť, alebo štát, pri ktorej zlyhaní, z dôvodu personálneho (napríklad štrajk zamestnancov), prírodných katastrof (napríklad pandémia) a ďalších môže stále dôjsť k výpadku celej siete. Čím je táto entita väčšia, tak tým jej šanca na zlyhanie klesá.

Dôležitá je ďalej ochrana informácie, pričom táto zodpovednosť leží v rukách tých entít, ktoré spravujú dátové centrá. Vezmime si napríklad zaujatosť zo strany takejto entity, ktorá by mohla svoje zdroje a prístup k dátam využiť na priemyselnú špionáž, alebo na cenzúru informácií. V tomto prípade čím väčšia entita je, tým väčší vplyv má a tým väčší môže byť jej negatívny dosah na informácie, ktoré spravuje.

Fyzická nemožnosť jednoducho a rýchlo preniesť dátové centrá spôsobuje, že sa tieto miesta stávajú náchylnými na všetky druhy fyzického poškodenia veľkého rozsahu, ako napríklad už spomínané prírodné katastrofy, teroristické útoky, alebo aj prevzatie politickej moci v danej krajine, v ktorej sa dátové centrum nachádza, totalitným režimom.

Odstránenie problémov siete typu klient-server

Na odstránenie problémov je potrebné rozdeliť úlohy, ktoré náležia jednému serveru ideálne tak, aby celá sieť spracúvala nejakú jeho časť a aby sa jednotlivé prvky siete o svoje úlohy delili. Takto zmizne potreba na jeden veľký, centrálny server a môže byť vytvorených niekoľko menších. Tieto už môžu by nasadzované vo veľkom množstve aj v menších dátových centrách, ktoré kontrolujú viaceré (menšie) entity.

Obsah dát by mal byť vždy šifrovaný na strane aplikácie a nespoliehať sa na šifrovanie na strane dátového centra a entity, ktorá ho spravuje. Takto sa zabezpečí odolnosť voči čiastočnej cenzúre informácií, avšak úplná cenzúra, spôsobená napríklad zmazaním obsahu pevných diskov obsahujúcich dané informácie, stále zostáva reálnou hrozbou.

Úlohy je možné dokonca rozdeliť až do takej miery, že každý účastník siete bude mať nejaký účel, ktorý bude nejakým spôsobom prospešný pre chod siete a bude zabezpečovať jej prevádzku. Sieť, založená na tomto princípe, kde každý jej člen sa podieľa na jej správnom fungovaní, je možné implementovať pomocou princípu peer-to-peer.

V sieti typu klient-server je potrebné platiť hosting u nejakej tretej strany, alebo investovať značné množstvo peňazí do kúpy vlastných serverov, ktoré budú obsah dávať k dispozícii verejnosti. So správou serverov, či už vlastných alebo cudzích prostredníctvom hostingu sú taktiež spojené problémy, ktoré vyžadujú často netriviálne technické znalosti z oblasti administrácie podobných systémov. Tieto náklady na hosting, vlastné servery, či administrátorov týchto systémov môžeme ušetriť práve vytvorením aplikácie, ktorá umožní dávať obsah k dispozícii verejnosti z počítačov redaktorov a šéfredaktora novín. Títo, najmä v dnešnej dobe, tak či tak tvoria obsah na počítačoch, ktoré teda môžme na jeho zdieľanie použiť. Je pravdou, že to pridáva isté riziko, pretože zatiaľ čo server je ťažké (ale nie nemožné) stratiť alebo ukradnúť, notebook redaktora je možné ukradnúť alebo stratiť omnoho jednoduchšie. Aplikácia však dáta perzistentne ukladá a je možné ich zálohovať. Výkon takýchto počítačov však je v porovnaní so servermi omnoho nižší, s čím klesá množstvo dát, ktoré môže daný redaktor zdieľať. V peer-to-peer prostredí však dochádza k rozloženiu nárokov na jednotlivých peerov a teda aj na výkon ich počítačov.

Sieť typu peer-to-peer

Sieť typu peer-to-peer je založená na decentralizovanom princípe, kde jednotlivý jej účastníci, nazývaní peeri, sa spoločne podieľajú na fungovaní tejto siete do takej miery, že každý jeden z nich tvorí istú časť siete. Sieť teda nie je postavená

okolo niekoľkých centrálnych prvkov, ale je úplne rozdelená medzi peerov. Tak tomu je len z fyzického uhla pohľadu, pričom však sa sieť javí ako jeden jednotný prvok.

Odstránenie jedného centrálneho prvku taktiež presunie zodpovednosť za ochranu informácie. Vzhľadom na to, že jednotlivé dáta sú rozdelené medzi ostatných členov siete, ktorí sú teraz za jej ochranu zodpovední.

Pre implementáciu novín nám prišlo ako vhodným riešením použiť sieť typu peer-to-peer. Aj keď hierarchia inštitúcie novín neumožňuje, aby došlo k úplnej decentralizácii, tak sú jednotliví peeri po technickej stránke úplne rovnocenní. Znamená to teda, že sieť, ako celok, umožňuje peerom si zakladať vlastné noviny, sťahovať si články z iných novín, publikovať svoje vlastné články pre buď to vlastné noviny alebo aj nejaké iné, kde môžu byť zamestnaní ako redaktori a podobne.

V celej sieti teda vznikajú rôzne podsiete, ktoré sú určené pre jedny konkrétne noviny. Jednotliví peeri týchto podsietí sa môžu nachádzať aj v ďalších podsieťach a mať pritom v danej podsieti úplne inú rolu. Rola v rámci jednej podsiete je síce proti princípu peer-to-peer z filozofického hľadiska, ale z hľadiska novín je istá centralizácia okolo šéfredaktora potrebná.

Komunikácia peerov

Pre jednotlivých peerov je potrebné navrhnúť, akým spôsobom budú spolu komunikovať, použitím zaužívaných prostriedkov z TCP/IP. Komunikácia musí zabezpečiť prenos všetkých potrebných informácií, ako napríklad identifikácia peera, jeho transportná adresa, jeho žiadosť o článok alebo zoznam článkov daných novín a podobne. Spôsob prenosu týchto informácií musí byť bližšie špecifikovaný, aby bol pre celú sieť a každého peera v nej jednotný.

Pravidlá prenosu informácií je vhodné zúžiť a špecifikovať pomocou komunikačného protokolu. V našom prípade sa jedná o protokol na aplikačnej vrstve, pričom na komunikáciu po sieti používame prostriedky z nižších vrstiev.

Ako rozumná voľba by bolo použiť iný protokol na aplikačnej vrstve. Ako príklad použijeme protokol HTTP. Prišlo nám však, že je príliš vysokoúrovňový a jeho korektná implementácia by vyžadovala veľké množstvo času, pričom by sme nakoniec nevyužili všetky jeho prostriedky a naopak, niektoré záležitosti, ktoré by sme chceli, aby protokol podporoval tak nejak sám od seba, by sme do týchto vysokoúrovňových protokolov museli doimplementovať sami, čo pri robustných a komplikovaných protokoloch nemusí byť jednoduché.

Použiť však nejaký protokol na aplikačnej vrstve takpovediac v „tandeme“ s našim protokolom je určite možné a možno aj vhodné, najmä keď sa jedná o problematiku, na ktorú je daný protokol úzko zameraný. Ďalej, ak je protokol

dostatočne jednoduchý, je spravidla aj ľahko rozšíriteľný a tak mu vieme jednoducho dodať potrebné funkcie, aby sme rozšírili jeho funkcionalitu tak, aby bol vhodný aj na náš účel.

Šifrovanie komunikácie

Aby sa zamedzilo nepovolaným osobám k získaniu informácií, ku ktorým nemajú mať prístup, napríklad pomocou odpočúvania sieťovej komunikácie, tak je žiadúce, aby bola táto komunikácia šifrovaná. Šifrovanie by malo prebiehať na oboch koncoch dvoch komunikujúcich peerov a nemalo by byť nijak závislé na nejakom centrálnom prvku siete, ak existuje, čo v našom prípade je šéfredaktor novín. Šifrovanie je teda univerzálne a vo funkčnom princípe jednotné pre celú sieť a nie len pre jedny noviny.

Ďalej je potrebné zabezpečiť, aby peeri, ktorí sú súčasťou siete, nemali prístup ku komunikácii ostatných peerov v sieti. Komunikácia každej dvojice peerov je teda samostatne izolovaná, útočník ju teda nedokáže odpočúvať len pomocou pripojenia samého seba do siete.

Univerzálna dostupnosť

Ďalej požadujeme, aby sieť bola univerzálne dostupná pre každého, kto sa do nej chce pripojiť. Takéto rozšírenie siete umožni lepšie rozloženie úloh jednotlivých peerov už len z toho princípu, že ich je viac a zvýši sa tak robustnosť siete.

Je taktiež vhodné zvážiť, napríklad v prípade platených novín, či by bolo možné zaviesť sieť, ktorá by bola len pre pozvaných peerov. Potrebné je taktiež vyriešiť otázku, ako by prebiehala správa členov v takejto sieti a či by bolo možné im odobrať práva prístupu do siete.

Replikácia dát na uzloch siete

Medzi úlohy peera v sieti patrí taktiež ďalšie zdieľanie článkov. Článok je teda stiahnutý na pevný disk, alebo iné pamäťové médium a následne, ak je vyžiadaný od niektorého z peerov, sa článok z disku prečíta a odošle. Článok je čitateľný a použiteľný aj v prípade, ak nie je peer do siete pripojený. Obsah je na disku buď to zašifrovaný, alebo je uložený v pôvodnom textovom formáte. V každom prípade je pri posielaní po sieti šifrovaný.

Správa redaktorov

Správu novín ako inštitúcie a jej hierarchiu musí sieť sprostredkovať prostredníctvom nástrojov pre šéfredaktora, ktoré poslúžia na povýšenie nejakého peera v podsieti pre dané noviny na redaktora.

Redaktor pridáva do novín články a vzájomne ich pomocou ostatných redaktorov a šéfredaktora indexuje. V prípade, že si niektorý z čitateľov vyžiada nejaký článok od daného autora, je jeho požiadavka rozposlaná ako ďalším členom siete, do ktorých patrí aj autor článku, alebo niekto z ostatných redaktorov. Aplikácie týchto peerov, ktorí daný článok majú, ho automaticky sprostredkujú žiadateľovi daného článku.

Smerovanie správ v modernej sieti

Z dôvodu vyčerpania adries IP verzie 4, skrátene IPv4, sú datagramy v modernej sieti často smerované smerovačmi, ktoré majú zapnutý preklad sieťových adries, skrátene NAT. Jedna lokálne sieť tak môže mať priradenú jednu adresu IP a pri pripájaní sa k vonkajšiemu svetu sa v smerovači vytvorí mapovanie, kde sa danej komunikácii pridelí transportná adresa, prostredníctvom ktorej je možné s daným peerom v lokálnej sieti komunikovať. Prítomnosť NAT-u spôsobuje, že nie je vždy možné, a v niektorých prípadoch to nie je možné nikdy, pripojiť sa zo siete k peerovi, ktorý leží za smerovačom s NAT-om. Navrhnutá sieť a jej protokol sa teda musia pokúsiť NAT nejakým spôsobom obísť a nadviazať spojenie aj s peerom, ktorý sa nachádza v lokálnej sieti za smerovačom s aktívnym NAT-om.

Tento problém budeme riešiť použitím protokolu STUN. Tento protokol slúži ako nástroj pri obchádzaní NAT-u a je jednoduchý na rozšírenie. Pridaním vlastných metód a atribútov tento protokol použijeme na náš vlastný účel a spôsob obchádzania NAT-u.

Kapitola 1

Špecifikácia zadania

1.1 Vrstvy modelu OSI, resp. TCP/IP

Sieťový komunikačný protokol pracuje na siedmej, aplikačnej vrstve sieťového OSI modelu, resp. na štvrtej, aplikačnej vrstve modelu TCP/IP. Na komunikáciu sa používajú prostriedky a protokoly nižších vrstiev.

1.2 Sieť peer-to-peer

Protokol je určený pre sieť typu peer-to-peer, čo znamená, že jednotliví členovia siete nie sú rozdelení medzi servery a klientov, ako tomu je v klient-server modeli, ale sú si v tomto ohľade rovnocenní a nazývajú sa peermi. Zatiaľ čo v klient-server modeli začína spravidla komunikáciu klient, v peer-to-peer sieti nie je toto poradie definované a započať komunikáciu môže ktorýkoľvek peer v sieti. Ak z technických príčin nie je možné túto požiadavku splniť, je potrebné čo najviac zjednotiť schopnosti a povinnosti jednotlivých peerov.

Tento protokol slúži ako súbor dohôd, ktoré jednotliví peeri musia dodržať, aby si navzájom dokázali medzi sebou zdieľať informácie. Štandardné povinnosti peera, ktoré vyplývajú z princípu peer-to-peer siete sú rozšírené o potreby protokolu tak, aby bola zabezpečená jeho prevádzka.

1.3 Noviny

Noviny reprezentujú inštitúciu, ktorá vydáva novinové články a ktorá je spravovaná šéfredaktorom. Tieto novinové články píšu redaktori novín, ktorým túto vlastnosť pridelí šéfredaktor. Novinové články si sťahujú a čítajú čitatelia daných

novín. Protokol definuje princípy, pomocou ktorých sú tieto novinové články zdieľané.

1.3.1 Novinová sieť

Jedny noviny tvoria tzv. sieť. Súčasťou tejto siete sú šéfredaktor daných novín, ich redaktori a čitatelia.

Úlohy jednotlivých členov siete

V rámci siete majú jednotliví jej členovia rôzne úlohy. Šéfredaktor je zakladateľom novín a rozhoduje o tom, kto je ich redaktorom. Šéfredaktor a redaktori môžu do siete pridávať a zo siete odoberať články. Čitatelia si môžu od redaktorov a šéfredaktora stiahnuť tieto články a uložiť si ich napríklad na pevný disk. Čitatelia sú schopní do siete zdieľať jednotlivé články, ktoré si stiahli, pričom im tento fakt poskytne šéfredaktor novín, resp. autor daného článku, teda redaktor, ktorý daný článok napísal. Článok zo siete nie je možné zmazať, pokiaľ ho má stiahnutý aspoň jeden čitateľ a je tento čitateľ pripojený do danej siete k daným novinám. Jednotliví členovia jednej siete si teda nie sú rovnocenní a sú usporiadaní v istej hierarchii. Toto obmedzenie vyplýva z toho, ako funguje inštitúcia novín v reálnom živote.

Šéfredaktor je teda správca siete pre dané noviny. Rozhoduje o tom, kto do siete môže vložiť ďalší obsah. Nerozhoduje už však o tom, kto daný obsah môže ďalej zdieľať.

1.4 Nadsieť

Z technického uhla pohľadu sú členmi tejto novinovej siete peeri a jedná sa teda o sieť typu peer-to-peer. Zjednotením jednotlivých sietí dostaneme tzv. nadsieť a v rámci tejto nadsiete si sú jednotliví peeri rovnocenní. Každý peer si môže založiť svoje vlastné noviny, každý peer sa môže stať redaktorom nejakých novín, aj viacerých, a každý peer môže byť čitateľom ľubovoľných novín. Jeden peer teda dokáže články vytvárať, sťahovať a spravovať ich na svojom disku. Peer má taktiež schopnosť vyžiadať si zoznam článkov od iného peera, ak je tento peer šéfredaktorom v niektorých novinách v rámci nadsiete.

V nadsieti sú peeri jednoznačne identifikovaní nejakým identifikátorom. Každý peer má taktiež priradenú sieťovú adresu a port protokolu na transportnej vrstve. Adresa protokolu IPv4, resp. IPv6 a číslo portu transportného protokolu spolu vytvárajú transportnú adresu.

1.5 Správy a ich sieťový prenos

Jednotky informácií, ktoré si jednotliví peeri vymieňajú, sa nazývajú správy. Správy obsahujú všetky potrebné dáta na to, aby boli dodržané všetky princípy, ktoré daný protokol vyžaduje od peerov nadsiete a aj šéfredaktora, redaktorov a čitateľov jednotlivých novín, sietí. Zvlášť je potrebné v správe nejakým spôsobom vymedziť verziu protokolu, ku ktorému daná správa náleží, hoci budúce vydania protokolu by mali byť o najviac spätne kompatibilné. Nejaká časť správy môže byť šifrovaná, pre zabezpečenie integrity novinových článkov.

1.5.1 Šifrovanie

Šifrovanie správ znamená, že potenciálny útočník, ktorý odpočúva sieťovú komunikáciu, nedokáže zo zašifrovanej správy získať pôvodnú správu. Zašifrovanú správu musí vedieť prijímateľ dešifrovať. Je preto potrebné buď to správu zašifrovať asymetrickou šifrou, alebo symetrickou. Použitý je druhý prístup, pričom je ale potrebné zabezpečiť prenos symetrického kľúča tak, aby nedošlo k jeho pozmeneniu a aby si ho útočník nemohol prečítať odpočúvaním sieťovej komunikácie medzi peermi. Na tento účel je možné použiť asymetrickú šifru a tento symetrický kľúč nie len zašifrovať, ale aj podpísať, aby sa overila identita odosielateľa.

Pri šifrovaní samotného obsahu správy, nesúcej obsah pre peerov, teda napríklad novinový článok, je potrebné šifrovať tak, aby bola správa nie len šifrovaná a teda pre útočníka, ktorý nemá symetrický kľúč k danej správe nerozlúštiteľná, ale aby bolo možné zistiť, ak by bolo nejakým spôsobom zo správou manipulované, teda je potrebné, aby bola zachovaná integrita správy počas jej prenosu.

1.5.2 Smerovač s NAT-om a brána firewall

Ďalším zásadným problémom pri peer-to-peer sieťach je preklad sieťových adries, anglicky Network address translation, skrátene NAT. Ak sa peer pripája z lokálnej siete k vonkajšiemu svetu, v smerovači, ktorý má zapnutý NAT, sa vytvorí mapovanie, ktoré pridelí nejakú transportnú adresu danému peerovi a akákoľvek sieťová komunikácia na túto transportnú adresu je následne presmerovaná priamo k danému peerovi. Túto transportnú adresu nazývame taktiež server-reflexívnou transportnou adresou (anglicky server-reflexive transport address), pričom používame terminológiu z [1] a [2].

Pripojenia z vonkajšieho sveta však nie sú, kvôli tomuto princípu, jednoduché. Peer totiž nemusí mať vytvorené mapovanie a akýkoľvek pokus o pripojenie k takémuto peerovi zlyhá, napríklad vypršaním času určenému na odpoveď. Dokonca, ak dané mapovanie neexistuje, tak nevieme ani určiť transportnú adresu, na ktorú by sa mal odosielateľ pokúsiť pripojiť. Niektoré implementácie NAT-u

neumožnia komunikovať s peerom v lokálnej sieti ani v tom prípade, že má na strane smerovača vytvorené mapovanie a prebieha nejaká komunikácia medzi ním a vonkajším svetom. Viac informácií o správaní sa NAT-u poskytujú adekvátne RFC, ako napríklad [3].

Obchádzanie NAT-u

Je preto potrebné nejakým spôsobom smerovač a jeho NAT obísť. Tento problém budeme riešiť dvomi spôsobmi. Najprv sa jeden peer pokúsi pripojiť k tomu druhému priamo, na transportnú adresu, či už sa jedná o jeho reálnu transportnú adresu, alebo o jeho server-reflexívnu transportnú adresu. Ak sa dané pripojenie nepodarí, predpokladáme, že bolo odmietnuté NAT-om. Preto sa pokúsime presmerovať túto komunikáciu cez nejakú tretiu stranu. V našom prípade sa jedná o šéfredaktora daných novín. Na to, aby mohol nejaký peer pristupovať do siete, je potrebná komunikácia so šéfredaktorom daných novín a teda s ním má peer nadviazané spojenie. Tento fakt použijeme k tomu, aby sme mu doručili obchádzkou správu, ktorú sa nám nepodarilo doručiť priamo.

Pre správne fungovanie siete je teda potrebné, aby mal šéfredaktor buď to verejnú, statickú IP adresu, alebo aby nakonfiguroval svoj smerovač tak, aby prepúšťal správy nášho protokolu, ak to konfigurácia daného smerovača umožňuje.

Brána firewall

Existenciu brán firewall budeme ignorovať a je zodpovednosťou užívateľa, aby ju nakonfiguroval tak, aby prepúšťala sieťovú komunikáciu nášho protokolu.

Prakticky je však pre obchádzanie brán firewall možné použiť prostriedky na obchádzanie NAT-u, keďže majú tieto dve technológie mnoho spoločných princípov.

1.6 Aplikácia

Súčasťou práce je taktiež aplikácia s užívateľským rozhraním, ktorá sprístupní hlavné prvky protokolu a siete. V aplikácii sú implementované všetky nástroje na to, aby mohol peer zabezpečiť chod siete. Aplikácia teda dokáže vytvoriť peera a prípadne aj jeho noviny, pridať noviny iného peera do svojho zoznamu a od takého peera si vyžiadať zoznam článkov jeho novín. Následne aplikácia umožňuje vyžiadať si článok od peera a jeho následné prijatie a uloženie na disk. To isté platí aj z druhej strany, teda aplikácia prijme požiadavku na článok, načíta ho z disku, zašifruje a odošle. Aplikácia generuje verejné a privátne kľúče a aj symetrické kľúče pre výmenu správ. Súčasťou je taktiež podpora pre smerovanie

aj cez NAT a to použitím protokolu STUN [2] a jeho rozšírením pre potreby ako protokolu, tak aplikácie.

1.6.1 Cieľová platforma

Aplikácia je určená pre 64-bitové vydania operačných systémov Linux a Windows. Napísaná je v jazyku C++ v štandarde C++17 a prekladaná prekladačom GNU C compiler (verzie aspoň 11), alebo prekladačom Clang (verzie aspoň 15), alebo prekladačom Microsoft Visual C++ (verzie aspoň 14). Technická dokumentácia sa skladá z textovej časti a dokumentácie doxygen, vygenerovanej z komentárov v zdrojovom kóde aplikácie. Generácia dokumentácie a aj celkový preklad a budovanie aplikácie sú riadené, pre všetky operačné prostredia, použitím platformy CMake.

Kapitola 2

Dokumentácia protokolu a logika aplikácie

Keďže v práci navrhujeme a implementujeme peer-to-peer sieť, ktorú budú tvoriť jej užívatelia, teda peeri, je potrebné, aby bola medzi nimi prebiehala koordinovaná komunikácia podľa istých pravidiel. Zavedieme si teda aplikačný protokol NP2PS, z anglického „Newspaper P2P Sharing protocol“. Tento protokol, ktorý navrhneme, nám poslúži na komunikáciu medzi jednotlivými peermi v sieti a teda na hlavnú úlohu siete, ktorou je vytváranie novín a zdieľanie ich obsahu, teda článkov. K aplikačnému protokolu patrí aj formát správ, ktoré si budú jednotliví peeri vymieňať. Formát týchto správ sme ponechali na Google Protocol buffers, ktoré slúžia práve na serializáciu údajov pomerne jednoduchým spôsobom.

2.1 Skladba entít protokolu – metadáta

Jednotlivé entity protokolu majú rôzne vlastnosti, ktoré sú pre rádových užívateľov relevantné len sporadicky, ak vôbec. Protokol však musí na takéto vlastnosti za každých okolností brať ohľad, nakoľko je ich existencia potrebná buď to pre entity samotné, alebo sú nevyhnutné pre správny chod protokolu. Tieto vlastnosti samé o sebe, bez entity ktorú opisujú, nemajú zmysel. Nazvime ich metadátami, teda „dátami o dátach“. Príkladom metadát sú napríklad EXIF informácie, ktoré sú vkladané k obrazovým súborom, najmä digitálnym fotografiám.

Jednotlivé entity majú rôzne metadáta, nakoľko ich vlastnosti, ktoré potrebujeme metadátami reprezentovať, sú odlišné.

Metadáta taktiež môžu poslúžiť ako kompaktnejšia a na pamäť menej náročná reprezentácia entity. Samotná entita tak nemusí byť vôbec v danej chvíli priamo k dispozícii a môžeme ju získať na základe údajov z metadát. V kontexte sieťového protokolu to znamená, že pokiaľ máme lokálne uložené metadáta entity, samotná

entita sa môže nachádzať na inom počítači alebo zariadení v sieti, pričom ju na základe metadát vieme zo siete použitím štandardných prostriedkov sieťovej komunikácie získať v kompletnej, nijak neokresanej forme. Správny návrh a použitie metadát teda vedie k zníženiu pamäťovej náročnosti programu a taktiež k optimálnejšiemu využitiu sieťovej komunikácie. V niektorých prípadoch postačí na reprezentáciu entity aj podmnožina jej metadát.

2.2 Inštitúcia novín a reprezentácia v protokole

Tradičné papierové noviny sú spravované a vydávané nejakou novinárskou spoločnosťou. Zamestnanci takejto spoločnosti spravujú noviny po ich obsahovej stránke (teda články novín), okrem iného.

Vzhľadom na to, že navrhujeme protokol na zdieľanie novinových článkov, je rozumné ho zamerať na obsahovú stránku novín. Protokol sa teda zaoberá ich článkami a ľuďmi, ktorých ich vytvárajú, spravujú a čítajú. Ostatné prevádzkové záležitosti, či už ekonomické, marketingové, alebo personálne, sú spravované nezávisle od navrhovaného protokolu.

Pri navrhovaní nášho protokolu uvažujeme pod pojmom noviny zjednodušenú variantu tradičnej novinovej inštitúcie, ktorá ale stále plní svoju pôvodnú funkciu informovania verejnosti v podobe čitateľov. Neskôr v sekcii sú vysvetlené jednotlivé roly v novinách a vzťahy medzi nimi. Rovnako návrh protokolu pre noviny s rádovo státisícami či miliónmi čitateľov a desaťtisícami článkov by bol mnohonásobne komplexnejší a najmä kvôli podpore väčšieho množstva rôznych rolí v novinách, bez ktorých sa veľké inštitúcie spravidla v dnešnej dobe nezaobídu, by z protokolu a aplikácií, ktoré by ho implementovali, vznikol monolitický systém, ktorý by sa časom zastaral, ako to s takýmito systémami býva. Obmedzenie na menšie noviny čo do počtu čitateľov a článkov zjednoduší celý návrh protokolu, ale zároveň bude výsledok stále dostatočne reprezentatívny.

2.2.1 Správa zamestnancov novín

Je teda potrebné nejakým spôsobom v protokole reprezentovať zamestnancov daných novín. Vzhľadom na to, že vytvárame protokol pre aplikáciu, ktorá sa stará výlučne o obsahovú stránku novín, navrhujeme protokol takým spôsobom, aby reprezentoval jedine tých zamestnancov, ktorí sa nejakým spôsobom starajú o tvorbu a správu ich obsahu, teda redaktorov. Noviny taktiež majú svojho riaditeľa, ktorý sa stará o prijímanie a vyhadzovanie redaktorov, pričom vzhľadom na náturu nášho protokolu plní túto úlohu ich šéfredaktor.

Protokol teda podporuje vytvorenie šéfredaktora spolu s vytvorením novín.

Ďalej protokol poskytuje podporu pre pridávanie nových redaktorov šéfredaktorom novín. Redaktorom novín sa môže stať akýkoľvek peer pripojený do siete daných novín, kde pripojením do siete novín sa rozumie, že sa daný peer stane ich čitateľom. Analýzou správy užívateľov a peerov sa zaoberá sekcia 2.4.

2.2.2 Správa obsahu novín

V každých novinách je potrebné vytvárať a neskôr aj spravovať ich obsah. O vytváranie obsahuje sa spravidla starajú redaktori daných novín. O ich neskoršiu správu sa starajú ako redaktori, tak aj šéfredaktor. Redaktori môžu mať rôzne špecializácie a medzi sebou sa nachádzať v netriviálnej hierarchii, chceme však zamedziť komplikáciám, ktoré môžu počas implementácie takýchto striktných a hlbokých hierarchiách nastať. Najmä chceme predísť tomu, aby došlo k výraznému obmedzeniu peer-to-peer vlastností protokolu.

Protokol teda stavia jednotlivých redaktorov ako seberovných, pričom ich nadriadeným je šéfredaktor novín, ktorý zároveň figuruje aj ako riaditeľ danej novinovej inštitúcie.

Protokol podporuje pridávanie a odoberanie súborov do siete. V našom prípade sa jedná o textové súbory, teda články. Voľba formátu je analyzovaná v podsekcii 2.3.1. Konkrétnej podobe komunikácie medzi peermi v prípade, že sa do siete pridá, alebo v sieti aktualizuje článok popisuje sekcia 2.5.

2.2.3 Konzumácia obsahu novín

Konzument obsahu novín sa v kontexte protokolu nazýva čitateľ. Čitateľ čítaním článkov novín konzumuje ich obsah. Čitateľ musí byť teda schopný nejakým spôsobom v týchto článkoch vyhľadávať a následne ich zo siete sťahovať. Aby sa optimalizovalo množstvo sieťového prenosu, po sieti sa prenášajú len metadáta, alebo hlavičky, článkov. V týchto hlavičkách sa nachádzajú nadpisy článkov a ďalšie potrebné informácie na identifikáciu článku v sieti. Tieto môžme použiť a vytvoriť zoznam článkov, v ktorom už čitateľ môže vyhľadávať. Takýto zoznam článkov môžme prenášať po sieti v menších častiach, aby sa predišlo príliš zdĺhavému odosielaniu a sťahovaniu veľkého množstva dát súčasne a blokovala sa tak ostatná komunikácia v protokole.

Výberom zo zoznamu článku sa článok stiahne lokálne a následne ho môže čitateľ ľubovoľne dlho a koľkokrát chce prečítať.

2.3 Článok

2.3.1 Formát súboru článku

Jednotlivé články sú uložené v súboroch a to hlavne z dvoch dôvodov. Prvým dôvodom je šetrenie pamäte, kde priestorová náročnosť programu rastie pomalšie, ak nie je potrebné, aby bol každý článok v pamäti neustále počas behu aplikácie načítaný. V pamäti sú načítané len jeho metadáta, ktoré sú ale priestorovo menšie, než články samotné. Druhým dôvodom je nárok na perzistentné ukladanie dát v čase, kedy aplikácia nie je spustená. Potenciálne formáty takýchto súborov je možné rozdeliť viacerými spôsobmi, kde jeden zo spôsobov je rozdelenie na formát textový a binárny. My sme vybrali formát textový. Voľba textového formátu súboru oproti binárnej forme je jasná, nakoľko samotný ukladaný text tvorí celú informáciu, ktorá čitateľa zaujíma. Použitie binárneho formátu by dávalo zmysel vtedy, ak by boli články pri ukladaní na disk, alebo počas sieťového prenosu, nejakým spôsobom upravované – napríklad komprimáciou.

Články sú teda ukladané do textových súborov, ktoré reprezentujú ich obsah. Ale aj pre textové súbory existuje viacero rôznych formátov, ktoré je možné použiť. Zhrnuli sme niekoľko kritérií, ktoré je potrebné pri voľbe formátu súboru článku dodržať.

Pre formát článkov je potrebné, aby podporoval základné formátovanie textu, delenie na odstavce, nadpisy. Vlastnosti formátu textového súboru vyplývajú z podoby článkov v printových novinách. Formátov súborov a ich značkovacích jazykov (z anglického „Markup languages“), ktoré podporujú takéto formátovanie textu je niekoľko. Ku príkladu slúžia Markdown, HTML, LaTeX, org-mode. Posledné tri spomínané sú pre bežného užívateľa náročnejšie na použitie, na vykreslenie vyžadujú komplikované mechanizmy, alebo sú natoľko ohýbané na najrôznejšie spôsoby použitia (HTML), že hoci existuje istý štandard, tak každý užívateľ má od daného jazyka iné očakávania, najmä kvôli tomu, že sa daný jazyk zriedkakedy používa samostatne.

Nakoniec je značkovací jazyk Markdown najprívetivejšou voľbou. Spĺňa naše požiadavky, ktoré sme si stanovili a zároveň je jednoduchý na vykreslenie. Rádový užívateľ od jazyka Markdown očakáva jeho prostotu. V prípade záujmu je možné súbor v Markdowne vykresliť rôznymi spôsobmi a podľa predstáv užívateľa samostatne, na rozdiel od značkovacieho jazyka HTML. Markdown taktiež vyžaduje, aby bol text delený na odseky a nadpisy tak, ako chceme aby sa nakoniec vykreslil. To vedie k tomu, že je možné súbory písané v Markdowne čítať aj v bežnom textovom prehliadači, alebo editore bez potreby formátovania. To je v kontraste s jazykom HTML, ktorý môže mať celý obsah na jednom riadku, čo ho robí ťažko čitateľným bez použitia externého vykresľovacieho nástroja.

2.3.2 Metadáta článku

V technickej dokumentácii sa metadátam článku taktiež hovorí hlavička článku. Aby sme mohli pri získavaní článku zo siete kontaktovať **autora** článku, potrebujeme jeho verejný identifikátor. To isté platí pre **noviny**, ktorých verejný identifikátor je v metadátach uložený. Pre ako noviny tak autora ukladáme aj mená, ktoré si pri zakladaní peera, resp. novín vybrali a to na použitie v aplikácii. Z dôvodu overenia autenticity článku si ukladáme v metadátach jeho **kryptografický hash**, ktorý je obzvlášť vhodný pri sťahovaní článku zo siete. Z technických dôvodov sme zaviedli ďalší hash, ktorý ale slúži na identifikáciu článku v rámci novín a nazveme ho **novinový identifikátor článku**. Tento nemá kryptografické vlastnosti, ale ani na daný účel nie je používaný. Ďalšia položka hlavičky, ktorá slúži na zjednodušenie navigácie v článkoch, ktoré nemáme lokálne stiahnuté je **nadpis článku**, ktorý aplikácia môže použiť napríklad v zozname článkov dostupných v daných novinách. Pri aktualizácii článkov používame časové položky ako **vytvorenie** a **posledná úprava**. Pri ukladaní článku na disk používame položku **verzie**, aby sme ich v **názve súboru**, v ktorom bude článok uložený, mohli odlíšiť. Jednotlivé mená a časy je možné taktiež použiť na triedenie článkov v aplikácii.

Kategórie článku

Špeciálnu položku metadát článku tvoria kategórie, kde každý článok môže patriť do ľubovoľného množstva kategórií. Článok nemusí patriť do žiadnej kategórie.

Kategórie by mali informovať čitateľa o tom, akej témy či problematiky sa daný článok týka. Ak daný článok do kategórie nepatrí, nemala by pri ňom byť daná kategória uvedená. Taktiež sú vhodné pri triedení článkov, kde čitateľ môže hľadať článok práve zvolením jednej kategórie, alebo viacerých kategórií.

Kategórie nie sú nič iné, než textové reťazce, ktoré nie sú nijakým spôsobom bližšie určené a v rámci siete nie sú nijakým spôsobom regulované, hoci budúca podpora pre „oficiálne“ kategórie, ktoré by si určovali noviny sami, je možná. Ku konsenzu ohľadom kategórií teda môže dôjsť len na užívateľskej úrovni v rámci siete. Autori článkov si môžu zadefinovať hocijaké kategórie a priradiť k nim ľubovoľné množstvo článkov.

Kategórie by mali byť čo najjednoduchšie a s čo najväčšou výpovednou hodnotou.

2.3.3 Čitatelia článkov

2.4 Správa užívateľov a peerov

Zamestnanci novín a ich čitatelia musia byť nejakým spôsobom reprezentovaní v sieti. Každý z nich figuruje v peer-to-peer sieti ako peer. Každý z peerov môže mať v rôznych sieťach rôznych novín inú rolu.

Celkovo sú tri roly:

- Čitateľ
- Redaktor
- Šéfredaktor

Všetky roly dokážu to, čo roly, resp. rola pred nimi.

Rola šéfredaktora je spojená najmä s rolou riaditeľa novín. Stará sa o pridávanie a odoberanie redaktorov a protokol, ktorý vytvárame, musí podporovať túto rolu podporovať. V praxi to znamená, že sa zmení stav z čitateľa na redaktora potom, čo to šéfredaktor schváli.

Pri povýšení čitateľa na redaktora je potrebné, aby najprv o to samotný čitateľ požiadal. Takto sa minimalizuje počet peerov pripojených do siete, nad ktorými by musel šéfredaktor pri vytváraní nového šéfredaktora vyhľadávať, čo by bol prípad, kde by šéfredaktor vyberal nových redaktorov zo zoznamu všetkých peerov pripojených do danej siete. Ďalšia alternatíva, kde by povýšil šéfredaktor čitateľa na redaktora priamo vpísaním jeho unikátneho identifikátora je krajne nepraktická. Takto, keď samotný čitateľ požiada o prácu redaktora v daných novinách, sa minimalizuje počet potenciálnych redaktorov, ktorých šéfredaktor musí schváliť, alebo odmietnuť. Pri odmietnutí práce redaktora sa pre žiadajúceho peera nič nemení a naďalej zostáva čitateľom novín, nakoľko sa názor šéfredaktora na celú vec môže v budúcnosti zmeniť.

Spôsob, akým prebieha povýšenie čitateľa na redaktora vytvára isté požiadavky na protokol. Od protokolu sa teda požaduje, aby umožňoval čitateľovi požiadať o prácu redaktora pre dané noviny a následne umožnil šéfredaktorovi na túto požiadavku reagovať kladne, alebo záporne. Protokol následne musí umožniť šíriť redaktorom vytvorené články v rámci daných novín, pričom ich vytváranie musí byť obmedzené len a len na redaktorov daných novín, nakoľko zo zrejmých dôvodov nechceme, aby ktorýkoľvek čitateľ smel pre akékoľvek noviny vytvárať a ďalej šíriť články.

O pravosti článkov vzhľadom k článku samotnému alebo k novinám, resp. jeho autorovi pojednáva sekcia 2.5.

Rola redaktora spočíva vo vytváraní obsahu pre noviny, teda vo vytváraní článkov. Aplikácia musí umožniť článok nahrať z pevného disku alebo ho vytvoriť, resp. upraviť priamo, pričom sa od protokolu vyžaduje, aby podporoval jeho ďalšie šírenie po sieti, či už na žiadosť niektorého z peerov, alebo automaticky. Sekcia 2.5 pojednáva o správe súborov v sieti a ich replikácii.

2.4.1 Založenie novín

Každý peer siete má možnosť založiť si jedny noviny, ktorých sa následne stane šéfredaktorom. Do týchto novín vie následne ľubovoľne prispievať svojimi článkami a sprístupňovať ich ostatným peerom. Peer taktiež začne spravovať zoznamy jednotlivých článkov, či už tých, čo napísal on, alebo tých, čo napísali ostatní redaktori. K správe novín patrí aj správa čitateľov článku, prevádzkovanie STUN serveru a NP2PS prijímateľa.

Obmedzenie na vytvorenie len jedných novín jedným peerom vyplýva z jednoduchšieho návrhu protokolu, takto je totiž možné spojiť dohromady verejný identifikátor šéfredaktora s verejným identifikátorom novín. V praxi si však môže jeden užívateľ vytvoriť niekoľko peerov, jednoducho spustením viacerých inštancií aplikácie.

2.4.2 Technické úlohy

Šéfredaktor má taktiež v protokole istú technickú úlohu. Za účelom *prvého* pripájania sa k novinovej sieti je totiž potrebné, aby aspoň jeden člen siete bol prístupný z WAN siete. Ak tomu tak nie je, nie je možné pripojiť ľubovoľného nového peera do siete. Šéfredaktor má teda úlohu mať nastavený svoj domáci smerovač a bránu firewall tak, aby vôbec bolo možné sa do siete pripojiť, nakoľko je veľmi pravdepodobné, že sa väčšina peerov siete nachádza za smerovačom zo zapnutým systémom NAT.

2.5 Správa súborov v sieti, ich replikácia

2.5.1 Perzistentné ukladanie dát

Pre správny chod siete a protokolu, ktorý ju implementuje je žiadúce, aby do nej boli jednotliví peeri pripojení. Ak to z akéhokoľvek dôvodu nie je možné, musí byť aplikácia schopná perzistentne uložiť svoj stav pri vypínaní a neskôr ho pri opätovnom spustení načítať, aby sa mohol peer opätovne pripájať do siete. Z toho vyplýva povinnosť ukladať nejakým spôsobom nie len samotné články,

ale aj rôzne metadáta o nich spolu s ďalšími metadátami potrebnými pre sieťovú komunikáciu, identifikáciu novín a podobne.

Preto sa od aplikácie vyžaduje, aby nejakým spôsobom perzistentne ukladala články a metadáta na obdobie, kedy nebude spustená. V našej implementácii sme zvolili Google Protocol buffers, kde sme upravili už existujúce Google Protocol buffers definície, ktoré sú používané v našom protokole pri sieťovej komunikácii a pridali sme ďalšie položky a štruktúry tak, aby sa serializovali a perzistentne uložili všetky dáta, ktoré sú potrebné na to, aby mohol byť stav aplikácie obnovený práve tak, aby bolo peerovi umožnené opätovne sa pripojiť do siete. Existujú alternatívne riešenia, napríklad framework „boost“ poskytuje nástroje na serializáciu, rovnako ako aj framework Qt. Tieto však majú svoje nevýhody, príkladom je potreba používať Qt dátové typy (`quint64`, `QString`, `QList` a pod.) ak chceme serializovať pomocou frameworku Qt. Hlavnou nevýhodou „boost“ frameworku bolo najmä pridanie ďalšej závislosti ako pri budovaní programu, tak pri návrhu jednotlivých objektov.

2.5.2 Optimalizácia sieťového prenosu – zdieľanie metadát

Množstvo sieťovej komunikácie chceme využiť čo najoptimálnejšie. Chceme zamedziť situácii, kde by bol obsah prenášaných dát príliš veľký, čo by mohlo viesť k zdĺhavému odosielaniu a prijímaniu pre prijímajúceho peera potenciálne zbytočných dát. Musíme totiž taktiež rátať s tým, že niektorí peeri majú prenos dát po sieti spoplatnený. Rovnako chceme zamedziť situácii, že veľkosť prenášaných dát je príliš malá, pretože to vedie k situácii, že je zbytočne niekoľkokrát inicializovaný prenos, ktorého časté spracovávanie môže byť nákladné na počítače peerov.

V našom prípade je teda vhodné posielať po sieti len metadáta k súborom, ktoré sú nositeľmi obsahovej stránky novín, teda k článkom. Či je potrebné prenášať všetky metadáta, alebo len niektoré, záleží od situácie. Viac o konkrétnych prenášaných údajoch pojednáva sekcia 2.9. Takto sa optimalizuje sieťový prenos, pretože sa zamedzí vo veľkom množstve prenosu informácií, ktoré sú pre prijímateľa zbytočné a pre odosielateľa potenciálnou príťažou. Ideálne by množina posielaných metadát bola minimálna čo do počtu prvkov. Prakticky sa však ukazuje, že ak sa odošle väčšie množstvo metadát, ktoré by pre prijímateľa mohli byť prínosné, tak sa minimalizuje počet prenosov alebo aj počet nových spojení. Takto sme sa dopracovali k problému posielania príliš malého množstva metadát.

2.5.3 Autenticita a pôvod článku

Vzhľadom na to, že sa nachádzame v peer-to-peer prostredí, tak je pri posielaní článku po sieti potrebné nejakým spôsobom overiť jeho autenticitu a pôvod, najmä z toho dôvodu, ak sa čo najviac zamedzilo chcenej, či nechcenej zmene

jeho obsahu a vytvoreniu falošného obsahu článku, alebo len čisto z dôvodu ochrany proti poškodeniu dát na pevnom disku a/alebo počas sieťového prenosu. Článok totiž môže pochádzať od ktoréhokoľvek peera v sieti. V tejto podsekcii sa častejšie spomínajú viaceré pojmy a závery z kryptografie, ktoré sú dôkladnejšie rozoberané a analyzované v kapitole 3.

Autenticita článku voči jeho obsahu

Článok zdieľaný do siete a uložený na disku niektorého z peerov môže byť lokálne pozmenený, buď to peerom samotným, alebo poškodením dát na disku. Po prijatí článku zo siete od niektorého z peerov je teda potrebné zistiť, že nebol pozmenený jeho obsah oproti jeho originálu.

Aby sme optimalizovali sieťový prenos, posielame pri sieťovej komunikácii len metadáta článku. Nakoľko článok prijatý po sieti nemáme s čím priamo porovnať, musíme ho nejakým spôsobom unikátne reprezentovať v jeho metadátach tak, aby sme akúkoľvek prijatú verziu článku vedeli nejakým spôsobom spracovať a s touto reprezentáciou ju porovnať. Keďže chceme sieťový prenos optimalizovať, chceme zároveň unikátna reprezentácia článku bola čo do veľkosti menšia. Pri kladení takýchto podmienok však môže nastať situácia, že dva články budú mať rovnakú reprezentáciu. Možným riešením sú kryptografické hashe, ktoré sa na rovnaký účel používajú v mnohých aplikáciách, napríklad pri ukladaní hesiel užívateľov, kde je z bezpečnostných dôvodov nežiadúce, aby boli ukladané v originále. My spočítame kryptografický hash z obsahu článku a následne ho uložíme do metadát. Pri prijímaní článku zo siete tak spočítame kryptografický hash z prijatého článku a porovnáme s tým hashom, ktorý máme uložený. Takto sme zabezpečili, že článok nebol lokálne na odosielajúcom peerovi pozmenený.

Pôvod článku voči daným novinám

Ktorýkoľvek peer v sieti by mohol svoj vlastný článok vydať ako článok hociktorých novín a preto je žiadúce, aby peeri oprávnení vydávať články, teda redaktori a peeri, mohli článok nejakým spôsobom označiť „značkou“, že sa skutočne jedná o článok pochádzajúci z daných novín. Takáto „značka“ nesmie byť ľahko kopírovateľná, aby ju nebolo možné sfalšovať. Musí byť teda unikátna pre danú kombináciu novín a článku, pričom ale musíme byť schopní nejakým spôsobom overiť pravosť tejto značky, teda že sa skutočne jedná o identifikáciu článku, ktorý náleží daným novinám. Na tento účel sa digitálne podpíše daný článok, alebo jeho kryptografický hash a priloží sa k správe, ktorá sa posiela po sieti. Takýto digitálny podpis je možné si overiť priamo a lokálne, bez potreby pripájania sa k šéfredaktorovi, alebo inému peerovi. Aby na overenie platnosti článku nebolo potrebné podpisovať celý článok, postačí podpísať jeho kryptografický hash. O

kryptografických hashoch sa viac pojednáva v sekcii 3.5.

Na účel digitálneho podpisu je potrebné, aby každé noviny mali vlastnú dvojicu kľúčov, verejného a privátneho. Na vytvorenie podpisu je potrebný privátny kľúč, ktorý musí byť šírený medzi redaktormi novín. Na overenie digitálneho podpisu postačí kľúč verejný a tento je zdieľaný medzi všetkých peerov, ktorí sú pripojení do siete daných novín. Hoci sa jedná o privátny kľúč a teda sa neráta s jeho zdieľaním po sieti, je to pre správne fungovanie siete potrebné. Takto sa totiž zabezpečí to, že ak pribudne do siete nový redaktor, tak o tom nemusí byť každý peer uvedomený, prostredníctvom našej siete, samostatne, ale redaktor sa overí sám, keďže článok podpíše spoločným privátnym kľúčom. Viac o asymetrickom šifrovaní a podpisoch pojednáva sekcia 3.3.

Pôvod článku voči jeho autorovi

Vzhľadom na to, že rôzni peeri zdieľajú rôzne články, ktorých autenticitu a pôvod voči novinám už vieme overiť, nie je overovanie ich pôvodu u autora až tak podstatné. To je najmä z toho dôvodu, že sa na noviny pozeráme ako na celok a inštitúciu a tak sa pozeráme aj na vydané články a teda nás nezaujíma kto ich napísal, ale ktoré noviny ich vydali. Ďalším dôvodom je predpoklad, že žiaden redaktor sa nebude vydávať za iného.

Prakticky však, ak článok odošle do siete inému peerovi jeho autor, je pôvod článku overený tým, že sa zašifrovaná správa dešifruje symetrickým kľúčom určeným danej dvojici peerov. Jeden z dvojice teda nutne musí byť autor článku a tým je len jeho odosielateľ. Toto platí len pri priamej komunikácii s autorom a to aj prostredníctvom presmerovania, o ktoré sa stará časť aplikácie s protokolom STUN a jeho nadstavbou, ktorá sú čo do najväčšej miery nezávislé na NP2PS časti.

2.5.4 Pridanie článku do siete, jeho aktualizácia, zdieľanie a replikácia v sieti

Zdieľanie článkov

Zdieľanie článkov medzi peermi patrí medzi základné vlastnosti nami vytváraného protokolu. Chceme totiž, aby si jednotliví peeri vzájomne a decentralizovane medzi sebou vymieňali informácie, teda aby bol peer A schopný vyžiadať nejaký súbor, v našom prípade článok, od peera B, pričom peerom B rozumieme akéhokoľvek peera v sieti, ktorý nie je peerom A.

Články nie sú jedinou entitou, ktorú, resp. ktorej metadáta si medzi sebou môžu peeri vymieňať. Čo sa týka konzumácie obsahu novín, je však článok tou najčastejšie používanou a teda aj zdieľanou entitou a tak jednotlivé nároky na protokol kladieme najmä s ohľadom na ich zdieľanie.

Peer môže vyžiadať článok od ktoréhokoľvek iného peera, pričom ten mu ho poskytne, ak ho sám má lokálne k dispozícii. Ak ho k dispozícii nemá, musí žiadajúci peer požiadať iného peera. Aby nedochádzalo k zbytočnej sieťovej komunikácii s peermi, ktorí článok nemajú, ale taktiež, aby sa zamedzilo priveľkému zaťaženiu niektorých peerov, pokúsime sa ich komunikáciu minimalizovať vytvorením kritérií a poradia, v ktorom sa budú jednotliví peeri kontaktovať so žiadosťou o článok.

Článok majú lokálne uložený jeho autor a šéfredaktor novín. Títo budú kontaktovaní, ak žiaden z čitateľov článok nemá k dispozícii. Takto rozložíme sieťovú komunikáciu medzi ostatných peerov, aby sme príliš nezaťažovali autora článku, redaktorov a šéfredaktora. Ďalej, ak by kontaktovanie čitateľov nebolo úspešné, sú kontaktovaní redaktori novín, autor článku a nakoniec šéfredaktor. Začíname síce na peeroch, u ktorých je šanca na nález najnižšia a postupne sa presúvame k peerov, u ktorých šanca na nález rastie.

Vzhľadom na to, že sme si zaviedli zoznam čitateľov článku, kontaktujeme len tých peerov, ktorí, podľa tohto zoznamu, článok majú. Tento zoznam nemusí byť nutne aktuálny ani presný, nakoľko prinajhoršom dostaneme, na našu žiadosť o článok, zápornú odpoveď. Stále je však takáto „zbytočná“ komunikácia užitočná, pretože postupne kontaktujeme ďalších a ďalších peerov. Pri veľkých novinách môže zoznam čitateľov narásť na veľký počet peerov a preto, aby sme nezahltili samých seba v krátkom časovom intervale veľkým množstvom komunikácie, bude aplikácia posielať žiadosti po častiach a buď to čakať na ich odpoveď, alebo daného peera preskočiť, ak ho nie je možné kontaktovať. Takto postupne kontaktujeme zaradom jednotlivé skupiny peerov, pokiaľ nedostaneme kladnú odpoveď s obsahom článku.

Spomínali sme, že každý peer môže slúžiť ako STUN server, teda je možné ho využiť na presmerovanie komunikácie na komunikovanie s peerom, ku ktorému sa nie je možné zo siete WAN pripojiť, či už kvôli prekladu sieťových adries, alebo bráne firewall, ktorá by blokovala všetku príchodziu komunikáciu. Na týchto peerov sú kladené vyššie nároky na komunikáciu a preto je vhodné pri získavaní článkov uprednostniť tých peerov, s ktorými máme nadviazané priame spojenie. Ušetríme tak peerov, ktorí slúžia ako STUN servery a v konečnom dôsledku môže byť priama komunikácia aj rýchlejšia.

Pridanie článku a jeho následná replikácia

Keďže chceme, aby jednotliví redaktori boli schopní písať pre noviny články a aj ich následne dávať k dispozícii čitateľom na čítanie, musí protokol podporovať pridávanie nových článkov do siete.

Pridanie, alebo publikácia nového článku do siete znamená uloženie jeho metadát do peerovej aplikácie a následná replikácia týchto dát ostatným peerom v

sieti. Dôsledkom je možnosť dotazovať sa na novovytvorený článok od ostatných peerov.

Keďže sa nachádzame v peer-to-peer prostredí, kde jednotliví peeri pripájajú do siete zo svojich osobných počítačov ktoré nie sú nepretržite zapnuté a pripojené do siete, je potrebné, aby sa novovytvorený článok do siete replikoval tak, aby ďalší peeri mohli pristúpiť k jeho obsahu aj v prípade, že oň majú záujem aj mimo doby, kedy je pripojený jeho autor. Replikáciou sa totiž článok prenesie aj na ďalších peerov siete a vzrastie tak jeho dostupnosť.

Jednoduchou formou replikácie je stiahnutie článku od jeho autora iným peerom. Takto sa replikuje článok na daného peera a v tej chvíli zostane článok k dispozícii aj v tom prípade, že sa autor odpojí. Samozrejme, takáto situácia nemusí nastať ak sa nenájde peer, ktorý by si článok stiahol. Je teda potrebné, aby replikácia prebiehala automaticky. Ďalej, protokol nemá žiadne nároky na počet pripojených peerov v sieti. Z toho dôvodu teda vyplýva, že distribuované systémy typu DHT (distributed hash table) nebudú spoľahlivo funkčné, nakoľko spravidla vyžadujú aspoň istý počet peerov, ktorí sú stále pripojení v sieti. Ak by sme aj vytvorili DHT bez takéhoto obmedzenia, s rastúcim počtom článkov a neustálym pripájaním a odpájaním sa peerov by dochádzalo k neustálemu prelievaniu dát medzi nimi. Tento proces trvá istú dobu a protokol rovnako nedisponuje super-peerom, či serverom, nakoľko chceme čo najviac zvýšiť mieru decentralizácie a vyhnúť sa centralizačným prvkom. Ak by tomu tak nebolo, bolo by možné použiť takéhoto super-peera alebo server na automatické ukladanie dát.

Náš postup replikácie teda požaduje, aby boli replikované dáta statické, čo znamená, že sa po ich replikácii na cieľového peera už z neho ďalej nepresúvajú; inými slovami, nechceme, aby sa pri odpájaní peera dáta z neho presúvali na ostatných peerov a po pripojení zas naspäť, ako by tomu bolo v prípade DHT. Množstvo dát, ktoré sa replikuje na jedného peera nie je závislé od počtu pripojených peerov v sieti. S týmito vlastnosťami nie je potrebné vedieť presný stav siete, ani neustále presúvať dáta medzi peermi, čím sa optimalizuje množstvo sieťovej komunikácie a teda minimalizuje sieťová komunikácia, ktorú peer samotný neinicioval. Z toho nám plynie, že skupina peerov, kam sa replikujú dáta musí byť vopred daná. Ideálne chceme takých peerov, ktorý budú spoľahliví, dôveryhodní a ochotní obsah nie len konzumovať, ale aj zdieľať. Vhodnou skupinou budú šéfredaktor a redaktori daných novín, nakoľko môžme predpokladať, že keďže sa jedná o ich vlastné noviny budú spĺňať požadované kritériá. Takýmto spôsobom sa vlastne decentralizuje centrálne úložisko dát novín, ktorého úlohu by najskôr tvoril samotný šéfredaktor. Prístup k obsahu novín sa takto rozprestrie medzi všetkých redaktorov a to v rovnakej miere. Replikujú sa vždy novovytvorené a aktualizované články a to priamou komunikáciou medzi autorom článku a ostatnými redaktormi, resp. šéfredaktorom. Aby sa zamedzilo hromadeniu starých článkov na pevných diskoch redaktorov je vhodné, aby aplikácia umožňovala

takéto staré články automaticky mazať.

Takýmto spôsobom sme dosiahli decentralizáciu údajov, avšak do rigidnej, sporadicky sa meniacej (vzhľadom k prísunu článkov) skupiny peerov. Takáto množina stále tvorí istú zraniteľnosť siete. Preto je vhodné ju rozšíriť o ďalších peerov. Nutne sa nemusí jednať o automatickú replikáciu ako tomu je medzi jednotlivými redaktormi a šéfredaktorom, postačí poloautomatická replikácia na strane aplikácie, ktorá si sama v pravidelných intervaloch vyžiada aktualizáciu obsahu novín a taktiež obsiahnutie nových článkov. Aplikácia musí používať vlastný časovač, ktorý sa spustí spolu s jej spustením. Takto sa predíde tomu, aby všetci peeri naraz vyžiadali aktualizáciu obsahu novín.

Vytváranie metadát článku Metadáta sú vyplnené adekvátne k peerovi, ktorý článok pridáva. Ako autor je nastavený teda peer sám. Aplikácia umožní pridávať články peerom len do tých novín, ktorých je redaktorom. Informácie o týchto novinách sa následne prepíšu do metadát. Nadpis prvej úrovne v súbore sa nastaví ako nadpis článku a spočítajú sa hashe, ako kryptografický, tak identifikačný. Verzia sa nastaví na hodnotu 1 a čas vytvorenia a poslednej modifikácie sa nastaví na ten čas, kedy je článok pridaný do aplikácia a nie kedy bol jeho pôvodný súbor vytvorený. Redaktor článok podpíše súkromným kľúčom novín. Nastaví sa cesta k článku a formát súboru článku, v našom prípade Markdown. Kategórie sú nastavené také, aké zadá redaktor pri publikácii článku, čo sú zároveň jediné metadáta vypĺňané priamo. Všetky ostatné sú vyplnené nepriamo z už zadaných informácií.

Aktualizácia článku

Informácie napísané v článku môžu byť z rôznych dôvodov, napríklad kvôli vývoju udalosti, ktorú popisujú s postupom času nepravdivé a/alebo neúplné. Výhodou elektronických novín oproti printovým je možnosť obsah článku upraviť bez toho, aby bolo potrebné vydať celé vydanie novín. Autor článku teda môže obsah článku upraviť a vydať jeho aktualizovanú verziu. Pravidlá pre replikáciu takýchto článkov sú rovnaké, ako pri vytváraní nových článkov.

Pri aktualizácii článku je možné zmeniť obsah článku akokoľvek a teda sa menia aj jeho metadáta. Zmení sa samozrejme aj jeho kryptografický hash. V konečnom dôsledku sa jedná o opätovné vydanie toho istého článku s úpravami. V metadátach je však záznam o predošlej verzii článku a časy vytvorenia sa zhodujú s pôvodným článkom, líšia sa len časy poslednej úpravy. Číslo verzie je taktiež o jedno vyššie ako predošlá verzia. Prijatie novej verzie článku je uložené samostatne a starú verziu neprepíše.

Úprava metadát článku Po aktualizácii článku, alebo po stiahnutí novej verzie článku sa väčšina metadát okopíruje z predošlej verzie, ale niektoré metadáta sa zmenia. Uloží sa novinový identifikátor článku predchádzajúcej verzie a zvýši sa číslo verzie o 1. Časy vytvorenia sa stále zhodujú, avšak čas poslednej úpravy je iný. Samozrejme sa líšia aj hashe, keďže sú vytvárané z obsahu článku, ktorý je v tomto prípade iný.

2.6 Decentralizácia

Pre náš účel nespornou výhodou je relatívne jednoduchá decentralizácia, nakoľko navrhujeme protokol pracujúci na princípe peer-to-peer. Jednotliví peeri sú si rovnocenní, teda dáta nie len sťahujú zo siete, ale ich aj do siete poskytujú. Dôvodom pre decentralizáciu je odstránenie centrálneho prvku. V našom prípade nám ide o odstránenie centrálneho úložiska a delegáciu istých centrálnych právomocí na nižie postavených peerov siete.

Na odstránenie centrálneho úložiska používame metódu replikácie dát, v našom prípade článkov. Články sú replikované vzájomne medzi jednotlivými redaktormi a šéfredaktorom daných novín a teda nie sú na jednom centrálnom mieste. Rovnako sú distribuované medzi čitateľmi daných novín, ako manuálne na základe toho, čo peer zo siete stiahol tak aj poloautomaticky, čo aplikácia sama sťahuje pri synchronizácii stavu novín.

Za účelom delegácie centrálnych právomocí rozumieme to, že šéfredaktor prišiel o niekoľko svojich právomocí a tie boli delegované na rádových redaktorov. Jedná sa najmä o schvaľovanie vydávania nových článkov, kde ak by táto právomoc zostala len šéfredaktorovi, tak by pridávanie článkov bolo možné len počas jeho prítomnosti v sieti, vytvárajúc centrálny prvok. A keďže publikácia nových článkov je v novinách dôležitá, chceme zabrániť aby bol ľahko zraniteľná.

Právomoc, ktorú by bolo náročné delegovať, ale ktorá zároveň nie je tak častá a sieť bez jej použitia stále bude plne funkčná, je pridávanie nových redaktorov. Alternatívou by bolo vytvorenie decentralizovaného, virtuálneho šéfredaktora, avšak vzhľadom na delegovanie čo najväčšieho množstva právomocí na redaktorov sa kladie väčší dôraz na ich znalosti a všeobecnú kvalitu ako redaktorov pri výbere. Ďalej, keďže je šéfredaktor novín zároveň ich riaditeľom, je vhodné zachovať jeho autoritu pri výbere ďalších redaktorov.

2.7 Sieťová komunikácia peerov

2.7.1 Prvotné pripojenie do siete

Pri pripájaní sa do NP2PS siete je potrebný tzv. „bootstrapping“, teda proces, pri ktorom sa peer pripojí na iného peera v sieti, ktorý mu poskytne údaje o sieti potrebné na pripojenie samotné. Predovšetkým sa jedná o transportné adresy ďalších peerov. V tomto prípade je potrebné, aby sa peer pripájal k peerom, ktorí majú o adresách ďalších peerov dostatočné informácie. Komunikácia v sieti je orientovaná najmä na články a teda prebieha primárne s redaktormi novín.

Z toho dôvodu bude pripájanie prebiehať práve k redaktorom novín. Aby sa dosiahlo konzistentného výsledku ohľadom informácií o peeroch v sieti, musia si ich transportné adresy medzi sebou aplikácie redaktorov pravidelne vymieňať. Jedná sa o proces o troch stavoch, kde najprv aplikácia jedeného redaktora vyžiada zoznam peerov od tej druhej vo forme verejných identifikátorov peerov. Následne sú vyžiadané ďalším zoznamom verejných identifikátorov úplné informácie o tých peeroch, o ktorých tieto informácie aplikácia ešte nemá.

Po takomto pripojení si už peer môže vyžiadať zoznam článkov a komunikovať v sieti ako každý ďalší peer, rovnako môže synchronizovať svoj zoznam novín so zoznamom novín ostatných peerov. Na čitateľov novín sa celý zoznam peerov neprenáša, avšak používa sa metóda tzv. „gossipingu“, našepkávania, kde si peeri vzájomne náhodne vymieňajú informácie o ďalších peeroch. Bootstrapping prostredníctvom peerov, ktorí nie sú redaktormi daných novín je možný, avšak s ním súvisí niekoľko problémov.

Ako peer nájde prvého peera, aby mohol previesť bootstrapping? Spôsobov je niekoľko. Peer môže ručne zadať jeho transportnú adresu a tak sa priamo pripojiť na niektorého z redaktorov, prípade nahradiť transportnú adresu doménovým menom. Takéto riešenie však nie je ideálne pre bežného užívateľa a bežnú prevádzku, hoci samotný bootstrapping nie je používaný často, len pri prvotnom pripájaní. Práve preto je táto možnosť akceptovateľným prístupom. Aby sa však pripájanie a bootstrapping ešte viac zjednodušilo, tak je možné uložiť informácie o novinách, ako ich metadáta a redaktorov do samostatného súboru a tento súbor neskôr použiť pri pripájaní sa do siete. Tento súbor môže byť na stiahnutie prostredníctvom iných protokolov, ako HTTP, SFTP a podobne. Rovnako sa môže šíriť na rôznych internetových fórach, alebo na fyzických prenositeľných médiách.

Metadáta novín

Noviny samotné majú svoje metadáta, ktoré je možné použiť ako prostriedok na pripájanie sa do ich siete, pretože obsahujú všetky na to potrebné informácie. Pôvodne vznikli metadáta novín ako obraz dátových položiek OOP objektu, ktorý

sčasti tvorí našu aplikáciu. Aby však bolo možné metadáta novín používať na pripájanie sa do ich siete, je potrebné, aby spĺňali isté podmienky a boli mierne upravené. Na pripojenie potrebujeme zoznam redaktorov s ich transportnými adresami, verejný identifikátor novín, ktorý je zároveň zhodný s verejným identifikátorom šéfredaktora. Alternatívne, pri výmene týchto informácií medzi peermi v rámci siete je možné pribaliť aj zoznam článkov, čím sa ušetrí sieťový prenos a zároveň dostane peer menší prehľad o obsahu zdieľaných novín. V metadátach sa nachádzajú ďalšie dodatočné informácie ako verejný kľúč novín na overovanie podpisov článkov alebo názov novín, pre účel aplikácie.

Pri žiadaní zoznamu článkov sa v danej sprievodnej správe nachádza položka s dátumom a časom poslednej aktualizácie obsahu novín. To je z toho dôvodu, aby sa informácie od rôznych peerov mohli porovnávať a aby pri žiadosti o zoznam článkov nebolo potrebné posielať všetky články ale len tie, ktoré sú novšie ako dátum a čas poslednej aktualizácie. Takéto opatrenie je najmä z dôvodu optimalizácie sieťového prenosu.

2.7.2 Ďalšia komunikácia peerov

Potom čo peer prejde bootstrappingom a je pripojený do siete môže komunikovať s ostatnými peermi. Peeri medzi sebou komunikujú použitím správ, ktoré sú vo formáte Google Protocol buffers, o správach pojednáva samostatná sekcia 2.9. Každá správa má inú počiatočnú cieľovú skupinu peerov, aj veľkosti jedna, ktorej sa správa odošle. Následne, buď to podľa odpovede peerov v prvej skupine, alebo aj automaticky, sa správa pošle ďalším skupinám peerov, ktoré ale nemusia existovať. Jedná sa o virtuálne skupiny, ktoré sú „vytvárané“ ad-hoc. Toto je zavedené opäť z dôvodu optimalizácie sieťového prenosu.

2.8 Definície a terminológia

Nasleduje niekoľko definícií, ktoré sa objavujú v texte. Pri protokole STUN boli definície ich pojmov prevzaté z adekvátneho RFC [2].

- NP2PS peer, alebo aj len „peer“: Základná entita celej siete. Použitím NP2PS klienta a NP2PS servera komunikuje s ostatnými peermi. Môže sa ľubovoľne pripájať k ostatným peerom a prijímať pripojenia od ostatných peerov. Každí dvaja peeri sú si navzájom, v rámci technickej časti protokolu, rovní. Jeden peer môže patriť do viacerých sietí.

- NP2PS klient, alebo aj len „klient“: Tá časť peera, ktorá pokladá požiadavky a začína komunikáciu z druhým peerom.

- NP2PS server, alebo aj len „server“: Tá časť peera, ktorá reaguje na požiadavky zaslané od klienta, sám od seba spravidla spojenie nenadväzuje.

- NP2PS sieť, alebo aj len „sieť“: Zoskupenie peerov. Jedna sieť spravidla znamená jedny noviny. Za administrátora siete je možné považovať šéfredaktora novín, pre ktoré je daná sieť vytvorená.

- Správa: Súčasť protokolu, slúži na komunikáciu po sieti medzi peermi. Posiela sa zašifrovaná symetrickým kľúčom, alebo podpísaná a zašifrovaná asymetrickým kľúčom.

- Metaspráva: Správa rozšírená o identifikačný bajt, ale aj o ďalšie informácie ako je dĺžka správy. Metaspráva je to, čo sa nakoniec pošle po sieti.

- Markdown: Jednoduchý značkovací jazyk. Jedná sa o jedinú vyžadovanú formu prenosu článkov po sieti. Je relatívne jednoducho čitateľný pre ako ľudí, tak aj stroje.

- Wire protokol: Protokol popisujúci spôsob ako prenášať dáta z jedného miesta na druhé. Je potrebný v prípade, že niekoľko, aspoň dve, aplikácie potrebujú navzájom spolupracovať a komunikovať.

- MSb: Most significant bit. Bit v bajte, ktorý má najväčšiu hodnotu, teda $2^7 = 128$.

- Varint: Celé číslo s premenlivou dĺžkou, záporné hodnoty sú v dvojkovom doplnku, alebo pomocou princípu zig-zag.

- `LF`: Line feed. Znak konca riadku bežne používaný v tzv. „Unix-like“ operačných systémoch.

- `CR LF`: Carriage return, line feed. Dvoj znak konca riadku používaný najmä v operačnom systéme Microsoft Windows.

2.9 Správy

Správy slúžia ako hlavný prostriedok na zaobalenie dát, a sú v konečnom dôsledku posielané medzi peermi v sieti. Obsah správ je spravidla obrazom metadát entít protokolu, najmä článkov a novín.

2.9.1 Google Protocol buffers

Správy sú serializované použitím Google Protocol buffers, čo je formát na výmenu dát. Štruktúra, do ktorej sa bude správa z jej aplikačného objektu prepisovať, je definovaná v samostatnom súbore a následne, pomocou externého prekladača, je vygenerované API pre potrebný programovací jazyk, v ktorom je aplikácia naprogramovaná. Súbor zo štruktúrou už neskôr nie je potrebný.

Google Protocol buffers podporuje základné typy ako napríklad čísla, textové reťazce či boolean ale aj zložitejšie konštrukty ako napríklad mapy. Jednotlivé položky v správach môžu buď to byť povinné pri každej správe alebo naopak, nemusia tam byť vôbec. Položky môžu byť podľa potreby aj viacnásobné. Podpora je taktiež pre enumeračné typy. Jednotlivé štruktúry je možné vkladať navzájom do seba.

Vzhľadom na to, že Google Protocol buffers je len nástrojom na serializáciu štruktúrovaných dát, mohli by sme ho pokojne zameniť za iné formáty, alebo nástroje. Dnes často používanými sú formáty XML, JSON alebo YAML. Tieto sú však len formátmi, to znamená, že na prácu s nimi sú potrebné externé knižnice. To vytvára pri niektorých programovacích prostrediach závislosti práve tak, ako Google Protocol buffers. Oproti spomínaným formátom sú Google Protocol buffers správy v konečnom dôsledku kratšie, čo ušetrí sieťovú komunikáciu. Nevýhodou je samozrejme neschopnosť správy čítať bez programu, ktorý pozná ich štruktúru. Textové formáty ako JSON či XML sú čitateľné aj „voľným okom“, alebo použitím jednoduchého parsovacieho programu. To však v našom prípade nie je potrebné, všetky správy sú totiž aplikáciou spracované do interných metadát.

Ďalšou výhodou Google Protocol buffers oproti XML, či JSON je takzvaná silná schéma. Táto zabezpečuje, že majú dáta potrebnú štruktúru, s čím súvisí kompatibilita medzi jednotlivými úpravami serializovaných dát. Taktiež to umožňuje generovať zdrojové kódy pre rôzne programovacie jazyky. Samozrejme, je možné overiť, či dané dáta podliehajú danej štruktúre, alebo nie. Všetky tieto úkony by sme v prípade spomínaných textových súborov museli programovať ručne, alebo použiť ďalšiu externú knižnicu, čím ale dosiahneme výsledok najviac tak dobrý, ako s Google Protocol buffers.

Alternatívami k Google Protocol buffers sú napríklad FlatBuffers alebo Cap'n Proto. Oboje tieto technológie sú však menšie a menej používané, ako nami zvolený Google Protocol buffers, v prípade FlatBuffers to napríklad znamená že niektoré vlastnosti nie sú k dispozícii vo všetkých programovacích jazykoch. Taktiež sa jedná o technológiu vyvíjanú spoločnosťou Google, ktorá sa môže rozhodnúť podporu pre FlatBuffers zrušiť a celý projekt zatvoriť. V prípade Google Protocol buffers sa taktiež nájdu vlastnosti, ktoré nie sú implementované pre všetky programovacie jazyky avšak po už značne dlhšom vývoji môžeme očakávať, že ich bude menej. Ďalej má Google vlastný nástroj na tzv. „Remote

procedure call“, skrátene RPC, ktorý používa práve tradičné Google Protocol buffers a nie FlatBuffers. Cap’n Proto je menší nástroj, s ktorého používaním súvisia práve problémy s malými nástrojmi spojené, medzi čo patrí napríklad ťažšie vyhľadávanie informácií ohľadom málo používaných zákutí tohto nástroja. Ďalej Cap’n Proto sám o sebe obsahuje podporu pre RPC, ktorá nie je v našom prípade vôbec potrebná. Cap’n Proto správy taktiež nie sú nijakým spôsobom kódované a teda sú dlhšie, kde Google Protocol buffers používa tzv. „wire“ formát.

Alternatívne nástroje nám teda prišli buď to ešte stále vo vývoji, ohrozené svojimi robustnejšími náprotivkami alebo vhodnejšie na iný účel.

2.9.2 Hlavička správy

Správa v sieti ide od peerovi k peerovi a odosielateľ potrebuje vedieť prijímateľa, teda komu správu poslať. Na druhú stranu, prijímateľ často potrebuje vedieť, kto mu správu poslal. Tieto informácie potrebuje každá správa, tak teda zavedieme tzv. hlavičku správy, kde ich obsiahneme. Ako z názvu vyplýva, hlavičku správy tvoria jej všeobecné metadáta, ktorých prítomnosť v správe nezávisí na dátach, ktoré nesie. Prijímateľa aj odosielateľa zaznamenávame pomocou ich verejného identifikátora, keďže sa jedná o číslo, ktoré má každý z nich jedinečné.

Aby prijímateľ prijímajúci správu vedel, o akú správu sa jedná je potrebné zaviesť typ správy a kontext správy.

Kontext správ

Kontext správy slúži na to, aby jej prijímateľ vedel, ako s danou správou naložiť, teda ako ju správne spracovať. Príkladom je napríklad situácia, kedy prijme peer správu, ktorá je typu Article All. V tejto chvíli peer nevie, či sa jedná o žiadosť o nejaký článok od niektorého z ostatných peerov, alebo je to odpoveď na článok, o ktorý požiadal iného peera a od ktorého zatiaľ nedostal odpoveď. Na kontext nám poslúžia dva typy, požiadavka (Request) a odpoveď (Response). Ako z neskoršieho modelovania správ vyplynie, takáto kombinácia postačí na drvivú väčšinu typov správ, ak ešte pridáme aj tretí kontext chyby (Error). Potrebujeme totiž rozlíšiť odpoveď úspešnú a neúspešnú. Ak by nebolo možné komunikáciu peerov pre daný typ dostatočne obsiahnuť do týchto dvoch štádií kontextov, pridáme štvrtý kontext, jednosmerný (One Way). Tento kontext je určený pre správy, ktoré vyžadujú zložitejšiu komunikáciu medzi peermi a ich stav sa rieši na úrovni obsahu správ samotných, alebo pre jednoduché správy, ktoré nepotrebujú odpoveď, alebo predošlú požiadavku. Tieto komplikovanejšie kontexty by bolo možné zaviesť pre všetky správy, ale nie je to potrebné a keďže chceme zachovať istú mieru prehľadnosti v protokole, najmä keď sa to týka takmer každej správy, je vhodné ponechať tieto kontexty na správy samotné.

Typy správ

Odosielateľ vie, akú správu do siete posiela, ale prijímateľ už nie. V momente, kedy správu celú prečíta a prijme, vie s istotou len to, že bude daná správa obsahovať hlavičku. Získanie ďalších dátových položiek by mohlo byť riešené metódou pokus-omyl, kde by sa prijímateľ pokúsil deserializovať jednotlivé dátové položky a na základe toho zistil o akú správu sa jedná. Táto metóda je však neefektívna a môže viesť k viac-zmyselným správam. Potrebovali by sme teda nejakým spôsobom zaznamenať, o aký typ správy sa jedná a na základe toho môžme určiť, aké dátové položky obsahuje. Túto informáciu môžeme umiestniť do hlavičky, keďže potrebujeme, aby bola prítomná v každej správe.

Typ správy v podstate určuje, akej entity sa správa týka a ktorej metadáta bude obsahovať. Prijímateľ na základe toho môže interpretovať prijaté dáta správnym spôsobom a adekvátne s nimi naložiť. Typy správ sumarizuje tabuľka 2.1. Pri používaní kontextu iného než One Way sa peerovi vytvárajúcemu Request a prijímajúcemu Response hovorí „žiadateľ“, alebo „odosielateľ“. Peerovi, ktorý reaguje na Request vytvorením správy v kontexte Response povieme „prijímateľ“.

ID	Typ	Účel
0	Empty	Správa obsahujúca len hlavičku, bez ďalšieho rozšírenia.
10	Article All	Správa nesúca celý článok, teda jeho hlavičku aj jeho samotný obsah.
20	Article Header	Správa nesúca len hlavičku článku, bez jeho obsahu.
21	Article Solicitation	Správa oznamujúca potenciálneho vlastníka hľadaného článku.
25	Article List	Správa obsahujúca zoznam správ.
40	Article Data Update	Správa nesúca informáciu o tom, že si nejaký peer stiahol niektorý článok.
70	Update Margin	Správa slúžiaca na aktualizáciu marginálií.
100	Credentials	Správa na vyžiadanie kľúčov, identifikátorov, IP adries a podobne.
110	Symmetric Key	Správa na výmenu symetrických kľúčov.
120	Public Key	Správa na výmenu verejných kľúčov peerov.

Tabuľka 2.1 Zhrnutie typov správ, ich použiteľnosti a stručného významu.

2.9.3 Správy o článkoch

Základnou úlohou protokolu je zdieľanie článkov v sieti medzi peermi. Táto podsekcia popisuje skupinu správ, ktoré sú určené na prácu s článkami, teda ich získavanie, pridávanie a vyhľadávanie.

ArticleAll

Aby bolo možné články v sieti zdieľať, je potrebné mať správu, ktorá bude samotným nositeľom článku a zároveň poslúži na jeho vyžiadanie. Správa teda musí obsahovať novinový identifikátor článku, aby bolo možné článok jednoducho identifikovať po jej prijatí.

V kontexte Response je správa obohatená o ďalšie položky. Predovšetkým sa jedná o vnorenú správu, ktorá obsahuje hlavičku článku – Article. Tá obsahuje všetky metadáta článku. Hlavička samotná sa však používa vo viacerých správach, napríklad v zozname článkov, preto je vhodné, aj zo sémantického hľadiska, ju od obsahu článku oddeliť. Dôsledkom toho je fakt, že článok samotný sa posiela v správe samostatne, ako textový reťazec. Tu je vhodné podotknúť, že do budúcna by bolo zaujímavé použiť bezstratovú kompresiu textu, čím by sa zmenšil obsah správy.

Spracovanie Request Po prijatí požiadavky o článok v jeho plnom znení musí peer, ak má daný článok lokálne uložený, vytvoriť správu typu Article All v kontexte Response. O aký článok sa jedná zistí z obsahu správy. Prijímajúci odpovie kladne len v prípade, ak má požadovanú verziu článku, teda neposiela sa stará verzia. Ak peer článok nemá, je možné tento fakt oznámiť žiadateľovi článku odoslaním správy v kontexte Error.

Spracovanie Response V prípade úspešnej odpovede žiadateľ zo správy uloží jeho obsah na lokálny disk a ak hlavičku daného článku nemal k dispozícii predtým, vytvorí aj tú. Je potrebné, aby žiadateľ skontroloval jeho kryptografický

ID	Kontext	Účel
0	Error	Chybová správa.
10	Request	Požiadavka na iného peera.
20	Response	Odpoveď na požiadavku. Neposiela sa samostatne, bez predošlej požiadavky.
30	One Way	Jednosmerná správa, nečaká sa odpoveď.

Tabuľka 2.2 Zhrnutie možného kontextu správ.

hash a taktiež overil, či sa nejedná o novšiu verziu už existujúceho článku. V tom prípade sa musí založiť nový článok a správne nastaviť metadáta tak, aby ukazovali na jeho predchodcu.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Novinový identifikátor článku, na identifikáciu článku v rámci novín.
- Položky, prítomné len pri odpovedi (Response kontext):
  - Hlavička článku. Obsahuje hlavičku článku, ktorý je posielaný.
  - Článok samotný. Textová časť článku je posielaná ako textový reťazec. V prípade potreby je možné ju posielať ako reťazec bajtov.

Article

Zo sémantických dôvodov je vhodné hlavičku článku oddeliť do samostatnej správy, keďže sa používa v ďalších správach: ArticleAll a ArticleList. Rovnako je používaná pri serializácii stavu aplikácie pri jej vypínaní. Správa nemá položky rozdelené podľa kontextu a všetky zrkadlia metadáta článku.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Meno autora.
- Verejný identifikátor autora.
- Názov novín.
- Verejný identifikátor novín.
- Novinový identifikátor článku; identifikačný hash.
- Nadpis článku.
- Kategórie článku.
- Marginálie článku.
- Typ, resp. formát článku.
- Kryptografický hash článku.
- Dátum a čas vytvorenia.
- Dátum a čas poslednej úpravy.
- Verzia.
- Predošlá verzia, ak existuje.

ArticleList

Aby peer so záujmom o články nemusel prechádzať jeden článok po druhom a všetky lokálne sťahovať, a prípadne hneď aj mazať, pošleme peerovi zoznam článkov, ktoré dané noviny ponúkajú. Neposiela sa celý zoznam článkov, ktorý noviny ponúkajú, ale len niekoľko najnovších, pričom toto číslo by malo byť možné zmeniť. V žiadosti o zoznam je taktiež potrebné poslať ktorých novín sa týka, keďže prijímateľ môže mať odoberaných niekoľko novín. Celý tento odsek sa týka kontextu Request.

V kontexte Response sa nachádza samotný zoznam hlavičiek článku, teda správ typu Article.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Položky, prítomné len pri žiadosti (Request kontext):
  - Verejný identifikátor novín, ktorých zoznam článku žiadam.
  - Maximálny počet článkov.
  - Dátum a čas poslednej úpravy najnovšieho článku žiadaného zoznamu. Všetky ďalšie posielané články sú teda staršie ako tento dátum a čas.
- Položky, prítomné len pri odpovedi (Response kontext):
  - Zoznam hlavičiek článkov. Článok v celku je možné požiadať prostredníctvom správy typu ArticleAll.

ArticleDataUpdate

V prípade, že niektorý z peerov zmaže, niektorý zo svojich článkov, je vhodné o tom informovať jeho čitateľov. K správe nepotrebujeme odpoveď je teda vhodné použiť kontext OneWay.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Verejný identifikátor článku.
- Čo sa s článkom stalo, teda či bol stiahnutý, alebo odstránený.

2.9.4 Identifikácia, kľúče

Počas komunikácie peerov dochádza k používaniu viacerých kryptografických kľúčov na overovanie podpisov a dešifrovanie správ. Na výmenu týchto kľúčov si teda potrebujeme zaviesť niekoľko špeciálnych druhov správ, ktoré budú nositeľmi ako kľúčov samotných, tak ďalších prípadných metadát.

Credentials

Správa žiadajúca o identifikačné primitíva peera. Význam jednotlivých dátových položiek závisí na tom, či sa jedná o požiadavku (Request) alebo odpoveď (Response). V prípade žiadosti fungujú dátové položky nasledovne:

- Bit, peer žiada IPv4.
- Bit, peer žiada IPv6.
- Bit, peer žiada verejný RSA kľúč.
- Bit, peer žiada o symetrický EAX kľúč.
- Textový reťazec, IPv4 žiadateľa (ako návrh príjemcovi).
- Textový reťazec, IPv6 žiadateľa (ako návrh príjemcovi).
- PublicKey, verejný RSA kľúč žiadateľa (ako návrh príjemcovi).
- SymmetricKey, symetrický EAX kľúč žiadateľa (ako návrh príjemcovi).

V prípade odpovede, majú dátové položky nasledujúci význam (kontext je vždy z pohľadu správy, teda role odosielateľa a prijímajúceho sú oproti predošlému prípadu vymenené):

- Bit, odpoveď obsahuje IPv4 odosielateľa.
- Bit, odpoveď obsahuje IPv6 odosielateľa.
- Bit, odpoveď obsahuje verejný RSA kľúč odosielateľa.
- Bit, odpoveď obsahuje symetrický EAX kľúč odosielateľa.
- Textový reťazec, hodnota IPv4 odosielateľa (ak je obsiahnutá).
- Textový reťazec, hodnota IPv6 odosielateľa (ak je obsiahnutá).
- PublicKey, hodnota verejného RSA kľúča odosielateľa (ak je obsiahnutá).
- SymmetricKey, hodnota symetrického EAX kľúča odosielateľa (ak je obsiahnutá).

Bitové polia nie sú pri odpovedi potrebné, avšak keďže sa už v správe nachádzajú, tak sa môžu použiť na uľahčenie situácie pri spracovávaní správy po prijatí.

SymmetricKey

Správa, nesúca informáciu o symetrickom kľúči pre dvojicu peerov odosielateľ, prijímateľ. Správa nie je šifrovaná a teda je posielaná v špeciálnom druhu metasprávy.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Bajtový reťazec obsahujúci kľúč (zašifrovaný).
- Bajtový reťazec obsahujúci podpis.

PublicKey

Ďalší typ správy, ktorý nie je šifrovaný a je teda posielaný aj pod iným druhom metasprávy, je správa obsahujúca verejný kľúč, čo tvorí aj jedinú dátovú položku, ktorú správa obsahuje.

2.9.5 Komunikácia o sieti

Nakoľko sa jedná o peer-to-peer sieť, tak je žiadúce, aby sa vymieňali jednotlivé informácie o peeroch, ktorí v sieti sú. Jedná sa o informácie týkajúce sa prevádzky siete, teda o čísla portov a sieťové adresy. Najmä je to potrebné pre redaktorov novín, nakoľko práve oni budú najpravdepodobnejším terčom STUN správ a teda by mali mať, za účelom presmerúvania komunikácie, o sieti prehľad.

UserInfo

Na výmenu informácií o peeroch medzi redaktormi si zavedieme samostatný typ správy. Aby sa neposielali všetky informácie a nezahlcovala sa tak sieť, je vhodné rozdeliť komunikáciu na niekoľko častí.

V prvej časti, peer čo začína komunikáciu, oznámi niektorému svojmu kolegovi redaktorovi, o ktorých peeroch nesie informácie. Môže sa jednať o relatívne dlhý zoznam, preto sa posielajú len verejné identifikátory peerov. Následne, v druhej časti, prijímateľ prejde zoznam peerov a skontroluje, o ktorých peeroch ešte nemá informácie. Opäť v podobe zoznamu verejných identifikátorov peerov odošle prijímateľ odosielateľovi žiadosť, o ktorých peeroch si žiada vedieť podrobnejšie informácie. V tretej časti odošle pôvodný odosielateľ všetky informácie o peeroch, o ktorých ho pôvodný prijímateľ žiadal.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Ktorá časť komunikácie prebieha.

- Zoznam peerov, o ktorých má odosielateľ informácie, resp. o ktorých chce mať prijímateľ informácie.
- Informácie o peeroch, ktoré boli vyžiadané pôvodným prijímateľom. Používa sa na tretiu časť komunikácie.

Gossip

Všeobecne medzi peermi prebieha výmena sieťových informácií pomocou tzv. „gossipingu“, kde si každý peer, v pravidelných časových intervaloch, vyberie niekoľkých susedných peerov a odošle im správu obsahujúcu informácie o opäť náhodných peeroch, o ktorých tieto informácie má. Takto sa postupne, ale náhodne, šíria medzi peermi sieťové informácie o ostatných peeroch.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Sieťové informácie o peeroch.

Ping

V momente, kedy nastane znovupripojenie do siete, musí peer o tomto fakte informovať ostatných peerov, aby v prípade, že došlo k zmene sieťovej adresy, mohli adekvátne upraviť svoje sieťové informácie. Na tento účel sa používa správa „ping“. Hoci je aplikácia navrhnutá tak, aby sa po pripojení peera automaticky aktualizovali jeho sieťové informácie, tak je najlepšie kontaktovať peerov, s ktorými priamo komunikujeme, čo najskôr.

Okrem hlavičky neobsahuje správa žiadne ďalšie dáta.

Správa je v kontexte OneWay, nakoľko na ňu nie je potrebná odpoveď, pretože má len informatívny charakter. Avšak peeri, ktorí využívali odosielajúceho peera ako STUN server musia obnoviť svoju alokáciu a teda ich odpoveď na túto správu je žiadosť o obnovenie alokácie.

2.9.6 Správa novín

Za účelom správy novín je potrebné taktiež si medzi peermi vymieňať správy.

NewJournalist

V prípade, že pridáme do novín nového redaktora je potrebné, aby sa to ďalší peeri redaktorov, ktorí už v novinách pracujú, dozvedeli. Na tento účel si zavedieme správu v kontexte OneWay, nakoľko sa jedná len o informatívnu správu. Správa obsahuje verejný identifikátor a sieťové informácie nového redaktora.

V správe sa okrem hlavičky nachádzajú nasledujúce dátové položky:

- Sieťové informácie o novom redaktorovi.

2.9.7 Serializačné správy

Keďže je potrebné, aby aplikácia dokázala svoj stav po ukončení uložiť a potom po spustení opäť načítať, musíme nejakým spôsobom aplikačné metadáta ukladať. Keďže už máme pre niektoré triedy vytvorené správy, kam budeme zrkadliť ich metadáta kvôli sieťovému prenosu, môžeme tieto správy ďalej rozšíriť a používať na lokálnu serializáciu údajov kvôli perzistentnému ukladaniu dát v aplikácii.

Jednotlivé správy nebudeme podrobne rozpisovať, nakoľko sa v podstate jedná o obraz aplikačných metadát. Spomenieme len, ktoré triedy sa serializujú a ak nejaké sú, tak zmeny oproti ich aplikačnej implementácii.

Serializujú sa, bez výraznejšej zmeny, tieto triedy:

- `IpMap`
- `NewspaperEntry`
- `Peer`

Trieda `IpWrapper` serializuje všetky svoje dátové položky, pričom naviac serializuje aj verejný identifikátor peera, ktorého sa týka. Táto vlastnosť sa ukázala ako vhodná počas vývoja aplikácie.

Ostatné triedy buď to nemajú stav, ktorý by bolo potrebné ukladať, alebo jeho ukladanie nemá zmysel. Tak to je napríklad v prípade triedy `Networking`, kde sa síce nachádza väčšie množstvo dátových štruktúr, ktoré držia informácie o, napríklad, čakajúcich správach. Keďže aplikácia môže byť vypnutá nejaký čas tak nevieme, do akej miery sú dané správy ešte relevantné.

2.9.8 Metaspráva

Metasprávy predstavujú ďalší spôsob rozdelenia správ. Metasprávy vzniknú pripojením metadát pred už serializovanú správu. Metasprávy slúžia na pripojenie tých informácií, ktoré nikdy nie sú šifrované. Ako príklad, prečo je to vlastne potrebné, môže byť to, že peer, ktorý prijme správu, nemusí vedieť od koho daná správa prišla a teda aký symetrický kľúč použiť na jej dešifrovanie. Na rozlíšenie druhov správ v aplikačnom prostredí slúži typ správy tak, ako je zadaný v hlavičke správy. Súčasťou metadát je taktiež trieda správy, ktorá hovorí o tom, či je daná správa symetricky šifrovaná, alebo nie. Asymetrické šifrovanie symetrického kľúča tvorí v tomto prípade výnimku, pretože v tom prípade nie je šifrovaná celá správa, ale len daný kľúč.

Ďalší dôležitý význam metasprávy je ten, že obsahuje položky, ktoré majú vždy pevnú veľkosť a bez ktorých by sme nevedeli, akú veľkosť majú položky s premenlivou veľkosťou. Napríklad dĺžka správy má vždy veľkosť 8 B. Použitím

tejto informácie tak zistíme, akú dĺžku má správa ako taká, nakoľko jej dĺžka je premenlivá podľa jej obsahu.

Existujú dve triedy správ:

- Trieda 0, `ENCRYPTED_MESSAGE`: správa je zašifrovaná symetrickým kľúčom. Obsah správy je pred deserializáciou potrebné rozšifrovať. Používa sa pri štandardných správach ako článok, zoznam článkov a podobne.
- Trieda 1, `PLAIN_MESSAGE`: správa nie je zašifrovaná. Správu je možné priamo deserializovať. Používa sa pri správach obsahujúcich kľúče, či už symetrický, alebo verejný.

Ďalšie informácie ako napríklad typ použitého šifrovania je možné získať z verzie protokolu, ktorý taktiež tvorí súčasť metasprávy s veľkosťou 2 B.

Metaspráva `ENCRYPTED_MESSAGE`

V prípade triedy 0 je okrem verzie a triedy správy uvedený aj inicializačný vektor a jeho dĺžka, dĺžka samotnej správy a verejný identifikátor peera, ktorý správu odoslal. Všetky tieto informácie sú potrebné, aby bolo možné správu dešifrovať.

Metaspráva `PLAIN_MESSAGE`

V prípade triedy 1 je okrem verzie a triedy správy v metaspráve obsiahnutá už len dĺžka samotnej správy, nakoľko všetky ostatné údaje sú obsiahnuté priamo v správe.

2.9.9 Proces výroby a odoslania správy

Z dátových štruktúr v implementácii peera sa pozbierajú všetky potrebné údaje pre vytvorenie požadovanej správy. Následne sa tieto údaje vezmú a vytvorí sa nová Google Protocol buffers správa, do ktorej sa dané údaje zapíšu. Google Protocol buffers správa sa následne serializuje a ak to je potrebné, tak sa správa taktiež zašifruje symetrickým kľúčom. Následne sa pred správu pripoja potrebné metadáta, aby vznikla metaspráva a v tejto chvíli je už možné odoslať metasprávu po sieti.

2.10 Alternatívy

Na alternatívy sa dá hľadieť z dvoch uhlov. Môžme hľadať alternatívy v peer-to-peer sieťach, alebo v nástrojoch na publikáciu článkov, ktoré však nutne nemusia byť peer-to-peer.

2.10.1 Nástroje na publikáciu článkov

Alternatívnymi nástrojmi na publikáciu článkov máme napríklad systémy na správu webového obsahu, medzi ktoré patrí napríklad aj WordPress, Shopify, či Joomla. Tieto sú jednoduché na používanie a ich nasadenie nie je náročné, pričom to zvládne väčšina užívateľov, ktorí si pozrú buď to oficiálnu dokumentáciu, alebo nájdu nejaký návod na používanie z druhej ruky. Jedná sa však o systémy, ktoré nie sú peer-to-peer a teda je potrebný centrálny server. Ďalej, hoci nie je potrebná znalosť webových technológií pri ich používaní, značne to zjednoduší ich používanie a prípadné riešenie problémov.

2.10.2 Alternatívne peer-to-peer siete

Alternatívne môžeme použiť iné peer-to-peer siete. Príkladom sú napríklad siete Gnutella, alebo eDonkey. Gnutella je relatívne stará sieť, ktorej hľadanie peerov funguje na podobných princípoch, ako v nami navrhnutom protokole. eDonkey je sieť, ktorá používa princíp decentralizovaného klient-server, kde síce komunikácia prebieha medzi peermi samotnými, ale stále je potrebná istá forma centrálneho serveru, ktorý slúži na ako vyhľadanie peerov, tak vyhľadávanie obsahu. Replikácia dát je založená čisto na peeroch a na tom, aké súbory sťahujú zo siete. Podobným prípadom sú aj siete založené na trackeroch, čím sú napríklad BitTorrent alebo Direct Connect. Sieť ActivityPub je federatívna, teda sa jedná o sieť zloženú z viacerých serverov, na ktoré sú dáta ukladané, nejedná sa teda o sieť peer-to-peer hoci je obsah do istej miery decentralizovaný. Zaujímavou alternatívou je sieť RetroShare, ktorá je založená na DHT, teda distribuovanej hash tabuľky. Všetky tieto siete majú rôzne prístupy k problematike zdieľania dát a ich decentralizácii, ale spravidla sa jedná o siete, ktoré slúžia čisto na zdieľanie súborov. Akékoľvek hierarchia musí byť prostredníctvom iných nástrojov, ak to je vôbec možné.

Používanie týchto sietí nemusí byť užívateľsky príjemné a vyžaduje často netriviálnu konfiguráciu, ktorá môže byť pre ľudí bez technického vzdelania, či bez záujmu o dané technológie, zložitá a frustrujúca. Náš protokol a jeho aplikácia vyžadujú minimálnu konfiguráciu a sú teda vhodné na pre užívateľov, ktorí sa chcú čo najskôr venovať tvorbe obsahu novín a nie konfigurácii siete, ktorú na tento účel použijú.

Kapitola 3

Kryptografia a šifrovanie

Každá prenášaná správa v protokole triedy `NORMAL_MESSAGE` je symetricky šifrovaná kľúčom, ktorý je samostatne vygenerovaný pre každú dvojicu peerov. Kľúč generuje ten peer, ktorý začína komunikáciu. Správy triedy `KEY_MESSAGE` nie sú šifrované buď to vôbec, tak je tomu v prípade výmeny verejných kľúčov, alebo je ich obsah asymetricky šifrovaný a to v prípade, kedy sa jedná o správy slúžiace na výmenu symetrických kľúčov medzi peermi.

Pri výmene symetrických kľúčov je daný kľúč asymetricky podpísaný a následne šifrovaný, použitím schémy RSA a zarovnania OAEP.

RSA patrí do skupiny schém, ktoré používajú dvojicu kľúčov, jeden verejný a jeden súkromný, resp. privátny. Pred komunikáciou musí prebehnúť výmena verejných kľúčov oboch peerov, aby bolo možné šifrovať ďalšie správy a overovať ich podpisy. Výmena prebieha vo forme čistého, nezašifrovaného textu.

3.1 Definície a terminológia

- Symetrická šifra: Šifra, kde sa na šifrovanie aj dešifrovanie používa ten istý kľúč.
- Ciphertext a plaintext: Ciphertext je text, ktorý vznikne zašifrovaním originálneho, nezašifrovaného textu, teda plaintextu.
- Dôvernosť ciphertextu: Z ciphertextu nie je možné získať plaintext bez znalosti šifrovacieho kľúča.
- Integrita ciphertextu: Ak ciphertext nebol pozmenený.
- AES: Špecifikácia šifrovania dát, vytvorená americkou štandardizačnou komisiou NIST. Jedná sa o variant blokovej šifry Rijndael.

- ECB, CBC: Režimy (anglicky „mode of operation“) symetrických šifier. Sú jednoduché a priamočiare, ale jednoduchšie na prelomenie. Nezabezpečia integritu dát.
- AEAD: Authenticated encryption with additional data. Jedná sa o režim symetrickej šifry, ktorá je overí ako dôvernosť, tak aj integritu ciphertextu.
- EAX, GCM, CCM: Režim symetrickej šifry ponúkajúci AEAD.
- Asymetrická šifra: Taktiež nazývaná ako šifra s verejným kľúčom, je šifra, pri ktorej je použitá dvojica kľúčov, jeden verejný a jeden privátny, na šifrovanie a dešifrovanie správ. Na šifrovanie správ sa používa verejný kľúč, na dešifrovanie privátny kľúč. Ako názov napovedá, verejný kľúč môže k šifrovaniu správy použiť ktokoľvek, ale k dešifrovaniu je potrebný privátny kľúč, ktorý má len jeho majiteľ.
- RSA: Asymetrická šifra popísaná pánmi Ron Rivest, Adi Shamir a Leonard Adleman, ktorých priezviská tvoria daný akronym. Široko používaná a v dnešnej dobe populárna asymetrická šifra, aj keď málo kedy používaná samostatne.
- Zarovnávacia schéma: Spôsob úpravy správy pred tým, ako bude šifrovaná, aby sa odstránili nedostatky používanej šifry, alebo, aby sa zamaskovala existencia nejakého vzoru v šifrovanej správe.
- OAEP: Zarovnávacia schéma, ktorá sa v kombinácii s RSA ukazuje ako silne odolná voči CCA útokom.
- MITM : Man in the middle. Typ útoku, pri ktorom útočník nabúra komunikáciu medzi dvoma komunikujúcimi stranami. Následne ich komunikáciu nie len odpočúva, ale môže ju aj meniť.

3.2 Symetrické šifrovanie, symetrické kľúče, AES a EAX

Pre väčší objem dát je vhodnejšie symetrické šifrovanie, pretože, na rozdiel od šifrovania asymetrického, nezväčšuje asymptoticky veľkosť zašifrovaného obsahu, množstvo dát, ktoré sa dá naraz zašifrovať je obmedzené a jednoduché rozdelenie na menšie celky nepomôže (výsledný obsah sa stane náchylným na tzv. Watermarking attack) a väčšina implementácií je pre asymetrické šifrovanie pomalšia, ako pre symetrické.

Z princípu teda plynie, že symetrické šifrovanie je vhodnejšie pre väčšie množstvo dát.

Symetrické šifrovanie prebieha použitím tzv. AES ktorý operuje nad ECB alebo blokovou šifrou CBC. Tieto však podporujú len dôvernosť informácií, teda to, že nedokáže nikto, kto nemá kľúč informácie rozšifrovať a prečítať. Problém ale nastáva, keď sa jedná o autenticitu týchto informácií, teda toho, či niekto pozmenil šifrované dáta, alebo nie (teda integrita dát ostane zachovaná). V tomto prípade ak by došlo k zmene informácií, tak by to protistrana zistila počas dešifrovania dát.

Preto AES operuje nad režimom EAX. To zaručuje ako autenticitu dát, tak aj ich dôvernosť [4]. EAX patrí do kategórie AEAD, teda nie len, že je možné dáta zašifrovať a neskôr overiť ich identitu, ale je taktiež možné priložiť ďalšie dáta, ktoré nie sú šifrované. Jedná sa o takzvané AAD (authenticated additional data). Tieto dáta síce nie sú zašifrované, a teda si ich môže prečítať ktokoľvek, ale ich integrita bude zachovaná počas prenosu, čo znamená, že akékoľvek neželané zmeny budú odhalené.

3.2.1 Alternatívy

Existuje viacero alternatív k jednotlivým spôsobom šifrovania, dôvernosti a autentikácie.

Prvou alternatívou je použiť na šifrovanie správ štandardnú šifru, napr. blokové CBC (spolu s AES) a následne pomocou hashovania overiť autenticitu. Zdá sa mi však, že v tomto prípade bolo lepšie ostať pri EAX, ktorá spĺňa obe požiadavky súčasne.

Druhou alternatívou by bolo použiť namiesto EAX režim GCM, ktorý taktiež patrí do skupiny AEAD režimov. GCM je na rozdiel of EAX paralelizovateľné, záleží však od implementácie, či túto vlastnosť použije. GCM operuje nad binárnym Galoisovým telesom, pričom operácie nad takýmito telesami vedia lepšie využiť reálny hardvér, čo je výhodou napríklad oproti režimu CWC ktorý používa 127-bitové teleso celých čísel ale s ktorým si je GCM veľmi podobné.

Treťou alternatívou je ísť ešte o úroveň vyššie a použiť IES (Integrated Encryption Scheme). IES je schéma, ktorá spája dôvernosť a integritu dát dokopy. Jedná sa o hybridnú schému a teda nie je potrebné samostatne používať symetrické a asymetrické šifrovanie. Druhmi IES sú napríklad ECIES (Elliptic Curve Integrated Encryption Scheme) a DLIES (Discrete Logarithm Integrated Encryption Scheme). Výhodou je teda jednoduchšie použitie a aj úspora, vzhľadom na to, že sa dá zašifrovať väčšie množstvo dát na rovnakú dĺžku kľúča (v porovnaní s RSA).

Ďalej by bolo možné nahradiť blokovú šifru AES inými alternatívami ako napríklad 3DES, ktorá vzniká z trojnásobného použitia DES. DES je navrhnutá tak, aby lepšie pracovala na hardvéri ako softvéri a slúži na šifrovanie textu

do 64-bitových blokov použitím 56-bitového kľúča. Z dôvodu technologického pokroku už však schéma DES nie je bezpečná, pretože je prelomiteľná bruteforce útokom. [5] 3DES sa ukazuje ako menej náchylné k bruteforce útokom, avšak jeho kľúč je stále len zloženie troch kópií toho istého, maximálne 56-bitového kľúča. AES operuje nad 128, 192 a 256 bitovými kľúčmi, čím rastie miera odolnosti voči bruteforce útokom. Pri použití AES s 128-bitovým kľúčom by nám overenie všetkých možných kľúčov pri rýchlosti päťdesiat miliárd kľúčov za sekundu trvalo $5 \cdot 10^{21}$ rokov, naproti tomu prejsť každý kľúč šifry 3DES, kde veľkosť kľúča je 112 bitov, rovnakou rýchlosťou, by trvalo len 800 dní [6]. Výhodou 3DES je však spätná kompatibilita s DES a je teda možné použiť existujúce riešenia z DES pre 3DES.

3.2.2 Generovanie kľúčov

Kľúče sa generujú s dĺžkou 16 bajtov (128 bitov). Generujú sa pre každú dvojicu peerov zvlášť.

V protokole sú symetrické kľúče, na rozdiel od kľúčov asymetrických, určené len a len na interné účely a nie je dovolené používať vlastne generované kľúče. Kľúč generuje spravidla ten peer, ktorý začína s komunikáciou.

Pri generovaní sa musí použiť pseudonáhodný generátor s čo najväčšou mierou entropie. Súčasťou generátora by napríklad mohol byť obsah súboru z Unix-like operačných systémov `/dev/urandom`, alebo jeho varianty pre jednotlivé distribúcie, kombinovaný s aktuálnym časom. Príkladom takého generátora je AutoSeededRandomPool z knižnice Crypto++.

3.3 Asymetrické šifrovanie, asymetrické kľúče, schémy RSA a OAEP

Vzhľadom na to, že sa na šifrovanie komunikácie používa symetrická šifra AES s režimom EAX, je potrebné nejakým spôsobom tento symetrický kľúč predať protistrane. Posielať ho ako nezašifrovaný, pôvodný text by znamenalo, že by mohol útočník podstrčiť falošný symetrický kľúč. Preto sa použije asymetrické šifrovanie a kľúč sa zašifruje a podpíše.

Pre asymetrické šifrovanie sa používa schéma RSA, ktorú je možné použiť aj pre digitálny podpis. Učebnicové RSA má však niekoľko nevhodných vlastností, preto ho rozšírime o OAEP padding, ktoré niektoré tieto nevýhody odtraňuje. O nevýhodách a rizikách RSA pojednávajú články [7], [8] a [9].

3.3.1 Alternatívy

Ako alternatívu voči RSA by bolo možné použiť kombináciu RSA a DSA. DSA je algoritmus na digitálne podpisovanie a overovanie. V prípade kombinácie týchto dvoch sa použije RSA na asymetrické šifrovanie a DSA na podpis.

Tieto algoritmy je možné ešte ďalej skombinovať a vytvoriť tzv. RSA–DSA kombinačný algoritmus. Tento algoritmus dosahuje lepších výsledkov čo sa týka jeho časovej dĺžky behu voči nijak neupravenej kombinácii RSA a DSA. Schopnosťou tohto algoritmu je ako šifrovanie, tak aj podpisovanie. Kombinačný algoritmus zaostáva len v rýchlosti generovania kľúčov, kde sú samostatné RSA a DSA spolu rýchlejšie. [10]

V kombinačnom algoritme však dochádza ku generovaniu dvoch verejných a dvoch súkromných kľúčov. Pre zjednodušenie komunikácie medzi peermi sa teda uprednostnilo čisté RSA.

Ďalej existuje väčšie množstvo algoritmov, ktoré je možné použiť. Napríklad ECDSA, ktoré je ale skôr alternatívou k DSA a nie RSA ako takému. Zaujímavou alternatívou je ale napríklad ECC, ktoré je rovnako asymetrickou šifrou. Kľúče sú spravidla kompaktnejšie, čo zaručuje vyššiu efektivitu. Jedná sa však o novšiu šifru a teda sme sa rozhodli použiť staršie RSA, ktoré je, vzhľadom na svoj vek, stále považované za bezpečné a tak vzbudzuje pocit spoľahlivosti.

3.3.2 Generovanie kľúčov

Kľúče (verejný aj privátny) sú buď to generované počas vytvárania peera, alebo môže užívateľ použiť už vopred vygenerované kľúče, napríklad, ak užívateľ požaduje použitie nejakého jedného kľúča vo viacerých aplikáciách. Aplikácia implementujúca protokol však musí vždy poskytnúť možnosť vygenerovať ako symetrický, tak asymetrický kľúč.

Ako pri symetrických kľúčoch, ak aplikácia generuje svoje vlastné asymetrické kľúče, tak sa pri generovaní musí použiť v pseudonáhodnom generátore zdroj s čo najväčšou mierou entropie, ako napríklad obsah súboru `/dev/urandom`, kombinovaný s aktuálnym časom. Príkladom takého generátora je AutoSeededRandomPool z knižnice Crypto++.

3.4 Výmena kľúčov

3.4.1 Výmena verejných kľúčov RSA

Sekcia obsahuje popis postup generovania a zdieľania verejného kľúča v NP2PS sieti medzi dvoma peermi. Uvažujeme teda dvoch peerov, pomenujeme ich Alica a Bob.

Alica chce poslať Bobovi správu a chce ju asymetricky zašifrovať. Môže sa napríklad jednať o výmenu symetrického kľúča. Alica teda potrebuje Bobov verejný kľúč.

Ak Alica má Bobov verejný kľúč, správu podpíše, zašifruje a pošle. Žiadna výmena kľúčov nie je potrebná.

Ak Alica nemá Bobov verejný kľúč, je potrebné, aby ho získala. Alica teda kontaktuje nejakú autoritu v sieti, ktorá jej bude schopná poskytnúť Bobov verejný kľúč. V NP2PS sieti sa jedná o šéfredaktora novín, v ktorých sieti sa snažíme komunikovať. Ak teda je Bob v tejto sieti, tak sa musel do nej prihlásiť práve skrz šéfredaktora novín. Tento šéfredaktor teda má jeho verejný kľúč a môže ho poskytnúť ďalším peerom v sieti. Ak Bob v sieti nie je, nebude u šéfredaktora zaregistrovaný a on nebude mať odkiaľ poskytnúť Bobov verejný kľúč.

Ak autorita, teda šéfredaktor, má Bobov verejný kľúč, odošle Alici pozitívnu odpoveď spolu s Bobovým verejným kľúčom. Ak však autorita tento kľúč nemá, musí odoslať negatívnu odpoveď.

Alica prijme odpoveď od autority. Ak bola odpoveď pozitívna, Bobov kľúč si uloží a použije ho pri šifrovaní správy, resp. pri overovaní Bobových digitálnych podpisov. Ak bola odpoveď negatívna, musí Alica odosielanie správy zanechať.

Z uhla pohľadu Boba sa deje to isté, avšak komunikáciu nezačína on, ale Alica.

3.4.2 Výmena kľúča AES

Sekcia obsahuje popis postup generovania a zdieľania symetrického kľúča v NP2PS sieti medzi dvoma peermi. Uvažujeme teda dvoch peerov, pomenujeme ich Alica a Bob.

Alica si chce s Bobom vymeniť správu. Obe strany si vymenili verejné kľúče a teraz si potrebujú vymeniť symetrický kľúč.

Keďže Alica iniciuje komunikáciu, tak ako prvá zistí, že symetrický kľúč ešte nebol vygenerovaný. Symetrický kľúč sa teda vygeneruje, podpíše Aliciným privátnym kľúčom, zašifruje Bobovým verejným kľúčom a odošle sa Bobovi. Pokiaľ Bob nepošle potvrdenie, že správu dostal, Alica pozdrží odosielanie správy.

Bob prijme symetrický kľúč od Alice a postupuje opačne. Najprv použije svoj privátny kľúč k dešifrovaniu obsahu správy a následne si overí Alicin digitálny podpis jej verejným kľúčom. Ak tento proces prebehne bezproblémovo, Bob si kľúč uloží a odošle Alici potvrdenie o tom, že kľúč prijal. Ak proces prebehne s chybami, alebo neúspešne, Bob správu zahodí.

Alica nakoniec prijme Bobove potvrdenie výmeny kľúča a odošle pôvodnú správu.

Poradie podpisu a šifrovania

Správu je možné buď to najprv zašifrovať a tak podpísať, alebo ju najprv podpísať a tak zašifrovať. Ukazuje sa, že prvý variant je slabý a nebezpečný [11], preto sme sa rozhodli použiť druhý prístup, teda text sa najprv podpíše a až tak sa zašifruje.

3.5 Kryptografický hash

Pri používaní aplikácie potrebujeme nejakým spôsobom overiť, či články získavané zo siete od peerov sú skutočne tie, ktoré boli pôvodne napísané ich autormi. Na tento účel potrebujeme tieto články nejakým spôsobom spracovať a z ich obsahu vytvoriť niečo, napríklad nejaké číslo, ktoré bude unikátne pre daný obsah článku a v momente, kedy by došlo k prepísaniu jeho obsahu by došlo aj k zmene daného čísla. Nazvime toto číslo hashom. Takto je možné zistiť, či došlo k zmene obsahu článku ak vieme, aké toto pre obsah článku unikátne číslo má byť. Jeho pravú hodnotu sa dozvieme od šéfredaktora, resp. redaktorov daných novín.

Rozdielom medzi kryptografickým hashom a „normálnym“ hashom je najmä čo najväčšie zníženie pravdepodobnosti kolízií. V našom prípade kolíziou rozumieme to, že by sa po šikovnej úprave obsahu článku mohlo stať, že sa z neho vyráta rovnaký hash, ako z originálu. Vlastností je tam viac, napríklad, že výsledný hash vyzerá náhodne a tak ďalej.

Použijeme niektoré z kryptografických hashov z knižnice Crypto++. Tá ich má k dispozícii niekoľko, vrátane najznámejších a najpoužívanejších ako SHA, MD, Whirlpool, ale aj Keccak. Z týchto spomenutých je práve MD5 známa náchylnosťou na hash kolízie, čo by v našom prípade znamenalo, že by sa po šikovnej úprave obsahu článku mohlo stať, že sa z neho vyráta rovnaký hash, ako z originálu. Spomedzi ďalších Whirlpool pevne operuje nad 512-bitovými hodnotami hashov, čo v prípade, že by bolo potrebné túto hodnotu navýšiť, bolo prekážkou. Ostáva nám teda SHA, kde je možné konfigurovať dĺžku hashu a Keccak, ktorý je postavený ako náhrada za SHA, často považovaný za SHA3. Rozhodli sme sa použiť práve tento algoritmus, nakoľko je SHA3 resp. Keccak považovaný za následníka SHA2, ktoré samotné bolo následníkom pôvodného SHA.

Kapitola 4

Sieťovanie

Vzhľadom na to, že sa jedná o sieťový komunikačný protokol, tvorí sieťovanie značnú časť implementácie protokolu. Protokol implementujeme na aplikačnej vrstve modelu TCP/IP. V modeli OSI to je siedma vrstva, ale tento model budeme ďalej ignorovať, poprípade ho budeme porovnávať s TCP/IP modelom, do ktorého kontextu budeme implementáciu klásť. Tento model je dostatočne zaužívaný a robustný na naše účely, ale aj všeobecne pre účely modernej internetovej komunikácie a nadbytočné, často nevyužité vrstvy OSI modelu by len prekážali pri abstrakcii, na ktorú sa budeme snažiť implementáciu protokolu namapovať.

4.1 Definície a terminológia

- STUN: Session Traversal Utilities for NAT. Protokol slúžiaci na obídenie a celkové vysporadúvanie sa s NAT-om. STUN ponúka prostriedky na zistenie IP adresy a portu priradených NAT-om, ktoré priamo súvisia s privátnou IP adresou a portom. Implementované podľa RFC8489.
- STUN agent: Entita implementujúca STUN. Môže sa jednať o STUN klienta alebo STUN server.
- STUN klient: Entita posielajúca STUN žiadosti, prijímajúca STUN odpovede a STUN indikácie. STUN indikácie môže STUN klient tiež aj posielať.
- STUN server: Entita prijímajúca STUN žiadosti, či STUN indikácie a posielajúca STUN odpovede. STUN indikácie môže STUN server tiež aj posielať.
- TURN: Traversal Using Relays around NAT. Protokol používaný TURN klientom a TURN serverom. Umožňuje TURN klientovi naviazať spojenie s TURN peerom s tým, že všetka komunikácia bude prebiehať skrz TURN server. Implementované podľa [2].

- NAT: Network Address Translation. Slúži na preklad privátnych adries v rámci lokálnej siete (LAN) do verejných adries v rámci WAN.

4.2 Využitie modelu TCP/IP

4.2.1 IPv4

Na sieťovej vrstve sa používa protokol IP verzie 4. Podpora pre protokol IP verzie 6 je čiastočne v implementácii zavedená, ale nie do takej miery, aby podporovala plnú funkcionalitu protokolu. Cieľom práce je však navrhnúť a ukázať funkčnosť protokolu v reálnej prevádzke, kde IPv4 stále tvorí dominantný podiel medzi sieťovými protokolmi. Ku dňu 10.5.2023 bolo podľa štatistík spoločnosti Google v Českej republike len 24,11% celkového počtu prístupov ku Google infraštruktúre skrz IPv6. V Slovenskej republike sú tieto čísla pre rovnaký deň ešte nižšie, len 8,9%.

4.2.2 Protokol TCP a alternatívy.

Na tretej vrstve, na rozdiel od predošlých dvoch vrstiev, existuje niekoľko protokolov, ktoré je možné použiť. Vyberali sme zo štyroch možností: TCP, UDP, SCTP a DCCP. Vzhľadom na to, že sme pri implementácii používali framework Qt, tak protokol DCCP neprichádza do úvahy, pretože ho tento framework nepodporuje. Implementácia tohto protokolov do frameworku Qt by bola možná, ale to nie je cieľom tejto práce.

Protokol SCTP je podporovaný frameworkom Qt. Avšak väčšie skúsenosti máme s používaním protokolov TCP alebo UDP, pričom nad týmito protokolmi je popísaný aj protokol STUN [2]. Využitie protokolu SCTP by teda vyžadovalo adekvátne prerobiť protokol STUN, aby ho bolo možné použiť ako SCTP aplikáciu.

Ostávajú už len dva protokoly, TCP a UDP. Obidva protokoly sú široko podporované a používané. Navzájom sú však veľmi odlišné, viď. tabuľku 4.1.

Vlastnosť	**TCP**	**UDP**
Spoľahlivosť	Spoľahlivý	Nespoľahlivý
Typ spojenia	Spojovaný	Nespojovaný
Doručenie	Usporiadané	Neusporiadané
Správa toku	Áno	Nie
Správa zahltenia	Áno	Nie

Tabuľka 4.1 Rozdiely medzi protokolmi TCP a UDP.

Z tabuľky vidíme, že jednoduchší na používanie bude protokol TCP, z jeho vlastností však neplynie len jeho jednoduchšie použitie, ale všetky z nich sú pre náš protokol žiadúce. Textová správa, ktorá sa posiela po sieti musí byť doručená tak, ako bola odoslaná, teda nesmie dôjsť k strate žiadneho paketu, z čoho vyplýva požiadavka na spoľahlivosť. Vytvorením spojenia zistíme, či sa nám podarilo sa skontaktovať s druhým peerom a ak nie, tak je vysoko pravdepodobné, že sa nám to ani nepodarí. Naproti tomu, ak by mal peer odoslať potvrdzujúcu správu UDP, nevedeli by sme, či sa správa stratila, alebo sa nám nepodarilo pripojiť, alebo sa nám pripojiť podarilo ale druhý peer odmieta komunikovať. Nadviazaním spojenia získame istotu komunikácie s protistranou.

Protokol UDP má však výhodu pri obchádzaní systému na preklad sieťových adries, tzv. NAT-u.

Aj tak sme sa však rozhodli pre použitie protokolu TCP. Protokol nám svojimi vlastnosťami ušetrí plno práce pri veciach, ktoré by sme nad UDP museli pracne implementovať sami. Spojovaných vlastností TCP taktiež využijeme pri obchádzaní NAT-u.

4.2.3 Štvrtá vrstva

Štvrtá vrstva je vrstva aplikačná. Na tejto vrstve implementujeme náš protokol, spolu v kombinácii s protokolom STUN. Tieto dva protokoly navzájom spolupracujú len skrz aplikáciu a inak bežia navzájom nezávisle.

Čísla portov

Pre tú časť peera, ktorá slúži ako prijímateľ komunikácie NP2PS, je otvorený TCP port s číslom 14128. Jedná sa o port z rozsahu užívateľských portov a na jeho otvorenie a obsluhu by nemali byť potrebné administrátorské práva, spustiť NP2PS a aj STUN protokol teda dokáže aj bežný užívateľ. Pre STUN server je otvorený TCP port s číslom 3478, tak ako ho definuje [2].

Je potrebné podotknúť, že číslo portu protokolu NP2PS nie je u IANA registrované a teda je možné, že hodnotu 14128 používa aj nejaká iná služba či protokol. V čase písania tejto práce nebolo však toto číslo portu priradené k žiadnej službe ani protokolu a to ani v jednom zo štyroch spomínaných transportných protokolov.

Čísla portov NP2PS a STUN je možné v konfigurácii zmeniť, aby sa bolo možné pripájať k peerom, ktorí ho majú premapované na svojich domácich routeroch. Zmena je možná aj počas exportovania súboru o novinách, ktorý popisuje ich redaktorov, a kde sa k novinám pripojiť.

4.3 Pripojenie do siete

Každý peer, ktorý sa chce pripojiť do siete a stať sa NP2PS peerom najprv kontaktuje šéfredaktora novín, do ktorých siete sa chce pripojiť na jeho transportnej adrese. Ten sa mu identifikuje a následne klient požiada o alokáciu svojej reflexívnej transportnej adresy na serveri. Táto alokácia je potrebná, nakoľko by inak server nemal odkiaľ vedieť, kde sa daný peer nachádza v prípade, že by s ním chcel iný peer komunikovať.

Po tom, čo prebehne táto počiatočná výmena informácií už môže peer posielať NP2PS časti servera NP2PS správy a komunikovať v danej sieti.

4.4 Komunikácia

Komunikácia prebieha použitím už spomínaného protokolu TCP. Každý peer, ktorý sa pripojí do siete otvorí porty 14128 a 3478. Týmto sa aktivujú dve časti peera, ktoré slúžia na prijímanie správ od peerov, ktorí chcú vytvoriť s daným peerom nové spojenie. Odosielanie správ funguje na rovnakom princípe, na akom funguje aj v prípade klient-server sietí. Spojeniu sa na pridelí náhodný port na strane vysielajúceho a následne skrz tento port prebieha všetka komunikácia s prijímajúcim peerom.

4.4.1 Triedy `QTcpServer` a `QTcpSocket`

V implementácii sa používa implementácia TCP socketu z frameworku Qt ktorá nesie názov `QTcpSocket`. Trieda dedí z objektu `QAbstractSocket`, ktorý deklaruje všetky potrebné funkcie a signály na komunikáciu po sieti.
`QAbstractSocket` je univerzálny a použiteľný aj pre protokol UDP.

Pripájanie, z pohľadu klienta

Na pripojenie sa k hostiteľovi, v našom prípade teda k ďalšiemu peerovi, sa používa funkcia `connectToHost`, ktorá má nasledujúce argumenty:

- `QString` alebo `QHostAddress` – Adresa, alebo názov hostiteľa (anglicky „hostname“), ku ktorému sa má klient, resp. peer pripojiť.
- `quint16` – Číslo portu protokolu transportnej vrstvy, na ktoré sa má klient, resp. peer pripojiť.
- `QIODeviceBase::OpenMode` – Režim, v akom bude možné socket použiť, teda či je určený na zápis, čítanie, alebo oboje.

- `QAbstractSocket::NetworkLayerProtocol` – Protokol sieťovej vrstvy, ktorý sa má použiť. Len v prípade, že sa v prvom argumente použije `QString` na predanie adresy. Inak objekt `QHostAddress` obsahuje informáciu o tom, aký protokol sieťovej vrstvy sa používa.

Po úspešnom pripojení sa k hostiteľovi, resp. peerovi, vyšle socket signál `hostConnected`. Stav socketu sa nastaví na `ConnectedState` a v tejto chvíli už môžeme začať komunikáciu na aplikačnej vrstve, či už cez port 3478 a STUN, alebo 14128 a NP2PS.

Ak pripojenie nie je úspešné a zlyhá, vyšle socket signál `errorOccured`. Pomocou metódy `error` je potom možné zistiť, aká chyba nastala. Pre náš účel budú dôležité najmä chyby `ConnectionRefusedError` a `SocketTimeoutError`, na ktoré si musíme dávať pozor, ak sa budeme pokúšať sa pripojiť k peerovi, ktorý sa môže nachádzať za NAT-om. Prvá znamená, že hostiteľ, resp. peer odmietol pripojenie, tá druhá zas hovorí o vypršaní času pri pripájaní sa k peerovi. Pri testovaní v modelovej sieti, ak sa k peerovi za NAT-om nepodarilo pripojiť, tak chyba bola vždy tá druhá, ohľadom vypršania času na pripojenie.

Ďalšia chyba, ktorá je v implementácií kontrolovaná a sledovaná, je chyba `RemoteHostClosedError`. Táto chyba sa vyšle v prípade, že sa klient, resp. peer odpojí od hostiteľa. Pomocou tejto chyby vieme skontrolovať, ktorý peer sa od hostiteľa odpojil a socket, ktorý pôvodne slúžil na komunikáciu s ním, už nie je potrebný. Taktiež ho môžme vyškrtnúť zo zoznamu čitateľov jednotlivých článkov, pretože už nie je súčasťou siete a ďalší peeri sa k nemu nedokážu pripojiť.

Pripájanie, z pohľadu hostiteľa

V prípade, že sa chceme umožniť ostatným klientom pripájať, môžeme použiť triedu `QTcpServer`. Táto trieda má definovanú metódu `listen`, ktorá spôsobí, že server začne počúvať nové pripojenia na nejakej adrese IP a porte TCP. Metóda má dva argumenty:

- `QHostAddress` – Adresa IP, na ktorej má server počúvať nové pripojenia. Aplikácia implementácie protokolu umožňuje si zvoliť žiadanú adresu IP spomedzi všetkých dostupných adries.

- `quint16` – Číslo portu transportného protokolu TCP. Vždy sa použije buď to hodnota 14128 pre protokol NP2PS alebo 3478 pre protokol STUN.

Po pripojení sa klienta k hostiteľovi vyšle `QTcpServer` signál `newConnection`. V tejto chvíli je možné si pomocou metódy `nextPendingConnection` vyžiadať súvisiaci `QTcpSocket` pre komunikáciu s daným klientom. To, že je správa pripravená na čítanie, zistíme prostredníctvom signálu `readyRead`.

Každý z protokolov NP2PS a STUN majú svoj objekt `QTcpServer`. To zodpovedá tomu, že dané protokoly pracujú nezávisle a ak nejaká ich spolupráca existuje, tak len skrz logiku implementácie.

Ukončenie spojenia

V prípade, že peer už nechce ďalej komunikovať po danom sockete, môže na ukončenie spojenia použiť funkciu `disconnectFromHost`. Implementácia však v bezchybnom chode nikdy túto funkciu nevolá, pretože všetky spojenia, či už z protokolu NP2PS alebo STUN, ostávajú otvorené a pripojené počas celej prítomnosti peera v sieti.

Po odpojení sa stav socketu nastaví na `UnconnectedState`. To však platí len pre klienta a nie pre hostiteľa, ktorého socket stále zostane v stave ako predtým. Naproti tomu sa mu však nastaví chyba `RemoteHostClosedError`.

Inštancie triedy `QTcpSocket`

Jednotlivé inštancie triedy `QTcpSocket` je možné získať dvoma spôsobmi.

Prvým spôsobom je ho získať z metódy `nextPendingConnection` na objekte `QTcpServer`. Táto metóda vráti ukazovateľ na socket, ktorý je možné používať pre komunikáciu s pripájajúcim sa klientom, resp. peerom. Tento ukazovateľ je uložený do triedy `IpMap` a použitý neskoršie na ďalšiu komunikáciu s pripájajúcim sa peerom. O uvoľnenie pamäte sa stará trieda `QTcpServer`.

Druhým spôsobom získame `QTcpSocket` jeho vytvorením pomocou kľúčového slova `new`. Takýto postup síce nie je vhodný pre moderné C++, ale je to postup, ktorý je zaužívaný vo frameworku Qt. O uvoľnenie pamäte sa postará slot `deleteLater`, na ktorý je potrebné tento socket pomocou `QObject::connect` napojiť.

4.4.2 Sieťový peer

Z pohľadu sieťovej komunikácie je vhodné si sieťového peera rozdeliť na štyri časti, pričom každá polovica je určená pre jeden aplikačný protokol. Jednotlivé polovice sú následne delené na klienta a na server, ako tomu hovoríme pri protokole STUN, alebo na časť prijímajúcu, skrátene prijímateľ, a časť odosielajúcu, skrátene odosielateľ, ako to nazývame pri protokole NP2PS. Prijímateľ spolu so serverom slúžia na nadviazanie nových spojení a sú reprezentovaní samostatne inštanciami triedy `QTcpServer`. Odosielateľ a klient slúžia na odosielanie správ a teda na nadväzovanie nových spojení s ostatnými peermi. Jedná sa o tie časti peera, ktoré začínajú sieťovú komunikáciu. Server, resp. prijímateľ, nikdy sieťovú komunikáciu nezačína, len odpovedá na prichádzajúce spojenia.

V kontexte klient-server sieťového modelu je peer v peer-to-peer sieti implementovaný ako kombinácia klienta a servera. Z toho vyplýva fakt, že jednotliví peermi nie sú z hľadiska sieťovej komunikácie nijak rozdelení a každý z nich istým spôsobom riadi danú peer-to-peer sieť a poskytuje v nej isté služby.

NP2PS prijímateľ

Jedná sa o tú časť peera, resp. aplikácie, ktorá prijíma správy z NP2PS peer-to-peer siete na porte 14128. Ak by sme po vzore klient-server rozdelili komunikačné role peera medzi klienta a server, patrila by táto časť k serveru. Na toto číslo portu, skrátene portu, sa pripájajú tí peeri, ktorí ešte s daným peerom nekomunikujú a chcú komunikáciu započať. Ak však už bola komunikácia započatá, je možné na komunikáciu používať už existujúce otvorené spojenie TCP a to aj v prípade, že komunikácia, kvôli ktorej bolo pôvodne spojenie nadviazané, už dávno skončila a to aj pre všetky náhodné čísla portov, ktoré boli pridelené pri vytváraní spojenia.

Pripojenie sa teda po ukončení komunikácie, kvôli ktorej bolo vytvorené nezatvára. To z toho dôvodu, že to pomôže v prípade, keď sa jeden z peerov nachádza za NAT-om. V tomto prípade by vzhľadom na tok dát v peer-to-peer sieti, ktorý rastie spolu s veľkosťou siete, bolo potrebné pri každej komunikácii nanovo otvárať spojenie prostredníctvom niektorej z techník na obchádzanie NAT-u, pričom v mnohých prípadoch by to aj tak skončilo presmerovaním komunikácie skrz niektorý zo STUN serverov. Podotknime, že toto označenie nie je úplne správne, pretože STUN server zo svojej definície neslúži na presmerovanie komunikácie, my ho však rozšírime a dodáme mu túto funkcionalitu. Na niečo podobné taktiež slúži protokol TURN [1], ten ale nie je vhodný pre náš účel.

Po prijatí prichádzajúceho spojenia si peer prečíta správu. Po prečítaní správy je správa spracovaná podľa toho, o akú správu sa jedná. Následne sa vytvorí nová správa ako odpoveď a odošle sa.

NP2PS odosielateľ

Jedná sa o tú časť peera, ktorá odosiela správy do NP2PS peer-to-peer siete. V kontexte klient-server modelu by sa jednalo o klienta. Úlohou odosielateľa je vytvárať nové spojenia s NP2PS prijímateľom a použiť toto spojenie na odosielanie správ.

Ak peer vytvorí spojenie s iným peerom, ktorý je napríklad šéfredaktorom niektorých novín, pridelí sa mu náhodný port odchádzajúceho spojenia a prostredníctvom tohto portu prebieha komunikácia. Ak je správa, ktorú peer posielal už odkomunikovaná, spojenie sa nezatvorí, ale ostane otvorené, aby daného peera mohol v prípade potreby druhý peer kontaktovať. Vzhľadom na to, že sa jedná o peer-to-peer sieť, očakáva sa, že bude komunikácia často prebiehať z dôvodu

prostého chodu siete, kde budú ostatní peeri od neho žiadať jeho stiahnuté, alebo napísané články.

STUN server

Časť peera, ktorá je zodpovedná za prijímanie STUN správ, sa nazýva STUN server. Protokol STUN je založený ne princípe klient-server a teda aj v jeho oficiálnej dokumentácii je toto označenie používané rovnakým spôsobom, ako v tejto práci. STUN server prijíma spojenia na čísle portu 3478.

Po prijatí správy je práva prečítaná a spracovaná. Následne sa po danom sockete odošle odpoveď. Podobne ako je to v prípade prijímateľa NP2PS, je po vytvorení spojenia toto spojenie udržiavané otvorené pre budúce použitie. Z tohto dôvodu je teda otázne, či sa skutočne jedná o tradičný, čistý server, ktorý nikdy nezačína komunikáciu, pretože z dôvodu presmerovania správ tento STUN server začne komunikáciu s nejakým STUN klientom. Zakaždým však komunikácia prebieha po spojení, ktoré bolo predtým týmto klientom vytvorené a otvorené. Opäť sa však nejedná o vlastnosť STUN-u tak, ako ho popisuje jeho definícia, ale o naše vlastné rozšírenie.

STUN klient

STUN klient je posledná časť sieťového peera. Vytvára spojenia so STUN serverom a posiela mu požiadavky, alebo indikácie. Pri požiadavkách očakáva STUN klient odpoveď s buď to úspešným vykonaním, alebo s chybou. STUN klient po ukončení komunikácie správy spojenie nezatvára a necháva ho otvorené pre účely obchádzania NAT-u.

Z dôvodu vlastného rozširovania protokolu STUN sa však, rovnako ako pri STUN serveri, nejedná o čistého klienta. Klient teda za istých podmienok môže prijímať správy od servera v rámci komunikácie, ktoré on sám nezačal.

4.5 Preklad sieťových adries

Vo všeobecnejšom význame je preklad sieťových adries zobrazenie, ktoré zobrazí sieťové adresy z jednej skupiny na sieťové adresy druhej skupiny. V užšom zmysle a v zmysle, v ktorom to v tejto práci používame, sa jedná o preklad transportných adries v rámci privátnej siete na transportné adresy verejnej siete, pričom sa ale pre celú privátnu sieť použije len jedna sieťová adresa.

Anglicky sa táto metóda nazýva „Network address translation“, pričom v tejto práci sa na označenie tejto metódy spravidla používa skratka NAT.

NAT má niekoľko rôznych variant, pričom ale všetky pracujú transparentne, čo sa týka prideľovania adries a ich následného prekladu. Existujú dva hlavné druhy prideľovania adries:

- Statický, kde sa jedná o pridelenie jednej verejnej sieťovej adresy jednej privátnej sieťovej adrese. V tomto prípade nie je potrebné NAT obchádzať, pretože jednotlivé IP adresy sú presne zobrazené jedna na druhú a pri pokuse o pripojenie sa skrz takýto NAT je adresa IP jednoducho preložená a správa je poslaná ďalej. Takýto NAT sa nijakým spôsobom nestará o prideľovanie adries rôznym pripojeniam počas reálnej prevádzky a všetky zobrazenia sú vykonávané jednoduchou tabuľkou.
- Dynamický, kde sú jednotlivé adresy prideľované klientov v privátnej siete podľa potreby. V prípade, že už dané pripojenie nie je potrebné, je toto zobrazenie zahodené. Jedná sa najmä o sieťové adresy, ale podobný princíp platí aj pre transportné adresy. Rôzne NAT implementácie môžu tieto adresy prideľovať rôznym spôsobom.

NAT je možné rozdeliť ďalej na niekoľko kategórií, napríklad aj podľa toho, či je možné komunikovať obojstranne, alebo len jedným smerom:

- Tradičný NAT – Jedná sa o NAT, ktorý je jednosmerný. Znamená to, že klient z privátnej siete dokáže vytvoriť spojenie s vonkajším svetom. Adresy, ktoré sú prideľované sú v rámci verejnej siete jedinečné, na rozdiel od privátnych adries, ktoré sú jedinečné len v rámci privátnej siete. Tento NAT sa taktiež delí na viacero častí:
  - Základný NAT, anglicky Basic NAT – NAT, ktorý si pre preklad vyhradí niekoľko sieťových adries z verejnej siete a následne ich prideľuje jednotlivým klientom, ktorí si žiadajú komunikovať s hostiteľom vo verejnej sieti. Datagram IP je upravený tak, že všetky polia týkajúce sa sieťovej adresy sú prepísané podľa toho, na akú verejnú adresu sa daná privátna adresa prekladá.
  - Preklad sieťových adries s prekladom portov, anglicky Network Address Port Translation, skrátene NAPT – Tento druh NAT-u prekladá ako sieťové adresy, tak aj čísla portov. Je teda možné to, že sa viacerým klientom v lokálnej sieti pridelí jedna verejná sieťová adresa a navzájom sa ich spojenia odlišujú číslami portov použitých transportných protokolov. V datagrame IP nie je teda upravovaná len sieťová adresa, ale aj číslo portu. V práci cielime práve na tento druh NAT-u, konkrétne na jeho dynamickú variantu.

- Obojsmerný NAT, anglicky Bi-directional NAT – Jednotlivé spojenia môžu byť nadviazané ako z privátnej, tak aj z verejnej siete. Privátne adresy dostanú pridelenú verejnú adresu staticky alebo dynamicky, či už je spojenie vytvárané zvnútra alebo zvonku. Doménové mená hostiteľov v privátnej aj verejnej sieti sa považujú za globálne unikátne a na ich preklad sa používa systém DNS.

- Dvojitý NAT, anglicky Twice NAT – NAT, pri ktorom dochádza k prekladu ako zdrojovej, tak aj cieľovej adresy naraz. V predošlých prípadoch šlo vždy o preklad len jednej z týchto adries.

- Multihomed NAT – NAT, pri kotorom sa pri pripájaní z privátnej siete použije niekoľko pripojení, pričom každé má svoje vlastné zariadenie s NAT-om, napríklad smerovač, ktorý má NAT implementovaný a zapnutý. Toto pomáha ako pri rozkladaní komunikácie a odľahčení tak jednotlivých NAT zariadení, tak aj ako záloha v prípade, že by niektoré z NAT zariadení vypadlo.

4.5.1 Problém pri pripájaní sa do privátnej siete z verejnej siete

Pre našu NP2PS peer-to-peer sieť je potrebné, aby bola schopná stále, v akomkoľvek časovom okamihu, komunikovať s ktorýmkoľvek peerov, ktorý je jej súčasťou. Vyplýva to z podstaty peer-to-peer siete, o ktorej prevádzku a správne fungovanie sa stará každý peer, ktorý je do nej zapojený. Musíme si teda nejakým spôsobom pamätať, kde sa daný peer nachádza a ako ho kontaktovať. Zavedením protokolu NP2PS sme problém formátu správ už vyriešili, ešte sa však potrebujeme pripojiť k druhému peerovi, s ktorým chceme komunikovať. Vieme ho identifikovať pomocou jeho transportnej adresy, ktorú nám zdelil, keď sa prvý krát pripojil do siete. Predpokladáme ďalej, že ak chce peer patriť do NP2PS siete, musí sa aspoň raz pripojiť v sieti k peerovi, ktorý neleží v privátnej sieti za NAT-om. Bez toho totiž ani nemáme ako vedieť, že sa daný peer vôbec do siete pripojil.

Pokus o pripojenie na túto transportnú adresu však môže zlyhať. Peer, ku ktorému sa snažíme pripojiť totiž môže ležať za tradičným NAT-om s prekladom portov, ktorý mu dynamicky pridelil transportnú adresu pri pripájaní sa do siete, ale po odpojení sa zo siete bolo toto zobrazenie zahodené a už neexistuje. Danú kombináciu už môže dokonca používať iná služba, s ktorou si neželáme komunikovať.

Nestačí si teda len pamätať jeho transportnú adresu, z ktorej prebiehala kedysi daný peer komunikoval. Už vôbec si nemôžeme pamätať jeho privátnu transportnú adresu, pretože tá v kontexte verejnej siete nedáva zmysel. Musíme teda

nejakým spôsobom peera, ležiaceho za NAT-om, presvedčiť, aby opäť otvoril spojenie z peer-to-peer sieťou a teda aby bolo možné s ním následne komunikovať. Predpokladáme, že peer sa komunikácie v sieti chce účastniť, len nevie, že s ním chce nejaký ďalší peer komunikovať.

Pripojenie sa k peerovi za NAT-om

Nadviazať spojenie s peerom sa teda nemusí prostredníctvom bežných prostriedkov podariť. Existuje niekoľko prístupov a techník, ktoré nám umožnia nadviazať spojenie s peerom, a umožnia nám tak obísť jeho NAT.

UDP hole punching

Jedna z možností je použiť techniku zvanú UDP hole punching. Uvažujeme jej popis z [12]. Princíp fungovania spočíva v tom, že dvaja peeri, ktorí chcú navzájom nadviazať medzi sebou spojenie, si pošlú navzájom v jeden moment správy. Odoslaním správ sa pri oboch peeroch vytvorí potrebné NAT mapovanie, ktoré peeri potrebujú k tomu, aby mohli spolu komunikovať.

Obaja peeri sa však musia zhodnúť v tom, že chcú spolu komunikovať. Rozdeľme si ich na peera A a peera B. Na tento účel slúži tzv. zoznamovací server, prostredníctvom ktorého peer A oznámi peerovi B, že s ním chce začať komunikovať. Obaja peeri sú k tomuto serveru vopred pripojení. Peer B následne, keď mu zoznamovací server oznámi zámer peera A priamo komunikovať, odošle datagram na transportnú adresu peera A, skrz ktorú je pripojený k zoznamovaciemu serveru. Peer A spraví to isté. Transportné adresy môžu peeri získať napríklad zo zoznamovacieho servera.

Ak by sa ktorýkoľvek z peerov pokúsil nadviazať spojenie priamo, bez toho, aby sa druhý peer pokúsil o to isté, mohlo by dôjsť k tomu, že sa toto pripojenie odmietne. Niektoré implementácie NAT-u zaregistrujú, že sa jedná o iného odosielateľa, napríklad peera A, než je zoznamovací server. Ak však peerovi A odošleme správu na transportnú adresu, z ktorej nám posiela správu on, vytvoríme v NAT-e potrebné mapovanie, ktoré umožní UDP datagramu prejsť až k peerovi B.

Problémom tejto metódy je to, že je potrebné správne načasovať odoslanie náprotivných správ daným peerom. NAT totiž, ak do nejakého času nebude po sieti tiecť komunikácia, potrebné zobrazenie zruší. Ďalšia vec, na ktorú je potrebné si dať pozor a ktorá do istej miery skomplikuje návrh a vývoj aplikácie, ktorá túto metódu používa je tá, že komunikácia musí prebiehať stále z tej istej privátnej transportnej adresy, alebo je aspoň potrebné nejakým spôsobom zariadiť, aby sa tieto adresy preložili na rovnakú verejnú transportnú adresu. To už ale vyžaduje zásah do nastavenia daného NAT-u a k tomu nemusíme mať prístup. Správy, ktoré teda prichádzajú na socket môžu byť buď to správy zo zoznamovacieho servera,

alebo správy z druhého peera. Aplikácia musí medzi týmito správami rozonávať, na čo je potrebná ďalšia logika aplikácie, ktorú je potrebné naimplementovať.

Táto technika zlyhá, ak sa peeri nachádzajú za rovnakým NAT-om. V tom prípade totiž NAT nemusí dovoliť vytvoriť mapovanie na klienta, ktorý sa nachádza v rovnakej privátnej sieti, ako odosielateľ. V tom prípade je možné ešte odoslať skrz zoznamovací server ako adresu verejnú, tak aj privátnu tak, ako ju detegujú daní peeri. Správy sa pošlú na obidve adresy a tá, ktorá príde ako prvá, bude považovaná za správnu a bude sa používať počas komunikácie, pričom sa k NAT-u ani nedostane. Toto riešenie však nebude fungovať, ak sa peeri nachádzajú za niekoľkými rôznymi NAT-mi, pretože na lokálnu adresu sa pripojiť nepodarí a peeri nemajú ako zistiť aké sú ich adresy medzi dvomi NAT-mi, po ktorých vedie cesta smerom do zoznamovacieho servera. Dokonca, ak na hranici leží len jeden spoločný NAT pre oboch peerov, tak sa musíme spoliehať, že nám tento NAT umožní komunikovať s hostiteľom, ktorý sa nachádza v rovnakej lokálnej sieti, ako my.

TCP hole punching

Podobne ako v UDP je definovaný aj TCP hole punching. Táto metóda sa spolieha na techniku zvanú „Simultaneous TCP open“, kde sa namiesto tradičnej trojcestnej dohody, anglicky three-way handshake, vykoná spojenie tak, že ak jeden z peerov pošle SYN paket, a krátko na to dostane SYN paket od peera, ktorého kontaktoval, tak sa spojenie považuje za otvorené, hneď po odoslaní potvrdzujúcich ACK paketov.

Nanešťastie nie je táto technika vždy implementovaná správne [12] a v konečnom dôsledku sa môže stať, že sa bude musieť použiť presmerovanie cez tretiu stranu, pretože podľa štatistík z [13] len asi na 60% celkových testovaných NAT zariadení fungovala táto metóda správne.

Náš prístup

Jedným možným riešením tohto problému by mohlo byť použitie protokolu UDP na obchádzanie NAT-u pomocou metódy UDP hole punching a následne použiť vytvorené zobrazenie v NAT-e použiť na komunikáciu takým spôsobom, že by sme zabalili TCP pakety do UDP datagramov a posielali ich týmto spôsobom. Takto sme však stratili všetky výhody protokolu TCP, ak by vôbec takéto zaobalenie bolo možné, vzhľadom na veľmi odlišnú náturu týchto dvoch transportných protokolov. TCP hole punching sme sa vzhľadom na nízku úspešnosť a problematickú implementáciu najmä kvôli správnemu načasovaniu odoslania SYN paketov rozhodli vynechať z implementácie.

Nás prístup je teda nasledovný. Majme opäť peera A a peera B. Majme zoznamovací server, ktorý je v kontexte našej peer-to-peer NP2PS siete šéfredaktor daných novín, ku ktorým sieť náleží. Technicky by ale bolo možné použiť hocijaký STUN server v celej NP2PS sieti. Peer A si vyžiada od peera B jeho transportnú adresu pre NP2PS od zoznamovacieho servera použitím protokolu STUN. Následne sa pokúsi na túto transportnú adresu pripojiť. Jedná sa o tú istú adresu, ktorú peer B oznámi šéfredaktorovi v rámci protokolu NP2PS počas alokácie.

V prípade, že tento mechanizmus zlyhá a spojenie je buď to odmietnuté, alebo spojeniu vyprší čas, presmerujeme danú správu skrz STUN server, prostredníctvom ktorého sme získali informáciu, že daný peer môže disponovať daným zdrojom, o ktorý máme záujem. STUN server správu peera A následne vezme a prepošle ju peerovi B. Peer B sa dozvie, že správa pochádza od peera A z jeho unikátneho identifikátora, ktorý je obsiahnutý v NP2PS správe, ktorá je do zabalená do správy protokolu STUN.

4.5.2 Protokol STUN

Session Traversal Utilities for NAT, skrátene STUN, je protokol na aplikačnej vrstve, ktorý slúži ako pomocný nástroj pri obchádzaní NAT-u. Protokol teda sám o sebe nie je kompletné riešením problému obchádzania NAT-u, ale je možné ho použiť a rozšíriť podľa potreby aplikácie a v našom prípade aj protokolu NP2PS.

Protokol STUN podporuje nasledujúce transportné protokoly: TCP, UDP, TLS-over-TCP a DTLS-over-UDP. My budeme používať rovnako ako pri NP2PS transportný protokol TCP.

STUN terminológia

Na komunikácii sa v protokole STUN podieľajú STUN server a STUN klient, ich spoločné pomenovanie znie „STUN agenti“. V sekcii o STUN-e budeme, kde to nebude spôsobovať problémy s nejednoznačnosťou, používať len výrazy „klient“, „server“ a „agent“.

STUN klient je agent, ktorý sa pripája a posiela požiadavky serveru a prijíma na ne odpovede. Posielať a prijímať je možné taktiež indikácie.

STUN server je agent, ku ktorému sa pripája klient a ktorý od neho prijíma požiadavky a naspäť posiela odpovede. Rovnako ako klient, aj server musí prijímať a posielať indikácie.

STUN indikácia je špeciálny typ správy, na ktorú nie je očakávaná, ani žiadaná odpoveď.

Reflexívna transportná adresa, resp. mapovaná adresa, je tá transportná adresa, ktorá náleží STUN klientovi z pohľadu siete, v ktorej leží STUN server. Na reflexívnu transportnú adresu sa pokúšajú pripojiť peeri pri metóde UDP hole

punching. Podobne sa na reflexívnu transportnú adresu pripájajú NP2PS peeri, keď sa pokúšajú komunikovať s iným peerom.

STUN správa

STUN agenti spolu komunikujú posielaním tzv. STUN správ. Každá správa má svoj typ, ktorý sa delí na metódu a na triedu. Za správou môžu následne nasledovať ľubovoľný počet jej argumentov, aj nulový. Celkový počet argumentov je však obmedzený dĺžkou správy, ktorá je obmedzená na 64 KiB. Toto obmedzenie je teda taktiež platné pre všetky správy, ktoré budeme pomocou protokolu STUN presmerovávať. Možným riešením tohto problému by bolo rozdeliť danú správu na niekoľko kúskov a následne ju odoslať samostatne, kde by sa na druhej strane poskladala zas dohromady. Druhým možným riešením je takúto správu odmietnuť odoslať.

Správy sa delia na štyri triedy.

- Požiadavka, anglicky request.
- Odpoveď s úspechom, anglicky success response.
- Odpoveď s chybou, anglicky error response.
- Indikácia, anglicky indication.

Požiadavka je správa, ktorú posiela klient serveru. Ten mu ako odpoveď odošle buď to odpoveď s úspechom, v prípade, že sa mu požiadavke podarilo vyhovieť, alebo odpoveď s neúspechom v prípade, že sa mu požiadavke vyhovieť nepodarilo. Klient môže taktiež poslať indikáciu, na tú však už nemusí byť odpoveď a ak áno, tak opäť vo forme indikácie. Dokumentácia protokolu STUN hovorí, že sa odporúča pri komunikácii zahrnúť atribút Software. Atribút Software slúži na identifikáciu softvéru, ktorý používa agent posielajúci správu. Tento atribút nemá na chod protokolu žiaden vplyv a slúži len na účely ladenia. Vzhľadom na to, že protokol používame len interne a s vlastnými atribútmi a metódami, sme sa rozhodli atribút nezahrnúť, pretože softvér, ktorý správu posiela je vždy ten istý.

Počet tried je fixný, čo vyplýva z ich kódovania v STUN type, kde sú vyhradené na tento účel len dva bity, dávajúc dokopy štyri možnosti.

Správy sa ďalej delia na metódy. Dokumentácia k STUN protokolu definuje len jednu metódu a to metódu Binding.

STUN atribúty

Dokumentácia protokolu STUN definuje niekoľko atribútov. Uvedieme len tie, ktoré používame v našej implementácii STUN a NP2PS protokolov.

- ErrorCode – Atribút obsahujúci kód chyby, ktorý bude pripojený k odpovedi s chybou
- UnknownAttributes – Zoznam atribútov, ktorým daný STUN server alebo klient nerozumie, aj keď by im rozumieť mal. Podľa dokumentácie sa jedná o takzvané „comprehension-required“ atribúty.
- Data – Atribút nesúci dáta pri presmerovaní sieťovej komunikácie skrz server. Definovaný v protokole TURN.
- XorMappedAddress – Základný atribút, ktorý umožňuje prenos transportnej adresy pre sieťové protokoly IPv4 a IPv6 a transportné protokoly TCP a UDP.
- XorRelayedAddress – Rovnako ako XorMappedAddress.
- PublicIdentifier – Slúži na prenos verejného identifikátora NP2PS peera. Definovaný pre účely protokolu NP2PS.
- RelayedPublicIdentifier – Rovnako ako PublicIdentifier. Definovaný pre účely protokolu NP2PS.
- PublicKey – Slúži na prenos verejného kľúča RSA. Definovaný pre účely protokolu NP2PS.

Metóda Binding

Metóda Binding je základná metóda definovaná v STUN protokole. Táto metóda má za úlohu oznámiť klientovi, ktorý jej požiadavku odoslal, aká je jeho reflexívna transportná adresa. Jedná sa o tú adresu, ktorá je uvedená v hlavičke IP datagramu a TCP paketu, ktorý server prijal. Ide teda o adresu, ktorá vznikla ako výsledok posledného prekladu v NAT-e, ktorý je z pohľadu servera k nemu najbližší. Za predpokladu, že sa daný server nachádza vo verejnej sieti, tak odpoveď, ktorú odošle klientovi, bude obsahovať jeho verejnú adresu, ktorú mu pridelil posledný NAT na hranici s verejnou sieťou. Vždy uvažujeme, že NAT prekladá buď medzi dvomi privátnymi sieťami, alebo medzi privátnou a verejnou sieťou.

Reflexívna adresa, ktorú týmto klient získa, môže byť ďalej posunutá napríklad zoznamovaciemu serveru, ktorý ju potom môže oznámiť ďalším klientom, ktorí by o to mali záujem. Rovnaký princíp používame pri zisťovaní reflexívnych adries peerov v NP2PS sieti. Na tento účel si však potrebujeme rozšíriť STUN protokol o niekoľko ďalších metód a atribútov.

Metóda Binding nebudeme v tejto práci viac do podrobna opisovať. Pre náš protokol ani implementáciu nemá žiaden význam a nepoužívame ju tam. Je tu

uvedená z toho dôvodou, aby bol demonštrovaný základý účel protokolu STUN a preto, lebo niektoré jej vlastnosti zdieľajú ďalšie metódy.

4.5.3 Vlastné rozšírenie protokolu STUN

Vzhľadom na to, že protokol STUN slúži len ako pomocný nástroj na obchádzanie NAT-u, potrebovali sme ho rozšíriť o ďalšie metódy a atribúty tak, aby vyhovoval nášmu prístupu k tejto problematike. Kódy jednotlivých atribútov a hodnôt boli vybrané takým spôsobom, aby nenastávala kolízia so žiadnym z existujúcich a definovaných atribútov pre protokol STUN a ďalšie protokoly, ktoré ho rozširujú. Ako referenciu sme v tomto prípade používali IANA databázu pre STUN parametre, teda pre metódy, atribúty a chybové kódy.

Metódy pridávajú STUN serveru možnosť alokovať v dátovej štruktúre `IpMap` záznamy o reflexívnych adresách peerov a následne v nej, na žiadosti ďalších peerov, vyhľadávať. Rozšírenie taktiež pridáva možnosť presmerovávať komunikáciu z jedného peera k druhému v prípade, že sa im z akéhokoľvek dôvodu nepodarí skontaktovať, najmä to však platí pre NAT. Ak by sme si však vzali bránu firewall, ktorá má pridané pravidlo povoľujúce komunikáciu len z nejakej konkrétnej sieťovej adresy IP, tak vieme na danej adrese pustiť náš rozšírený STUN server a bude možné posielať správy aj peerom za touto bránou firewall.

Metóda Allocate

Metóda Allocate slúži na „alokáciu“ transportnej adresy klienta na serveri. To znamená, že si danú adresu klienta server uloží do dátovej štruktúry `IpMap`, do ktorej potom neskôr bude pristupovať. Podobne ako v metóde Binding, aj pri metóde Allocate sa naalokuje tá adresa, ktorú ako výsledok prekladu vydal posledný NAT z pohľadu klienta. Je to teda ten NAT, ktorý leží na hranici s verejnou sieťou.

Dôvod existencie metódy Allocate je ten, aby sa z bežného STUN servera stal tzv. zoznamovací server. Ďalší klienti si budú môcť od tohto servera vyžiadať transportnú adresu nejakého iného klienta a pokúsiť sa s ním komunikovať priamo.

Metóda Allocate používa pri formulácii požiadavky nasledujúce atribúty:

- PublicIdentifier – Obsahuje hodnoty verejenáho identifikátora peera NP2PS siete, ktorý žiada o alokáciu.
- PublicKey – Obsahuje hodnotu verejného kľúča RSA peera NP2PS siete, ktorá žiada o alokáciu.

V implementácii je pri vytváraní požiadavky taktiež obsiahnutý atribút Lifetime. Jedná sa o atribút, ktorý má hovoriť o dĺžke životnosti alokácie. Vzhľadom na to, že peer má stále otvorené a aktívne spojenie do siete, tak nie je tento atribút potrebný a slúži pre budúce rozšírenie.

Metóda Allocate používa pri formulácii úspešnej odpovede nasledujúce atribúty:

- XorMappedAddress – Slúži ako náhrada za metódu Binding. Server použije atribút na to, aby oznámil klientovi jeho reflexívnu transportnú adresu.
- PublicIdentifier – Verejný identifikátor STUN servera, ktorý formuluje odpoveď. Klient tento údaj použije na to, aby daný server pridal ako STUN server do svojej databázy.

Metóda Identify

Metóda Identify slúži na vyžiadanie reflexívnej transportnej adresy peera, ktorému chceme poslať správu. Server nám následne odpovie kladne, s priloženou reflexívnou transportnou adresou vo forme atribútov správy, alebo záporne, ak server danú transportnú adresu nemá naalokovanú. Peer, ktorý chce, aby bolo možné jeho transportnú adresu si u nejakého zo serverov vyžiadať pomocou metódy Identify, si musí najprv vyžiadať u daného servera alokáciu použitím metódy Allocate.

Metóda Identify používa pri formulovaní požiadavky nasledujúce atribúty:

- PublicIdentifier – Obsahuje verejný identifikátor peera, ktorého reflexívnu transportnú adresu žiadame. Atribút nie je povinný a v prípade, že sa v správe nenachádza, žiadame verejný identifikátor servera, ktorému správu posielame.

Metóda Identify používa pri formulovaní kladnej odpovede nasledujúce atribúty:

- XorRelayedAddress – Žiadaná reflexívna transportná adresa.
- PublicIdentifier – Verejný identifikátor peera, o ktorého transportnú adresu sme žiadali.
- PublicKey – Verejný kľúč RSA peera, o ktorého transportnú adresu sme žiadali.
- RelayedPublicIdentifier – Verejný identifikátor servera, ktorý formuluje odpoveď. Slúži rovnako ako v metóde Allocate.

Metóda Identify používa pri formulovaní negatívnej odpovede nasledujúce atribúty:

- PublicIdentifierAttribute – Verejný identifikátor peera, o ktorého reflexívnu transportnú adresu sme žiadali.

Metóda Send

Metóda Send slúži v prípade, že sa peerovi A, ktorý sa snaží pripojiť k peerovi B nepodarí pripojiť priamo. V tomto prípade sa obsah správy, ktorý peer A chce poslať peerovi B zabalí do atribútu Data metódy Send. Server následne metódu Send prijme a prepošle peerovi, ktorému je správa určená.

Pri formulovaní požiadavky metódy Send sa používajú nasledujúce atribúty:

- PublicIdentifier – Verejný identifikátor odosielateľa správy.
- RelayedPublicIdentifier – Verejný identifikátor prijímateľa správy.
- Data – Správa, ktorá mala byť pôvodne odoslaná prijímateľovi priamo. V našom prípade sa vždy bude jednať o NP2PS správu.

Pri formulovaní indikácie metódy Send sa používajú rovnaké atribúty, ako pri formulovaní požiadavky. Počas spracovania sa ale zmení RelayedPublicIdentifier na verejný identifikátor STUN servera, cez ktorého je správa presmerovaná. Účelom tejto úpravy je to, aby mohol prijímateľ presmerovať svoju odpoveď cez rovnaký STUN server, z akého prišla.

Metóda Send nemá definované odpovede. Dochádza tu teda k tomu, že sa odošle požiadavka, na ktorú ale neexistuje odpoveď. V skutočnosti sa pri metóde Send nejedná o požiadavku v slova zmysle, ako tomu je pri ostatných metódach, ale jedná sa o indikáciu, ktorej ako trieda je uvedená požiadavka. Toto netradičné a pre definíciu protokolu STUN aj nesprávne riešenie je z toho dôvodu, že význam jednotlivých atribútov sa po prijatí požiadavky a formulovaní indikácie zmení a ak by obidve správy mali triedu indikácia, mohlo by dôjsť k zámene týchto dvoch, svojím spôsobom odlišných správ. Z technického hľadiska by k tejto zámene nedošlo, pretože Send indikáciu, ktorú by prijal server by vždy bola určená na presmerovanie a Send indikácia, ktorú by prijal klient by vždy bola už presmerovaná správa, ktorá dorazila do svojej destinácie. Avšak počas prenosu z jedného peera na server a z neho na peera druhého, by došlo k zmene významov atribútov. Existovali by teda dve správy s rovnakou triedou a metódou, zatiaľ čo by ale ich význam bol iný.

Ideálnym riešením tejto situácie by bolo pridanie ďalšej metódy, ktorá by oddelila tieto dva významy. Naše riešenie je ale funkčné, jednoduché a postačujúce

na náš účel, zatiaľ čo pridanie ďalšej metódy by vytváralo dojem, že sa jedná o úplne odlišné správy, ktoré spolu nijakým spôsobom nesúvisia. V konečnom dôsledku sme teda uprednostnili riešenie, ktoré je najbližšie ľudskému chápaniu, aj za cenu toho, že porušíme zásady, ktoré definuje protokol STUN. Avšak vzhľadom na to, že sa STUN servery používané v NP2PS sieti nebudú aj tak nedajú použiť na iný účel, nám to nepríde ako problém.

4.5.4 Protokol TURN

Traversal Using Relays around NAT je protokol ktorý definuje obchádzanie NAT-u spôsobom presmerovania toku údajov cez TURN server. Pôvodne sme zvažovali použitie tohto protokolu, avšak nakoniec sa ukázalo, že je na náš účel nevhodný.

Protokol TURN totiž slúži na to, aby mohol TURN klient vytvoriť pre TURN peerov verejnú transportnú adresu, na ktorej s ním budú môcť komunikovať. Táto adresa sa tiež nazýva obchádzková adresa. Práve na tento účel sa používa metóda Allocate, ktorej názov sme prebrali do nášho riešenia. TURN však už nemá metódu Identify a obchádzkovú adresu TURN klienta si TURN peer musí zistiť spôsobom, ktorý už tento protokol nešpecifikuje a tento peer ani nemusí vedieť, že s ním chce daný TURN klient komunikovať. Toto je jeden z dôvodov, prečo nám protokol TURN prišiel nevhodný na používanie. Ak by sme pridali do protokolu TURN metódu Identify, čo by bolo možné, dostali by sme ten istý systém, aký sme vytvorili my pre náš účel s tým rozdielom, že by bol všeobecnejší pre použitie s ľubovoľným TURN klientom a nie len s NP2PS klientmi. Táto vlastnosť je ale pre nás zbytočná a pre NP2PS nadsieť nepotrebná.

Mechanizmus obchádzkových adries je však rovnako náročný ak nie náročnejší ako náš mechanizmus s verejnými identifikátormi. TURN server totiž po prijatí správy musí zistiť, na ktorú z obchádzkových adries správa prišla a následne v nejakej dátovej štruktúre vyhľadať, kam sa má ďalej správa preposlať. V našom prípade sa deje to isté s verejným identifikátorom, ktorý je však jedno 64-bitové bezznamienkové číslo, zatiaľ čo v prípade TURN-u sa v závislosti od použitého rozhrania na sieťovanie môže jednať o netriviálne dátové štruktúry.

TURN server preposiela dáta od TURN klienta TURN peerovi tak, že TURN klient uvedie reflexívnu transportnú adresu TURN peera, ktorému chce správu poslať. Opačne to funguje tak, že TURN peer odošle správu na obchádzkovú adresu TURN klienta. V našom prípade, kde si každý NP2PS peer aj tak naalokuje svoju reflexívnu transportnú adresu, takto ale vzniknú dva spôsoby odosielania správ, pričom jeden z nich je zjavne zbytočný.

4.6 Modelová sieť

Implementáciu protokolu bolo potrebné počas jej vývoja nejakým spôsobom testovať a vzhľadom na to, že sa jedná o sieťový protokol, bolo potrebné nejakým spôsobom simulovať sieť, na ktorej by bolo možné vyskúšať správnosť implementácie jednotlivých funkcionalít protokolu. Najjednoduchším spôsobom ako docieliť takmer reálneho prostredia a zároveň nepoužívať veľké množstvo hardvéru je použiť virtuálnu sieť. Takáto sieť umožní ľubovoľne meniť topológiu siete pridávaním a odoberaním kontajnerov. Ako kontajnerizačný engine sme použili Docker, vzhľadom na jeho popularitu a jednoduchú dostupnosť veľkého množstva materiálu, či už vo forme oficiálnej dokumentácie, alebo rôznych príspevkoch na fórach.

Na správu jednotlivých kontajnerov sme použili súbor docker-compose.yml v ktorom sme každého peera nakonfigurovali, ako samostatný service. Ďalej sme Vytvorili dve siete, taktiež v docker-compose.yml, aby sme mohli vyskúšať, ako bude vyzerať komunikácia dvoch peerov v dvoch rôznych sieťach. Títo medzi sebou totiž priamo nedokážu komunikovať, čím nasimulujeme príplad, kedy sa nie je možné k peerovi priamo pripojiť, ako tomu býva v prípade brán firewall, alebo NAT-ujúcich routerov.

Jedná sa o siete typu „bridge“. Hostiteľ je pripojený ako do oboch sietí.

Testované boli štruktúry, v ktorých prebiehala komunikácia v rámci jednej siete s jedným šéfredaktorom, teda STUN serverom, ale aj viacerými šéfredaktormi, teda viacerými STUN servermi, pričom sa jeden zo STUN serverov vypol, aby sme overili, či bude použitý ďalší z nich. Testované bolo aj opätovné pripájanie peera do siete a samozrejme, základná komunikácia a vlastnosti protokolu, ako pridanie nového redaktora, či komunikácia prebieha s daným redaktorom a podobne.

Program bol skompilovaný v samostatnom kontajneri a ostatné kontajnery, ktoré tvorili úlohu peerov používali tú istú verziu programu.

Prístup k jednotlivým peerom bol prostredníctvom programu na vzdialený prístup zvaného Xpra.

4.7 Spojenie protokolov NP2PS a STUN

Vzhľadom na to, že protokol STUN je len pomocný nástroj na obchádzanie NAT-u a sám o sebe je len jednoduchý protokol na výmenu správ, by bolo možné ho naimplementovať ako súčasť protokolu NP2PS. Jednotlivé metódy by sa prepísali do správ a atribúty by boli voliteľnými dátovými položkami Google Protocol buffers správ. Dokonca by ich bolo možné implementovať použitím kľúčového slova `repeated` a mať ich hlavičky uložené v podstate vo forme poľa. Následne

by každá z daných hlavičiek atribútov obsahovala `oneof` a zoznam atribútov, z ktorých by práve jeden bol použitý.

Dôvod, prečo sa od tejto myšlienky upustilo je ten, že by sme museli mať vždy spustenú celú aplikáciu, aj keby sme nevyužívali služieb NP2PS. Takto by bolo možné oddeliť od seba STUN a NP2PS časť, pričom by mohla bežať len jedna z nich. Takto by sme mohli nasadzovať do siete viaceré STUN servery, pričom by sme mohli odstrániť funkcionalitu NP2PS. V našej implementácii potrebujeme uložiť reflexívnu transportnú adresu pre NP2PS časť pre jednotlivých peerov za účelom metódy Identify, takže túto metódu by tieto STUN servery nepodporovali. Podpora by ale bola možná pre presmerovanie komunikácie cez STUN server použitím metódy Send, čo je náročnejšie ako na výpočtové zdroje, tak na kapacitu siete a teda práve to by sme dokázali odľahčiť.

Tento nedostatok by sa ale dal vyriešiť pridaním novej metódy, ktorá by umožnila peerom oznámiť STUN serverom svoje reflexívne transportné adresy pre protokol NP2PS.

Kapitola 5

Technická dokumentácia

Program je rozdelený do niekoľkých tried a pracuje s externými knižnicami Crypto++ a Google Protocol buffers a s platformou Qt. Dôvod výberu platformy Qt je ten, že ponúka knižnice ako na vývoj užívateľského rozhrania, tak na vývoj sieťovania. Vďaka tejto vlastnosti bolo možné rozumným spôsobom navzájom pospájať jednotlivé sokety, prvky užívateľského rozhrania a logiku aplikácie, ktorá ležala medzi nimi.

5.1 Doxygen

Súčasťou práce je aj dokumentácia programu vygenerovaná frameworkom Doxygen.

5.2 Správa pamäte na halde

Všetky objekty žijúce na halde sú spravované tzv. „smart“ ukazovateľmi a nemusíme teda riešiť uvoľnenie ich pamäte. Neplatí to však pre všetky objekty a tie z platformy Qt sú na halde alokované použitím kľúčového slova `new`. Uvoľnenie ich pamäte ďalej rieši slot `deleteLater` definovaná na triede `QObject`, na ktorý napojíme všetky dané objekty. Jedná sa o objekty typu `QTcpSocket` alebo položky v rôznych zoznamoch a stromoch, ako napríklad `QTreeWidgetItem`.

5.3 Signály a sloty v Qt

Každý objekt, ktorý dedí od objektu triedy `QObject` má možnosť využívať mechanizmus signálov a slotov. Signály sú funkcie, ktoré je možné vyvolať použitím

kľúčového slova `emit`. Signály sú len definované a programátor ich nedeklaruje, sú deklarované frameworkom. Signály sa dajú pomocou statickej funkcie `QObject::connect`, predaním ukazovateľov na objekty a na ich funkcie pripojiť na sloty. Sloty sú opäť funkcie, ktoré definuje a aj deklaruje programátor. Obsah slotu nie je nijakým spôsobom obmedzený a jedná sa o štandardnú C++ funkciu.

Typová bezpečnosť je zabezpečená tým, že signatúra signálu aj slotu sú rovnaké, alebo signatúra slotu obsahuje menej argumentov, ako signatúra signálu. Pri vysielaní signálu sa predajú vo forme argumentov funkcie, ktorá signál reprezentuje dáta, ktoré sú neskôr predané slotu, resp. slotom, ktoré sú na daný signál napojené.

Implementáciu programu si vieme rozdeliť, podľa toho, ako vedú signály a sloty, na tri časti. Užívateľskú časť, logickú časť a sieťovú časť. Pod užívateľskú časť patrí všetko to, čo patrí k užívateľským rozhraniam, teda ich objekty a ich funkcie, spravidla sú tieto objekty potomkami triedy `QWidget`. Pod sieťovú časť patria objekty tried `QTcpSocket` a `QTcpServer`. Pod logickú časť patria všetky ostatné objekty, ktoré nepatria do predošlých dvoch častí.

V našej implementácii teda používame tento mechanizmus na komunikáciu medzi týmito troma časťami. Framework Qt tento mechanizmus používa na ohlásenie zmeny stavu svojich objektov a teda nám prišlo jeho používanie vhodné a jednoduché. Možné je tento mechanizmus použiť napríklad pri práci so soketmi, kde pri úspešnom pripojení sa vyšle socket signál `hostConnected`. Ak by sme to chceli implementovať bez použitia signálov a slotov, museli by sme napríklad v cykle čakať, pokiaľ by sa stav soketu nezmenil na `ConnectedState`. Počas toho by sme ale buď to museli vytvoriť nové vlákno, alebo by sme blokovali užívateľské rozhranie. Problém by ale nastal, ak by sme chceli zistiť, do akého chybového stavu sa soket dostal. Enumerácia `SocketError` triedy `QAbstractSocket` totiž neobsahuje žiadnu položku, ktorá by označovala bezchybný stav. Ak skúsime na sokete, o ktorom vieme, že sa v chybnom stave určite nenachádza zavolať metódu `error`, zistíme, že sa vráti hodnota `UnknownSocketError`. Problém je ale ten, že ak by sme sa na tento fakt spoľahli a daný výsledok by sme vždy brali ako bezchybný soket, tak si nemôžeme byť v žiadnom prípade istí, či sa nejedná o nejakú nezdokumentovanú chybu a či skutočne je soket v použiteľnom stave. Podobných funkcií existuje viac a správnym experimentovaním a zdĺhavým štúdiom dokumentácie Qt by sa nám najskôr podarilo nejakým spôsobom správne detegovať chybný soket aj pomocou jeho funkcií, bez použitia signálov a slotov. Trieda `QTcpSocket` však v prípade, že nastane chyba, vyšle signál `errorOccured` a v tom prípade vieme s istotou, že sa soket dostal do chybového stavu a nemusíme to pracne zisťovať pomocou množstva rôznych funkcií, ktoré nemusia dokopy pokryť všetky scenáre, kedy by sa soket mohol nachádzať v chybovom stave.

Ďalej sa signály a sloty používajú na komunikáciu v rámci užívateľskej časti medzi jednotlivými oknami aplikácie a medzi logickou časťou a výstupom z

rôznych sekundárnych okien, teda všetkých, čo nie sú hlavným oknom aplikácie.

Signály a sloty sa definujú v hlavičkovom súbore kľúčovými slovami `signals` a `slots`.

5.4 Dôležité triedy

Ďalej opíšeme niekoľko tried, ktoré tvoria zásadnú rolu v logickej časti implementácie a celého programu.

5.4.1 Trieda `Peer`

Trieda `Peer` je základnou triedou programu, ktorá spravuje NP2PS Peera počas celého behu programu. Nachádzajú sa tu databázy novín a ich článkov, čitateľov rôznych článkov, zoznam redaktorov novín a podobne. Ďalej obsahuje trieda základné informácie o peerovi, ako sú jeho verejný identifikátor a v prípade šéfredaktora aj názov a verejný identifikátor novín. Trieda `Peer` sa nachádza na vrchole hierarchie logickej časti aplikácie a obsahuje inštancie všetkých ostatných tried ako sú `Networking`, `Article`, `Margin` alebo `NewspaperEntry`. Nie všetky tieto triedy sú uložené priamo, ale ležia v rôznych dátových štruktúrach triedy `Peer`.

Použité dátové štruktúry

Samozrejme, existujú triedy, ktorých inštancií máme viac ako jednu. Príkladom je trieda `Article`, ktorá reprezentuje jeden novinový článok, ako jeho text, tak aj jeho metadáta. Ako dátové štruktúry sa najčastejšie používajú ich neusporiadané verzie týchto dátových štruktúr, pretože majú priemernú konštantnú časovú zložitosť tak, ako je to pre dátové štruktúry ktoré používajú hashovanie typické. Kľúče sú jednoduchého skalárneho typu a tak teda uvažujeme, že je tento hash jednoduchý na výpočet. Vzhľadom na veľkosť novín, na akú sú protokol aj jeho implementácia stavané, by nemalo dochádzať k prílišnému zaplneniu jednotlivých dátových štruktúr a teda aj potenciálneho preplneniu ich priehradok. Usporiadanie, ktoré by nám poskytli tradičnejšie verzie týchto dátových štruktúr nepotrebujeme.

Asi najdôležitejšou dátovou štruktúrou je zoznam novín `news_`, obsahujúci objekty triedy `NewspaperEntry` a v rámci nich sa nachádzajúci zoznam článkov daných novín `_articles`. Ďalej sa používa dátová štruktúra `readers_` na ukladanie zoznamov čitateľov jednotlivých článkov a dátová štruktúra `journalists_` obsahujúca množinu verejných identifikátorov peerov, ktorí sú registrovaní v daných novinách ako redaktori, resp. novinári.

Jednotlivé dátové štruktúry sú prístupné buď to cez ukazovateľ alebo referenciu, ako sa to robí v prípade komunikáciou triedy `Networking` s niektorými dátovými položkami a štruktúrami triedy `Peer`, alebo je to prostredníctvom funkcií a metód, ktoré trieda `Peer` prezentuje ako svoje API. Tieto funkcie vedia napríklad hľadať v zozname článkov všetkých novín dohromady, alebo len pre jedny noviny samostatne.

Dôležitou úlohou triedy `Peer` je spracovanie NP2PS správ. V momente, kedy trieda `Networking` prijme a spracuje prijatú správu, ju predá triede `Peer`. Tá už následne vykoná potrebné kroky, aké od peera NP2PS siete vyžaduje NP2PS protokol.

5.4.2 Trieda `Networking`

Táto trieda obsahuje všetky dátové položky a štruktúry na to, aby mohol peer komunikovať po sieti s ostatnými peermi a to či už použitím soketu priamo, dohľadaním jeho reflexívnej transportnej adresy v dátovej štruktúre a následné vytvorenie nového pripojenia na danú adresu, alebo komunikovaním so STUN serverom a komunikáciou pomocou presmerovávania.

Trieda taktiež spravuje všetky štyri sieťové časti peera, teda STUN klienta aj server a potom NP2PS peera, ako prijímateľa tak aj odosielateľa. Tie sú buď to inicializované v konštruktore, alebo v samostatnej metóde, ktorá je volaná zvonku.

Trieda obsahuje niekoľko surových ukazovateľov, taktiež nazývaných ako observery, na kontajnery, ktoré sa nachádzajú v triede `Peer` a sú inicializované potom, čo sú inicializované v triede `Peer`. Tieto sa tu nachádzajú z toho dôvodu, aby do nich bol prístup aj z ďalších tried, ktoré trieda `Networking` má ako dátové položky, ako napríklad `StunServer`, či už priamo, alebo prostredníctvom mechanizmu signálov a slotov.

Dôležitou súčasťou triedy `Networking` je objekt typu `IpMap`. Práve tento objekt obsahuje všetky potrebné údaje, aby sa mohlo vykonať pripojenie po sieti, alebo aby sa mohlo použiť už jestvujúce pripojenie k ďalšiemu peerovi. `IpMap` teda disponuje dátovou štruktúrou, ktorá zobrazuje verejné identifikátory na ich transportné adresy a poprípade, ak sa jedná o peera, s ktorým už bolo naviazané spojenie, tak aj STUN a NP2PS `QTcpSocket` sokety, cez ktoré toto naviazanie spojenia prebehlo. Na zaobalenie týchto dát do rozumnej formy sa používa štruktúra `IpWrapper`, ktorá má taktiež definované rôzne funkcie a metódy na prístup k jednotlivým dátovým položkám, ktoré zaobaľuje.

Po tom, čo NP2PS soket pre danú komunikáciu vyšle signál `readyRead` je správa prečítaná, dešifrovaná, v prípade asymetricky šifrovaného symetrického kľúča sa skontroluje digitálny podpis. Ďalej sa správa deserializuje na Google Protocol buffers správu a predá sa triede `Peer` na spracovanie pomocou signálu

new_message_received.

Výber QByteArray ako objektu, do ktorého sa budú zapisovať dáta, nám prišiel vhodný preto, pretože vieme jednoduchým spôsobom zapísať „surové“ dáta tak, aby sme nemuseli riešiť, či ich daný objekt nejakým spôsobom nezmení, ako tomu môže byť napríklad pri objekte QString. Ďalej sa tento objekt dá jednoducho odoslať prostredníctvom QTcpSocket.

Serializácia a deserializácia správ

Správa Google Protocol buffers je serializovaná a deserializovaná použitím metód SerializeToString a ParseFromString. Následne je tento std::string prevedený na objekt typu QByteArray a následne je celý obsah zapisovaný, resp. čítaný pomocou triedy QDataStream, ktorá, napojením na ďalší QByteArray doň zapíše dáta a jeho obsah je možné rovno odoslať použitím soketu do siete. Pred správu sú ešte pred odoslaním predpísané metadáta, ktoré nakoniec spolu tvoria NP2PS metasprávu.

Pre STUN správy je situácia veľmi podobná s tým rozdielom, že správy nie sú formátu Google Protocol buffers, takže ich serializácia prebieha priamo zápisom do QDataStream.

5.4.3 Potomkovia triedy StunMessageAttribute

Jednotlivé atribúty, ktoré je možné pripájať k STUN správam sú v implementácii reprezentované triedami, ktoré všetky dedia od jedného spoločného predka, triedy StunMessageAttribute. Táto trieda definuje virtuálne metódy read_stun_attribute a write_stun_attribute, ktoré prečítajú alebo zapíšu správu do objektu triedy QDataStream. Tieto metódy sú neskôr prepísané ako virtuálne metódy ich potomkami tak, aby správne serializovali, prípadne deserializovali ich dátové položky.

Takto nám postačí ukazovateľ na triedu StunMessageAttribute a vieme prechádzať zaradom jednotlivé atribúty a volať správne funkcie podľa toho, aký objekt v skutočnosti leží na mieste, kde ukazuje ukazovateľ.

Dôvod, prečo sme takto riešili len atribúty a nie aj hlavičky, je ten, že by nakoniec vzniklo príliš veľké množstvo tried, v ktorom by sa dalo ťažko orientovať. Tento princíp by sme vedeli využiť teoreticky v prípade, že by sme mali jednu virtuálnu funkciu process(...) predka, ktorá by bola schopná spracovať akúkoľvek správu podľa toho, aká správa by to v skutočnosti bola. Problémom tejto metódy je ale to, že metóda process(...) by pre rôzne kombinácie STUN triedy a STUN metódy potrebovala iné argumenty a dávala iné výsledky. Tento stav by sa ešte dal obísť teoreticky vytvorením jednej veľkej štruktúry, ktorej obsah by tvorili ukazovatele na objekty, ktoré by tvorili zjednotenie argumentov

a výstupov všetkých kombinácií STUN metódy a STUN triedy. Toto riešenie však taktiež nie je ideálne, pretože vkladanie jednotlivých údajov a získavanie výsledkov by aj tak vytvorili potrebu mať nejaký `switch` blok, v ktorom by sa naplnili, resp. prečítali podľa typu a metódy. Nakoniec je tento `switch` možné použiť už v základe a vkladať argumenty do adekvátnych funkcií rovno. To potom ale nemôžeme mať jednu spoločnú funkciu pre všetky triedy a museli by sme mať v predkovi definovanú každú variantu tejto funkcie. To ale môžme rovno zadefinovať samostatne alebo to nezadefinovať vôbec a spracovávať jednotlivé správy v iných objektoch, čo je prístup, ktorý sme zvolili.

5.4.4 Trieda `StunServer` a `StunClient`

STUN server aj klient sú v logickej časti aplikácie reprezentovaní triedami `StunServer`, resp. `StunClient`.

`StunServer`

Trieda `StunServer` má za úlohu prijímať STUN požiadavky a spracovávať ich. Nové pripojenie je spracované pripojením slotu `new_connection` na signál `QTcpServer::newConnection`. Následne sa pomocou metódy `nextPendingConnection` získa `QTcpSocket`, prostredníctvom ktorého prebieha ďalšia komunikácia.

Správy sú spracováva metóda `reply` prostredníctvom metód `read_message_header`, pomocou ktorej sa prečíta hlavička STUN správy a `read_message_attributes`, pomocou ktorej sa prečítajú atribúty správy. Následne sa podľa triedy a metódy STUN správy sformuluje odpoveď a odošle sa po tom istom sokete, po ktorom prišla.

Každá kombinácia STUN triedy a metódy má svoju vlastnú funkciu na spracovanie a neskôr na vytvorenie odpovede, ktorých názvy majú formát `process_[trieda]_[metoda]`, resp. `create_[trieda]_[metoda]`. Pri vytváraní do týchto funkcií vstupuje ako argument STUN správa, ktorá sa upraví podľa požiadaviek danej kombinácie triedy a metódy. Pri spracovaní do týchto funkcií vstupujú aspoň dve STUN správy, jedna pôvodná a jedna nová. Funkcie, ktoré spracovávajú požiadavku často volajú funkcie na vytvorenie odpovede.

`StunClient`

Trieda `StunClient` má za úlohu vytvárať nové spojenia a odosielať STUN požiadavky, resp. prijímať STUN odpovede. Trieda má zadefinované funkcie, ktoré vytvoria požadovaný typ správy. Následne sa daná správa pošle použitím funkcie

send_stun_message, kde je prijímajúci identifikovaný NP2PS verejným identifikátorom, alebo použitím funkcie send_stun_message_transport_address, v ktorej je prijímateľ identifikovaný pomocou jeho reflexívnej transportnej adresy.

Pripojenie prebieha pomocou signálov a slotov, kde sa pripojí klientov slot hostConnected na signál connected triedy AbstractSocket. Podobným spôsobom sa napoja chybové signály.

Avšak medzi volaním funkcie connect a pripojením je potrebné niekde uschovať všetky dáta, ktoré je potrebné odoslať. Preto má trieda niekoľko dátových položiek, ktoré obsahujú dáta čakajúce na odoslanie:

- address_waiting_to_connect – sieťová adresa servera, ku ktorému sa pripájame
- port_waiting_to_connect – port servera, ku ktorému sa pripájame
- header_waiting_to_connect – hlavička správy, ktorú chceme poslať
- connecting_to – verejný identifikátor servera, ku ktorému sa pripájame

Po úspešnom pripojení sa tieto dátové položky zmažú.

5.4.5 Trieda PeerReceiver a PeerSender

Trieda PeerReceiver reprezentuje príjemcu NP2PS správ, teda v klient-server modeli by sme hovorili o serveri. Príjemca však na rozdiel od StunServer triedy nespracováva prijaté správy a slúži len na ich prijímanie, dešifrovanie, deserializovanie na Google Protocol buffers správu a následné odovzdanie triede Peer, ktorá už dané správy spracuje a sformuluje na ne odpoveď.

Trieda Peer predá svoje odpovede triede PeerSender, teda odosielateľovi. Odosielateľ správu vezme, skontroluje prítomnosť symetrického kľúča pre prijímateľa, ktorý ak neexistuje, tak je po výmene verejných kľúčov RSA vygenerovaný, odoslaný protistrane a následne použitý na zašifrovanie správy.

Pri nových pripojeniach, či už pri ich vytváraní alebo prijímaní sa používajú rovnaké princípy použitím signálov a slotov ako pri triedach StunServer a StunClient. To znamená, že pri pripájaní sa na príjemcu je potrebné použiť pomocné dátové položky triedy, na uloženie dát na dobu, pokiaľ dôjde k pripojeniu.

5.5 Reprezentácia správ

5.5.1 Kódovanie správ v Google Protocol buffers

Na serializáciu Google Protocol buffers správ sa používa takzvaný „binary wire format“, kde sa jedná o binárne dáta určené pre wire protokol. Z tohto formátu

nie je možné presne špecifikovať jednotlivé prvky a teda k zisteniu, čo vlastne kóduje je potrebná externá špecifikácia. Znalosť tohto kódovania nie je potrebná pre bežné použitie Google Protocol buffers.

Na kódovanie čísel sa používa tzv. „varint“. Táto metóda spočíva v tom, že je jedno číslo zakódované pomocou ľubovoľného počtu bajtov. MSb pre každý bajt určuje, či už číslo skončilo, alebo nie, teda či ešte nasleduje nejaký ďalší bajt. Bit s hodnotou 1 znamená, že ešte nasleduje ďalší bajt, bit s hodnotou 0 naopak. Napríklad, takto by bol zakódované číslo s hodnotou 1:

```
0000 0001
```

MSb je nulový a teda žiaden ďalší bajt už nenasleduje. Situácia sa zmení pri kódovaní čísla s hodnotou 300:

```
1010 1100  0000 0010
```

Tu vidíme, že že bajty sú usporiadané v tzv. „little-endian“ poradí a teda ako prvý je bajt s najmenšou pozičnou hodnotou. Prvý bajt má hodnotu MSb rovnú 1 a teda vieme, že nasleduje ešte jeden bajt. Dekódovanie tejto informácie už nie je také zložité a funguje nasledovne:

1. odstránia sa MSb bity, tým akurát stratíme už nepotrebnú informáciu o tom, kedy reťazec bajtov končí.

   ```
      1010 1100  0000 0010
   ->  010 1100   000 0010
   ```

2. Prehodíme poradie tak, aby bajt s najväčšou pozičnou hodnotou bol na začiatku a bajt s najmenšou pozičnou hodnotou na konci.

   ```
      010 1100  000 0010
   -> 000 0010  010 1100
   ```

3. Spojíme tieto bajty, teda z tohto reťazca bitov vyrobíme bajty tak, aby obsahovali bity s práve tou hodnotou, akú máme po prehodení pôvodných bajtov. Dostaneme teda:

   ```
   0000 0001  0010 1100
   ```

4. Z čoho dostávame $2^8 + 2^5 + 2^3 + 2^2 = 256 + 32 + 8 + 4 = 300$.

Kódovanie správy

Správy sú kódované pomocou dvojíc kľúč-hodnota, pričom tieto dvojice sú serializované do bajtového toku. Aby sa zachovala spätná kompatibilita pre staršie parsery, je potrebné povedať, akú dĺžku daná dátová položka má, teda aby bolo

možné ho preskočiť. Kľúč sa teda rozšíri na dve hodnoty: poradie danej položky v rámci správy a jej dĺžka, kódované vo „wire“ formáte. Vznikne tzv. tag. Každá dátová položka v rámci Google Protocol buffers správy má svoje poradie a teda nejaký unikátny identifikátor, ale len v kontexte jednej správy.

Na začiatku dátového toku sa nachádza varint, ktorý kóduje kľúč. Spodné tri bity tohto varint kľúča (bez MSb) označujú, o aký typ dátovej položky ide (viď. Tabuľka 5.1) a ktoré je jej poradie v rámci správy. Poradie získame bitovým posunutím o tri bity doprava.

Čísla, ktoré nie sú kódované pomocou varint metódy (`fixed64`, `fixed32` alebo aj `double`), sú uložené v „little-endian“ poradí bajtov, tak, ako je pri reprezentácii týchto čísel bežné.

Textové reťazce a vnorené správy

Typ 2 kóduje textové reťazce. Znamená to, že hodnotou, ktorá je kódovaná je najprv dĺžka a potom nasledujú bajty tvoriace daný textový reťazec. Použité kódovanie je UTF-8.

Vnorené správy sú kódované úplne rovnako ako textové reťazce. Hodnotou je samotná správa „rekurzívne“ zakódovaná.

Opakované a voliteľné dátové položky

Opakované dátové položky sú vhodné v prípade, že chceme serializovať napríklad pole, alebo iný dátový typ, u ktorého dáva zmysel, že má viacero hodnôt.

Opakované položky sa buď to vôbec do správy nedostanú, ak nemajú žiadnu hodnotu, alebo sú kódované podobne, ako textové reťazce. Najprv je kľúč, potom dĺžka a potom jednotlivé položky tak, ako by boli zakódované normálne s tým rozdielom, že nemá každá z nich samostatný kľúč, ten je pre všetky spoločný.

Typ	Význam	Použitie
0	Varint	Číslo s premenlivou dĺžkou, boolean, vymenovacie typy.
1	64-bit	Číslo s fixnou dĺžkou, 64-bitové.
2	Dĺžkou delimitované	Textové a bajtové reťazce, vnorené správy, opakujúce sa položky.
	...	(vyradené s používania)
5	32-bit	Číslo s fixnou dĺžkou, 32-bitové.

Tabuľka 5.1 Typy v Google Protocol buffers.

Serializácia a poradie dátových položiek

Na poradí dátových položiek v rámci správy nezáleží a môže byť teda ľubovoľné. Počas serializácie nie je nijakým spôsobom zaručené v akom poradí sa dátové položky zakódujú a teda táto vlastnosť závisí od implementácie k implementácii. Akýkoľvek parser musí zvládnuť hocijaké poradie, v akom mohli byť dátové položky zakódované a serializované.

5.5.2 Kódovanie STUN správ

Správy protokolu STUN sú založené na princípe hlavičky a atribútov. Atribútov môže byť ľubovoľný počet, ktorý vieme zistiť postupným odčítavaním ich veľkostí od celkovej veľkosti správy. Atribúty sú zarovnané vždy na celé štyri bajty pridaním prázdnych bajtov na koniec atribútu, pričom ale toto zarovnanie sa nepočíta do veľkosti atribútu.

Hlavička

Hlavička STUN správy sa skladá z piatich častí, pričom začíname na „least significant“ bite „least significant“ bajtu:

- Dvojica nulových bitov. Tieto bity sú vždy nulové. Dĺžka: 2 bity.
- Typ STUN správy. Dĺžka: 14 bitov.
- Veľkosť STUN správy. Dĺžka: 16 bitov. Počíta len veľkosti atribútov. Toto pole a jeho dĺžka obmedzujú celkovú možnú veľkosť odosielanej správy pri presmerovaní NP2PS správy.
- Pole „Magic Cookie“. Dĺžka: 32 bitov. Slúži na detekciu STUN paketu od ostatných správ.
- Identifikátor transakcie. Dĺžka: 96 bitov. Generovaný náhodne, ale správy v rovnakej transakcii sa v jeho hodnote zhodujú.

Typ hlavičky obsahuje triedu a metódu STUN správy. Ak rozdelíme toto 14 bitové číslo na jednotlivé bity, kde prvý bit je „least significant“ bit celého čísla, tak šiesty a desiaty bit tvoria triedu STUN správy a ostatné bity tvoria metódu STUN správy.

Atribút

Atribút STUN správy je tvorený trojicou typ, veľkosť, hodnota:

- Typ. Dĺžka: 16 bitov.
- Veľkosť. Dĺžka: 16 bitov. Hovorí o dĺžke hodnoty, pričom sa nepočítajú zarovnávacie bajty a ani 2 bajty, ktoré zaberú typ a veľkosť dokopy.
- Hodnota. Dĺžka: premenlivá.

Jednotlivé atribúty môžu byť usporiadané v ľubovoľnom poradí. Ak sa jeden atribút nachádza v správe dvakrát, STUN agent musí spracovať aspoň prvý z nich. To, či spracuje aj ostatné výskyty alebo ich nijakým spôsobom nepoužije, závisí od implementácie STUN agenta. V našej implementácii sa opakované atribúty nevyskytujú.

STUN atribút musí byť zarovnaný na veľkosť štyroch bajtov. Vyriešené je to tým spôsobom, že dané atribúty pri písaní do siete zavolajú funkciu, ktorá ich správne zarovná. Zarovnávanie prebieha pripájaním nulových bajtov na koniec atribútu tak dlho, pokiaľ jeho dĺžka nie je deliteľná štyrmi. Analogicky to funguje pri čítaní zo siete, kde sa ale jedná o čítanie, nie zápis bajtov.

5.5.3 Vlastné STUN atribúty

Protokol STUN nám neposkytuje všetky nástroje ktoré potrebujeme na implementáciu nášho spôsobu obchádzania NAT-u. Preto sme sa rozhodli ho v súlade s jeho špecifikáciou rozšíriť. Do protokolu sme pridali ako nové metódy, tak aj nové atribúty. Zatiaľ čo metódy sa medzi sebou líšia len číslom metódy, atribúty sa líšia usporiadaním dát vo svojich hodnotách. Prišlo nám teda vhodné rozpísať, aké tieto hodnoty sú.

Atribút PublicIdentifier

Prvý atribút, ktorý sme zadefinovali slúži na prenos verejného identifikátora peera. Štruktúra atribútu je jednoduchá:

- Verejný identifikátor. Dĺžka: 64 bitov. V implementácii zastúpený číslom s typom `quint64`.

Obsahuje len jednu 8 bajtovú položku a tou je práve daný verejný identifikátor. Atribút je týmto zarovnaný na násobky štyroch bajtov a teda ďalšie zarovnávanie nie je potrebné.

Hodnota typu atribútu: `0x001F`.

Atribút RelayedPublicIdentifier

Atribút analogický s atribútom PublicIdentifier. Význam jeho použitia je ten, že niekedy je potrebné v STUN správe odoslať dvojicu verejných identifikátorov a potrebujeme ich nejakým spôsobom rozlíšiť. Druhý z týchto identifikátorov však vždy nejakým spôsobom súvisí s presmerovaním správy, alebo so STUN serverom. Preto nám prišlo prehľadnejšie pridať ďalší atribút a adekvátne ho pomenovať, než pridávať nejaký ďalší identifikátor do obsahu atribútu, ktorý by určoval jeho význam, alebo niečo podobné.

Hodnota typu atribútu: `0x002C`.

Atribút PublicKey

Atribút PublicKey slúži na prenos verejného kľúča. Jeho reprezentácia už nie je taká jednoduchá, pretože musí rátať s kľúčmi variabilnej dĺžky.

- Verejný kľúč. Dĺžka: premenlivá. V implementácii zastúpený textovým reťazcom s typom `std::string`.

Dĺžka reťazca je zakomponovaná do dĺžky atribútu a nie je na ňu vyhradená samostatná hodnota. Vzhľadom na variabilnú dĺžku kľúča je potrebné atribút pri písaní do siete správne zarovnávať. Naopak pri čítaní zo siete je zas potrebné toto zarovnanie prečítať, aby nedošlo k poškodeniu nasledujúcich atribútov.

Hodnota typu atribútu: `0x002B`.

Atribút Data

Dátový atribút, ktorý sa používa ako obal na NP2PS správu, ak ju chceme poslať presmerovaním cez STUN server. Podobne ako PublicKey, aj tento atribút má variabilnú dĺžku, ktorá je uložená v hlavičke atribútu.

- Dáta určené na presmerovanie. Dĺžka: premenlivá. V implementácii zastúpený poľom bajtov s typom `QByteArray`.

Rovnako ako pri PublicKey je potrebné správne atribút pri zápise do siete zarovnať a po prečítaní zarovnanie prečítať.

Hodnota typu atribútu: `0x0013`.

Kapitola 6

Užívateľská dokumentácia

6.1 Užívateľské rozhranie

Na uľahčenie interakcie s programom bolo vytvorené grafické užívateľské rozhranie na platforme Qt.

6.2 Prehľad jednotlivých okien aplikácie

6.2.1 Okno Newspaper P2P sharing

Po spustení programu užívateľ uvidí okno s nadpisom „Newspaper P2P sharing“. Je to hlavné okno, cez ktoré interagujeme s programom. Okno je rozdelené na niekoľko častí, pričom jednotlivé časti tvoria tzv. karty, „taby“.

Karta Newspaper browser

Táto karta slúži na prezeranie obsahu novín. V ľavej hornej časti máme tlačidlá, ktoré slúžia na pridávanie novín a to tlačidlá „Add newspaper“ a „Add newspaper from file“. Tlačidlo Add newspaper potvorí dialógové okno Add newspaper, kde je možné pridať noviny ručne, prostredníctvom vypísania ich IP adries alebo doménového mena. Tlačidlo Add newspaper from file otvorí dialógové okno, kde je možné zvoliť súbor obsahujúci metadáta novín, ku ktorým sa chceme pripojiť.

Pod týmito tlačidlami sa nachádza zoznam novín, ktoré užívateľ sleduje. Po kliknutí na položku tohto zoznamu sa automaticky stiahne zoznam článkov daných novín. Vedľa tohto zoznamu sa nachádza tlačidlo „News list“ ktoré si od pripojených peerov vyžiada zoznam novín, ktoré sledujú. Tieto sú následne k dispozícii, ak by ich užívateľ chcel začať sledovať, prostredníctvom okna „Add newspapers from list“.

Pod zoznamom novín sa nachádza riadok obsahujúci informatívny text: „Search article headings (regex support)“. Do tohto riadka je možné zadať výraz a následne, stlačením klávesy Enter, alebo tlačidla „Search“, ktoré sa nachádza napravo od daného riadka, sa v zozname článkov, ktorý sa nachádza pod týmto riadkov na vyhľadávanie zobrazia len tie články, ktoré sa s daným výrazom zhodujú. Jedná sa o výraz typu „regex“. Výraz ignoruje veľké a malé písmená.

Pod riadkom na filtráciu podľa regexových výrazov na nachádza zoznam článkov, ktorý sa mení podľa toho, ktoré noviny sú zvolené v zozname novín. V tomto zozname sa nachádzajú všetky články, ktoré má peer lokálne stiahnuté a aj ktoré nemá. Po kliknutí na článok sa článok automaticky stiahne zo siete, ak sa nájde peer, ktorý ho má k dispozícii.

Pod zoznamom článkov sa nachádza zoznam obsahujúci zjednotenie všetkých kategórií, ktoré niektorý z článkov má. Po zvolení kategórie sa v zozname článkov zobrazia len články, ktoré danej kategórii vyhovujú.

Pod zoznamom kategórií sa nachádza dvojica tlačidiel, ktorá určuje, podľa ktorého času, či podľa času úpravy, alebo vytvorenia, budú články radené.

Napravo od zoznamu článkov, zoznamu novín a pod. sa nachádza plocha, kde sa zobrazí článok potom, čo je naň kliknuté v zozname článkov a je stiahnutý zo siete.

Chief editor

Táto karta obsahuje nástroje potrebné na prácu šéfredaktora. Úplne na vrchu karty sa nachádza pole, kde je možné vyplniť názov novín v prípade, že daný peer ešte noviny založené nemá. Napravo od tohto poľa sa nachádza potvrdzovacie tlačidlo. Obidve tieto položky nie sú aktívne, ak peer už noviny založené má.

Následne sa pod daným poľom nachádza dvojica tlačidiel, najprv ale povedzme, čo sa nachádza pod týmito tlačidlami. Jedná sa o zoznam peerov, ktorí žiadajú o prácu redaktora v daných novinách. V prípade, že je zvolený jeden z týchto peerov v tomto zozname, sprístupnia sa tlačidlá „Confirm journalist“, ktoré prácu redaktora danému peerovi sprístupní a „Deny journalist“, ktoré ju odmietne. Úplne naspodku sa nachádza tlačidlo, ktoré vyexportuje noviny do súboru, ktorý je možne neskôr použiť na pripájanie k nim. Nad týmto tlačidlom sa nachádza pole, do ktorého je možné vpísať IP adresu, ktorá sa má použiť pri pripájaní sa k do súboru exportovaným novinám.

Journalist

Táto karta obsahuje nástroja na prácu redaktora. Uprostred sa nachádza zoznam článkov, ktoré daný redaktor pre dané noviny napísal. Tieto noviny je možné zvoliť v poli na týmto zoznamom. Následne sa vedľa zoznamu novín nachádzajú

štyri tlačidlá, ktoré, v tomto poradí: otvoria okno „Edit article“ na úpravu článku, otvoria dialógové okno „Add article“ a pridanie nového článku, a vyžiadajú od šéfredaktora daných zvolených novín prácu redaktora.

Settings

Karta settings obsahuje nastavenia aplikácie, ktoré sa používajú počas jej behu. Časť „Newspaper settings“ je samostatná pre jednotlivé noviny, pričom je možné si medzi novinami vyberať v zozname, ktorý sa nachádza na jej vrchu. Pod tým sa nachádza časť o peerovi, ktorá je pre všekty noviny spoločná. Tieto nastavenia samostatne prechádzať nebudeme.

Napravo sa nachádza druhá časť, ktorá obsahuje možnosť nastaviť porty protokolov ako NP2PS, tak aj STUN. Voľba sa potvrdí stlačením tlačidla „Set ports“.

Tlačidlá „IC“, „Allocate“, „Gossip“ a „Save“ (to vpravo) slúžia na demonštračné účely.

6.2.2 Okno New peer

Okno „New peer“ sa zobrazí automaticky po prvom spustení aplikácie.

- Pole „Name“ do ktorého je možné zadať názov peera.
- Zaškrtávacie políčko „Create newspaper“, ktoré založí nové noviny pre práve vytváraného peera a umožní ich meno napísať do poľa s názvom „Newspaper name“.
- Pole „Newspaper name“. Do poľa nie je možné písať, pokiaľ sa nezaškrtne zaškrtávacie políčko „Create newspaper“. Do poľa sa napíše názov novín, ktoré chceme založiť.
- Dvojica tlačidiel „OK“ a „Cancel“. Tlačidlo „OK“ vykoná zmeny, ktoré užívateľ zaniesol do tohto okna. Tlačidlo „Cancel“ všetky zmeny zahodí.

Po kliknutí na tlačidlo „OK“ sa okno uzavrie a tlačidlo „Add news“ sa aktivuje, umožňujúc odoberanie novín iných peerov. V prípade, že sme sa rozhodli si noviny založiť, tak sa záznam o našich novinách zobrazí v Strome novín.

V prípade, že bolo stlačené tlačidlo „OK“ a založili sme noviny, sa tlačidlo „New peer“ v primárnom okne aplikácie deaktivuje, nakoľko už neponúka žiadne ďalšie možnosti konfigurácie.

6.2.3 Okno Add newspaper

Okno „Add newspaper“ zobrazíme stlačením tlačidla „Add news“ v karte Newspaper browser.

Pole „Name“ slúži na zadanie názvu novín, ktorý bude neskôr zobrazený v stĺpci „Name“ v strome novín. Nemusí to byť presne názov novín, ktoré chceme odoberať, ale môže sa jednať o ľubovoľný názov, aký nám vyhovuje.

Pod poľom „Name“ sa nachádza pole „IP“, do ktorého je potrebné zapísať sieťovú adresu šéfredaktora novín, ktoré chceme odoberať. Na tejto sieťovej adrese bude prebiehať komunikácia s danými novinami. Ďalej sa tam nachádzajú polia na zadávanie čísla portov pre protokoly NP2PS a STUN, tak ako sú v aplikácii označené.

Úplne dole sa nachádzajú tlačidlá „OK“ a „Cancel“. To prvé z nich potvrdí výber, pridá zvolené noviny, zmaže obsah polí pre budúce použitie a zatvorí okno. To druhé zahodí zmeny a zatvorí okno.

6.2.4 Okno Add article

Okno „Add article“ sa zobrazí potom, čo klikneme na tlačidlo „Add article“ nachádzajúce sa na karte „Journalist“. Okno sa skladá z niekoľkých častí, prejdeme ich zhora nadol a zľava doprava.

Ako prvý je rozbaľovací zoznam obsahujúci zoznam odoberaných novín plus našich novín, ak sme nejaké založili. V tomto zozname si vieme vybrať, ku ktorým novinám sa článok má pridať. Poznamenajme, že program nevie, v ktorých novinách sme redaktormi, takže nám to umožní poslať článok do ktorýchkoľvek novín. Posiela sa však len hlavička článku, jeho obsah ostáva uložený lokálne u redaktora, ktorý to napísal. V prípade, že nemáme právo prispievať do niektorých novín a predsa tam článok pošleme, dané noviny článok nezverejnia.

Pod týmto rozbaľovacím zoznamom sa nachádza pole „Category“, do ktorého môžme napísať meno kategórie, do ktorej chceme článok zaradiť. Zaradenie potvrdíme stlačením tlačidla „Add category“. Následne sa pridá daná kategória do zoznamu „Categories for article“, ktorý sa nachádza pod poľom „Category“.

V tomto zozname vidíme všetky kategórie, do ktorých sme sa rozhodli článok pridať. Ak by sme pridali nejakú kategóriu dvakrát, tak nakoniec do nej bude článok zaradený len raz.

Ak by sme si rozmysleli zaradenie článku do niektorej z kategórií, vieme ho z nej odstrániť zvolením danej kategórie v zozname a stlačením tlačidla „Remove category“.

Ak sme už ukončili pridávanie článku do kategórií, potvrdíme to použitím tlačidla „Accept“, ktorá sa nachádza na vľavo dole. Ak by sme sa nakoniec rozhodli článok nepridávať, použijeme tlačidlo „Cancel“, ktoré sa nachádza napravo od

tlačidla „Accept“. Obidve tlačidlá toto okno zatvoria a zmažú obsah použitých polí pre ďalšie použitie.

6.2.5 Okno Add margin

Okno „Add margin“ obsahuje len dve polia. Zhora prvé má názov „Type“ a slúži na určenie typu marginálie. Typ marginálie si vieme ľubovoľne zvoliť.

Pod ním sa nachádza druhé pole „Contents“, ktoré slúži na zadanie obsahu marginálie. Obsah marginálie môže byť ľubovoľný, ale mal by byť ideálne čo najstručnejší.

Nakoniec sa v spodnej časti okna nachádza dvojica tlačidiel „OK“ a „Cancel“. Po stlačení tlačidla „OK“ sa marginália pridá k článku, ktorý bol v tej chvíli zvolený v Strome novín. Tlačidlo „Cancel“ zahodí vykonané zmeny. Obidve tlačidlá zmažú obsahy polí tak, aby boli pripravené na ďalšie použitie.

6.2.6 Okno Edit article

Okno „Edit article“ pozostáva z dvoch hlavných častí. Napravo sa nachádza veľké pole, kde sa zobrazí aktuálna verzia článku, bez formátovania a naľavo sa nachádza ďalšie pole, kde sa zobrazuje už sformátovaný obsah poľa naľavo. Obsah napravo je aktualizovaný automaticky po úprave obsahu poľa naľavo.

Dole sa nachádza dvojica tlačidiel, „Save“, ktoré úpravy článok uloží a „Cancel“, ktoré úpravy zruší. Všetky úpravi sú po uložení automaticky odosielané do siete peerom.

6.2.7 Okno Add newspapers from list

Okno je rozdelené na ľavú a pravú časť. Časť pravá (s nadpisom „Available“) obsahuje noviny, ktoré peer získal od ostatných peerov a ktoré môže začať sledovať. Časť pravá (s nadpisom „Subscribed“) obsahuje zoznam novín, ktoré peer sleduje. Ak chce peer sledovať niektoré noviny z pravého zoznamu, môže kliknúť na veľké tlačidlo so šípkou uprostred, ktoré pridá noviny do zoznamu sledovaných novín.

6.3 Často vykonávané postupy

6.3.1 Vytvorenie nového peera a novín

Nového peera je možné vytvoriť pri prvom spustení aplikácie. Otvorí sa okno, do ktorého je možné zadať meno peera a meno novín, ak ich chceme založiť. Ak chceme založiť noviny, je potrebné najprv zaškrtnúť zaškrtávacie políčko „Create

newspaper“, ktoré nám následne umožní zadať názov nami zakladaných novín. Meno peera, ktorého vytvárame, zadáme do poľa „Name“ a názov novín, ktoré spolu s ním zakladáme, zadáme do poľa „Newspaper name“.

Nakoniec náš výber potvrdíme stlačením tlačidla „OK“.

Ak by sme sa najprv rozhodli noviny nezaložiť a neskôr by sme sa rozhodli toto rozhodnutie zmeniť, tak nám stačí ich názov zadať do poľa „Name your news here...“ na karte „Chief editor“ a kliknúť na tlačidlo napravo („Start news“).

6.3.2 Pridanie článku do novín

Na karte „Journalist“ nájdeme tri tlačidlá. To prostredné z nich, tlačidlo „Add article“, otvorí okno na pridanie nového článku do novín.

V tomto novom okne si úplne navrchu vieme vybrať, do ktorých novín chceme článok napísať. Následne vieme do poľa „Category“ napísať kategóriu, do ktorej chceme článok pridať a potvrdiť tento fakt tlačidlom „Add category“. Tento proces vkladania kategórií opakujeme tak dlho, pokiaľ nezadáme všetky kategórie, do ktorých chceme článok pridať. V zozname s názvom „Categories for article“ vidíme zoznam kategórií, do ktorých sme článok zaradili.

Náš výber potvrdíme stlačením tlačidla „Accept“.

6.3.3 Pridanie cudzích novín

Ak chceme pridať cudzie noviny, musíme najprv kliknúť na tlačidlo „Add news“, ktoré sa nachádza v hornej časti karty „Newspaper browser“. Po kliknutí na toto tlačidlo sa nám otvorí okno, obsahujúce štyri polia, „Name“, „IP“, „NP2PS port“ a „STUN port“. Do poľa „Name“ napíšeme ľubovoľný názov, aký chceme, aby sa zobrazil v Strome novín v stĺpci „Name“, ideálne tak, aby sme v budúcnosti vedeli medzi sebou viacero novín rozlíšiť.

Do poľa „IP“ zadáme sieťovú adresu novín, ku ktorým sa chceme pripojiť. Zadaním nesprávnej adresy sa k novinám nepripojíme.

Do polí s portami zadáme adekvátne čísla portov, ak to požadujeme, inak sa použijú ich štandardné hodnoty.

Následne už len potvrdíme svoj výber stlačením tlačidla „OK“.

Záver

Na záver zhodnotíme výsledok našej práce, do akej miery sa nám podarilo splniť požiadavky, ktoré sme si zadali v úvode a bližšie určili v špecifikácii zadania, prípadne sa pokúsime zanalyzovať ako by sa naše riešenie mohlo dať ešte dodatočne vylepšiť.

Eliminácia nevýhod modelu klient-server

Spomínané nevýhody klient-server modelu sme znížili alebo úplne odstránili implementovaním peer-to-peer siete. Odstránili sme existenciu jednej alebo viacerých menších, ale stále silných entít a nahradili sme ich veľkým množstvom malých entít, ktoré každá riadia samé seba a zároveň sa podieľajú na prevádzke siete ako celku. Každá entita teda rozhoduje, čo spraví so svojimi dátami, ale nerozhoduje o tom, čo sa spraví s dátami iných, čím ostáva zachovaná prístupnosť peera k jeho dátam nezávisle od integrity ostatných peerov v celej sieti, teda jeho vlastné údaje má peer sám a nie server v dátovom centre. Takto sme rozdelili úlohu servera, ktorý by v normálnom klient-server modeli spravoval každého klienta, medzi všetkých peerov v celej sieti. Pomocou replikácie dát v sieti sa nám podarilo rozložiť nároky na úložisko do celej siete, čo prakticky, pokiaľ bude sieť rásť úmerne s jej obsahom, značne znižuje požiadavky na veľkosť úložného priestoru pre články u jednotlivých šéfredaktorov.

Stále však ostáva závislosť na šéfredaktoroch novín. Títo v podstate tvoria akési oporné piliere celej siete a bez nich by najskôr nebola vôbec funkčná. Avšak nasadzovanie šéfredaktorov je v porovnaní s nasadzovaním dátových centier jednoduché, rýchle a nenáročné na pamäť, či výkon výpočtových jednotiek, ak by sme chceli dosiahnuť rovnakých záruk, aké nám dáva naša peer-to-peer sieť. Výpadkom jedného servera by zároveň mohlo utrpieť niekoľko novín, zatiaľ čo v našej implementácii pri výpadku šéfredaktora noviny utrpia len po personálnej stránke, kedy nie je možné pridávať nových redaktorov, ale pridávanie a sťahovanie článkov je naďalej funkčné. Závislosť plynie taktiež z toho, že je potrebné nejakým spôsobom obchádzať systém NAT, pričom v mnohých prípadoch bude potrebné, aby sa dáta presmerovali cez nejakého iného peera, ktorý má verejnú

sieťovú adresu. Túto úlohu tvoria práve šéfredaktori jednotlivých novín.

Potrebu šifrovania sme vyriešili pomocou symetrického šifrovania na oboch koncoch komunikujúcich peerov, pričom prenos súkromných kľúčov je zabezpečený asymetrickým šifrovaním a digitálnym podpisom.

Sieť peer-to-peer

Na implementáciu siete peer-to-peer sme si vytvorili vlastný aplikačný protokol, ktorý nesie skratku NP2PS. V tomto protokole sme stanovili, že na komunikáciu peerov sa použije transportný protokol TCP a sieťový protokol IP verzie 4. Ďalej sme popísali komunikáciu na aplikačnej úrovni, teda aký formát majú jednotlivé správy, ktoré si posielajú peeri NP2PS peer-to-peer siete. Použitím Google Protocol buffers sme zabezpečili prenos všetkých potrebných údajov, napríklad hlavičky článku, jeho kryptografického hashu, podpisu tohto hashu redaktorom novín a nakoniec aj jeho obsahu. Na prenos informácií, ktoré sa týkajú sieťovania, ako napríklad transportná adresa, ale aj niektoré nízkoúrovňové záležitosti protokolu ako napríklad výmena verejných kľúčov RSA, sme použili protokol STUN. Do protokolu STUN sme pridali potrebné atribúty a metódy takým spôsobom, aby zvládal prenos týchto nízkoúrovňových a sieťových správ.

Protokol STUN sme použili aj na jeho pôvodný účel a to ako pomocný nástroj pri obchádzaní systému NAT. Jeho jediná metóda Binding nám však nestačila a preto sme si protokol rozšírili o niekoľko ďalších, ako Allocate, Identify a Send. Niektoré z týchto metód boli inšpirované protokolom TURN, ale väčšinou ide len o názov metódy a ich chovanie je rozdielne. Dostupné atribúty, ktoré ponúka protokol STUN nám taktiež prišli nepostačujúce a teda sme aj do tejto skupiny pridali nové položky, ako PublicKey a PublicIdentifier. Na protokole STUN sme teda vybudovali nástroj na obchádzanie NAT-u, ktorý je nasaditeľný aj v reálnej prevádzke, pričom sme skombinovali priamy prístup, kde sa peer pokúsi najprv pripojiť priamo a presmerovanie, kde sú správy presmerované skrz nejaký STUN server, najčastejšie sa jedná o šéfredaktora niektorých novín.

Program s užívateľským rozhraním

Pomocou grafického užívateľského rozhrania sme sa pokúsili vytvoriť čo najviac univerzálne rozhranie medzi používateľom a všetkými prostriedkami, ktoré protokol ponúka prostredníctvom jeho implementácie. Používateľ tejto aplikácie má teda možnosť sa kedykoľvek pripojiť do siete, resp. nadsiete a začať odoberať noviny. Prostredie ďalej ponúka prostriedky na založenie si svojich vlastných novín a taktiež správu ich redaktorov. Aplikácie dokáže nahrať súbory z pevného

disku a publikovať ich ako články či už pre noviny daného peera, alebo pre akékoľvek iné noviny, kde ho považujú za redaktora. Články stiahnuté zo siete sú replikované na pevnom disku a následne, ak to je potrebné, zdieľané s ostatnými peermi.

Aplikácia ďalej umožňuje serializáciu metadát do súborov a ich opätovné načítanie, ak by to bolo potrebné.

Aplikácia používa knižnice, ktoré sú voľne dostupné buď to ako balíky v rôznych distribúciách Linuxu, alebo prostredníctvom hostingov typu ako github. V operačnom systéme MS Windows sme jednotlivé knižnice inštalovali prostredníctvom správcu závislostí „vcpkg“. Program sa podarilo skompilovať na virtuálnom počítači s jedným jednojadrovým procesorom a veľkosťou RAM 1 GiB, čo považujeme za celkom prijateľné požiadavky.

Budúce rozšírenia

K zlepšeniu funkcionality a zážitku s používania by bola vhodná aj podpora multimediálnych dát, ako obrázky, videá a podobne. Tu by ale bolo taktiež potrebné, aby sa príliš dlhé správy delili na menšie a teda by nebol problém s horným obmedzením veľkostí STUN správ. Obrázky a najmä videá sú však vysoko náročné na pamäť a značne by zahltili sieťovú komunikáciu jednotlivých peerov.

Rozšírením, ktoré by ďalej znížilo záťaž na sieťovú komunikáciu, ale ktoré by znamenalo väčšiu výpočetnú záťaž, by bola podpora pre kompresiu článkov, alebo aj všetkých správ, počas sieťového prenosu.

Zoznam použitej literatúry

[1] Tirumaleswar Reddy.K et al. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 8656. Feb. 2020. DOI: 10.17487/RFC8656. URL: https://www.rfc-editor.org/info/rfc8656.

[2] Marc Petit-Huguenin et al. *Session Traversal Utilities for NAT (STUN)*. RFC 8489. Feb. 2020. DOI: 10.17487/RFC8489. URL: https://www.rfc-editor.org/info/rfc8489.

[3] Matt Holdrege a Pyda Srisuresh. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. Aug. 1999. DOI: 10.17487/RFC2663. URL: https://www.rfc-editor.org/info/rfc2663.

[4] Mihir Bellare, Phillip Rogaway a David Wagner. "The EAX mode of operation". In: *International Workshop on Fast Software Encryption*. Springer. 2004, s. 389–407.

[5] Noura Aleisa. "A Comparison of the 3DES and AES Encryption Standards". In: *International Journal of Security and Its Applications* 9.7 (2015), s. 241–246.

[6] Hamdan Alanazi et al. "New comparative study between DES, 3DES and AES within nine factors". In: *arXiv preprint arXiv:1003.4085* (2010).

[7] Shafi Goldwasser a Silvio Micali. "Probabilistic Encryption; How to Play Mental Poker Keeping Secret All Partial Information". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. San Francisco, California, USA: Association for Computing Machinery, 1982, s. 365–377. ISBN: 0897910702. DOI: 10.1145/800070.802212. URL: https://doi.org/10.1145/800070.802212.

[8] Mihir Bellare a Phillip Rogaway. "Optimal asymmetric encryption". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994, s. 92–111.

[9] Eiichiro Fujisaki et al. "RSA-OAEP is secure under the RSA assumption". In: *Annual International Cryptology Conference*. Springer. 2001, s. 260–274.

[10] Farah Jihan Aufa, Achmad Affandi et al. “Security system analysis in combination method: RSA encryption and digital signature algorithm”. In: *2018 4th International Conference on Science and Technology (ICST)*. IEEE. 2018, s. 1–5.

[11] Ross Anderson a Roger Needham. “Robustness principles for public key protocols”. In: *Annual International Cryptology Conference*. Springer. 1995, s. 236–247.

[12] Bryan Ford, Dan Kegel a Pyda Srisuresh. *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*. RFC 5128. Mar. 2008. DOI: 10.17487/RFC5128. URL: https://www.rfc-editor.org/info/rfc5128.

[13] Bryan Ford. “Peer-to-Peer Communication Across Network Address Translators”. In: *USENIX Annual Technical Conference*. Anaheim, CA, 2005.

Dodatok A

Inštalácia a používanie

A.1 Linux

Na inštaláciu sú potrebné nasledujúce knižnice a platformy:

- Platforma Qt verzie aspoň 5.
- Doxygen verzie aspoň 1.9.1.
  - Graphviz verzie aspoň 2.42.2.
- GNU C compiler verzie aspoň 10.
- Google Protocol buffers verzie aspoň 3.21.2. V tomto prípade je potrebné špeciálne si dať pozor, pretože niektoré distribúcie, napríklad Debian, nemajú túto verziu vo svojich balíkoch a teda je potrebné ju stiahnuť z internetu [1] a samostatne nainštalovať [2]. Pre staršie verzie to odmietne skompilovať vytvorené `.proto` súbory.
- CMake verzie aspoň 3.17.

Všetky uvedené verzie sú tie verzie, na ktorých prebiehal vývoj aplikácie. Je teda možné, že postačia aj nižšie verzie spomenutých knižníc a platforiem.

Na operačných systémoch založených na distribúcii Debian je možné nainštalovať jednotlivé programy a knižnice takto:

```
sudo apt install autoconf automake libtool curl make g++ \
  unzip cmake doxygen graphviz qtbase5-dev libcrypto++-dev
```

[1] `https://github.com/protocolbuffers/protobuf`
[2] `https://github.com/protocolbuffers/protobuf/blob/main/src/README.md`

Následne je potrebné si naklonovať repozitár s Google Protocol buffers a nainštalovať jeho obsah:

```
git clone https://github.com/protocolbuffers/protobuf.git
cd protobuf
git submodule update --init --recursive
./autogen.sh
./configure
make -j$(nproc) # $(nproc) zaisti pouzitie vsetkych CPU
```

Ďalej sa odporúča spustiť:

```
make check
```

Avšak po dokončení tohto príkazu mi na Debiane 11 vždy vyskočila chyba:

```
FAIL: protobuf-test
```

Všetky ostatné testy prebehli v poriadku. Avšak táto chyba nevyzerá mať vplyv na funkčnosť programu. Následne je potrebné spustiť nasledujúce príkazy ako root:

```
make install
ldconfig # obnova cahce zdielanych kniznic
```

Ak by posledný príkaz zlyhal na chybu:

```
-bash: ldconfig: command not found
```

je potrebné ho spustiť priamo:

```
su
/sbin/ldconfig
exit
```

V tejto chvíli je všetko nainštalované a pripravené na vybudovanie a spustenie. Význam niektorých nainštalovaných balíčkov:

- `doxygen` a `graphviz` – Slúžia na vygenerovanie dokumentácie z komentárov v zdrojovom kóde a na kreslenie dedičných grafov v nej.
- `qtbase5-dev` – Inštalácia Qt verzie 5 a jej vývojárskych knižníc.
- `libcrypto++-dev` – Kryptografická knižnica Crypto++.
- `cmake` – Nástroj CMake na budovanie aplikácie

Následne je potrebné v koreňovom adresári programu vytvoriť priečinok s názvom „build“ a presunúť sa doň. Potom je už len potrebné spustiť CMake na rodičovský adresár:

```
mkdir build
cd build
cmake ..
```

V tejto chvíli nám môžu vyskočiť chyby, ak sme nejakú knižnicu či program nenainštalovali správne, ale nič také by sa nemalo stať. Ďalej vybudujeme program príkazom:

```
make -j$(nproc)
```

Po vybudovaní je možné program spustiť príkazom:

```
./news_p2p_sharing
```

Dokumentácia v Doxygene je k dispozíci v priečinku, kde bol spustený `make`. Vytvorí sa tam adresár s názvom „html“, ktorý obsahuje súbor „index.html“, ktorý je hlavnou stránkou tejto dokumentácie.

A.2 Windows

Pre MS Windows je potrebné si nainštalovať Visual Studio 2022, alebo také, ktoré podporuje CMake.

Potrebné balíky boli nainštalované použitím balíkovanieho systému „vcpkg“. V tomto sa nachádzajú všetko potrebné balíky, pre potrebné architektúry. Osobne sme mali problém s knižnicou Crypto++, kde bolo potrebné nainštalovať jej staticky kompilovanú verziu. Výber správnej verzie ponecháme na čitateľa. Po stiahnutí „vcpkg“ systému je potrebné jeho obsah rozbaliť, otvoriť Príkazový riadok alebo PowerShell a premiestniť sa do priečinka, ktorý sa rozbalil a v ktorom sa nachádza spustiteľný binárny súbor pre „vcpkg“. Inštalácia prostredníctvom „vcpkg“ prebieha nasledovne takto:

```
.\vcpkg install protobuf
```

Nainštaluje protocol buffers kompilátor aj knižnice. Analogicky nainštalujeme aj ostatné knižnice. Na stránke [3] nájdeme zoznam balíkov, ktorými „vcpkg“ disponuje.

Použitie „vcpkg“ nie je povinné a ponechá sa táto voľba na čitateľovi.

Následne je už len potrebné otvoriť celý projekt vo Visual studiu a vybudovať ho. Následne bude možné program spustiť.

[3] `https://vcpkg.io/en/packages`