Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Ondrej Kaprál

Distropine - Distribuovaný systém pro komentování webových stránek

Katedra Softwarového Inženýrství

Vedoucí bakalářské práce:	RNDr. Filip Zavoral, Ph.D.
Studijní program:	Informatika
Studijní obor:	Obecná informatika

Praha 2012

Ďakujem pánovi RNDr. Filipovi Zavoralovi, Ph.D. za odborné rady, čas a odporúčania, ktoré mi dal.

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov.

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

V .... dňa .... Podpis autora

Název práce: Distropine — Distribuovaný systém pro komentování webových stránek

Autor: Ondrej Kaprál

Katedra: Katedra Softwarového Inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.

Abstrakt: V tejto bakalárskej práci si kladieme za cieľ navrhnúť a implementovať peer-to-peer variantu služby pre komentovanie internetových zdrojov. Z funkčného hľadiska sa jedná o implementáciu diskusného fóra v P2P prostredí. Pokúšame sa o dosiahnutie čo najviac decentralizovaného riešenia pri splnení podmienok: ukladanie obsahu v distribuovanej hašovacej tabuľke, zavedenie konceptu identity užívateľa a ochrana tejto identity pred odcudzením. Pri analýze dochádzame k záveru, že v prostredí kde užívatelia neustanovujú väzby a generovaný obsah je broadcastového typu je nutná globálne dôverovaná autorita. Zavádzame teda PKI s certifikačnou autoritou (CA). Záväzok maximálnej decentralizácie napĺňame viacerými opatreniami na úrovni dizajnu aplikácie. Umožňujeme beh viacerých inštancií CA, minimalizujeme moc CA monitorovať činnosť užívateľov, cenzurovať ich obsah apod. Užívateľské rozhranie aplikácie integrujeme do webového prehliadača, čim sa nám darí odtieniť užívateľa od zložitosti P2P modelu.

Klíčová slova: peer-to-peer, distribuované hashovací tabulky, Kademlia, PKI

Title: Distropine - Distributed System for Anotation of Web Pages.

Author: Ondrej Kaprál.

Department: Department of Software Engineering.

Supervisor: RNDr. Filip Zavoral, Ph.D.

Abstract: The aim of this bachelor thesis is to design and implement peer-to-peer version of a service for commenting of internet resources. From a functional point of view it is an implementation of P2P discussion forum. We attempt to achieve the highest possible degree of decentralization while meeting these requirements: usage of distributed hash table to store data content and deployment of user identity concept while protecting this identity against theft. In the analysis we came to realize that in the environment where users do not establish any bindings and content distribution is broadcast in nature there is need for globally trusted authority. For this reason it is deployed PKI with certification authority (CA). Our pursuit of maximum decentralization is reached by multiple measures in the design of the application. We allowed simultaneous operation of multiple CA instances, we have also minimized the power of CA to monitor user activities or to censor the content of the system. User interface of the software is integrated into web browser what helps us to shield user from complexity caused by P2P design.

Keywords: peer-to-peer, distributed hash tables, Kademlia, PKI

Obsah

1 Úvod **2**

2 Distribuované hašovacie tabuľky **4**
2.1 DHT obecne . . . 4
2.1.1 Sieťové overlay . . . 4
2.1.2 Definícia DHT . . . 4
2.1.3 Adresový priestor DHT . . . 5
2.1.4 Pojem vzdialenosti v adresovom priestore . . . 5
2.1.5 Blízkosť uzlov v topológii siete . . . 5
2.1.6 Pripojenie nového uzla (bootstrap algoritmus) . . . 6
2.1.7 Fluktuácia uzlov a protiopatrenia . . . 6
2.1.8 Útoky na DHT . . . 7
2.2 DHT Kademlia . . . 8
2.2.1 Topológia . . . 8
2.2.2 Vnútorný stav uzlu . . . 9
2.2.3 Protokol . . . 10
2.2.4 Bootstrap algoritmus . . . 11
2.2.5 Replikácia . . . 12
2.2.6 Protiopatrenia voči útokom na DHT Kademlia . . . 12

3 Dizajn riešenia **14**
3.1 Model aplikácie z pohľadu užívateľa . . . 14
3.1.1 Komentovanie webových stránok . . . 14
3.1.2 Pojem diskusného vlákna . . . 14
3.1.3 Hierarchia diskusných vlákien . . . 15
3.1.4 Identita užívateľa . . . 15
3.2 Komentár a hierarchia diskusných vlákien . . . 16
3.2.1 Zobecnenie na problém súborového systému . . . 16
3.2.2 Metadáta . . . 17
3.2.3 Získanie metadát z DNS . . . 17
3.2.4 Hašovanie zachovávajúce lokalitu . . . 18
3.2.5 Metóda s referenčnými záznamami . . . 20
3.3 Identita užívateľa . . . 23
3.3.1 Požadované vlastnosti . . . 23
3.3.2 Nasadenie PKI a CA . . . 24
3.3.3 Dátové štruktúry certifikátu a komentára . . . 25
3.3.4 Identifikačné karty . . . 27
3.3.5 Expirácia certifikátu a jeho obnovenie . . . 28
3.3.6 CA ako užívateľ . . . 28
3.4 Plánované rozšírenia . . . 30
3.4.1 Zavedenie reputácie užívateľov . . . 30
3.4.2 Revokácia . . . 31

4 Užívateľské rozhranie 34
4.1 GUI . . . 34
4.1.1 Rozšírenie webového prehliadača . . . 34
4.1.2 Dizajn . . . 34
4.1.3 Postup: Registrácia nového užívateľa . . . 35
4.2 Konzolové rozhranie, konfigurácia . . . 35
4.2.1 Prehľad funkcionality konzolového rozhrania . . . 35
4.2.2 Konfigurácia programu . . . 37
4.2.3 Príklad . . . 38

5 Štruktúra aplikácie a detaily implementácie 41
5.1 Dekompozícia problému . . . 41
5.2 Implementácia vrstevnatého dizajnu . . . 41
5.2.1 Trieda `Layer` . . . 42
5.2.2 Akcie . . . 42
5.2.3 Pakety . . . 43
5.3 Triedy ležiace mimo zásobník vrstiev . . . 43
5.3.1 Trieda `CommonCrypto` . . . 43
5.3.2 Trieda `Consts` . . . 44
5.3.3 Trieda `MyLogger` . . . 44
5.4 LHT . . . 45
5.4.1 Perzistentné uloženie dát . . . 45
5.5 Rozhranie program-GUI . . . 46
5.5.1 Podrobne o *rozšírení* prehliadača Chrome . . . 46
5.5.2 HTTP server . . . 46
5.5.3 Medzipamäť adresárových záznamov a komentárov . . . 46
5.5.4 Uloženie stavu GUI . . . 47

Záver 48

Zoznam použitej literatúry 50

1. Úvod

V posledných rokoch došlo k masívnemu rozšíreniu sociálne-orientovaných internetových služieb, v ktorých je užitočný obsah generovaný výlučne užívateľmi (ďalej UGC[1] služby). UGC služby bývajú verejne dostupné a bezplatné. Ide o aplikácie pre budovanie sociálnych sietí, (mikro)blogovacie služby, systémy typu *wiki*, systémy pre anotáciu webu, *social bookmarking*, diskusné fóra atď.

Charakteristické pre tieto služby je, že užívatelia (resp. tvorcovia obsahu) si sú navzájom rovní, alebo existuje určitá hierarchia do ktorej sú zaradzovaní na základe svojej reputácie[2]. Ďalej platí, že generovaný obsah je prístupný verejne (wiki, diskusie) alebo je prístup k nemu daný väzbami, ktoré užívatelia medzi sebou ustanovujú (sociálne siete, mikroblogy). Na úrovni tejto koncepcie teda nie je v systéme nutná žiadna centrálna autorita.

Na nižšej úrovni abstrakcie však majú UGC služby prevažne centralizovanú architektúru — fungujú na báze modelu klient-server. Existuje centrálny prvok, ktorý uchováva dáta celého systému a ktorý sprostredkováva všetky interakcie užívateľov so službou.

Z pohľadu poskytovateľa služby je problematické škálovanie pri vzrastajúcom počte klientov a množstve dát a tiež fatálny dopad prípadného zlyhania centrálneho prvku. Z pohľadu tvorcu obsahu sú opodstatnené obavy v prípade, keď sa jedná o dáta súkromného charakteru a poskytovateľom je komerčný subjekt[3].

Na základe uvedeného je teda prirodzené hľadať spôsoby ako UGC služby realizovať decentralizovane, tj. na báze modelu peer-to-peer, ktorý lepšie zodpovedá podstate týchto služieb. V rámci tejto snahy v súčasnosti vznikajú aplikácie pre budovanie sociálnych sietí (Safebook[1], Diaspora[2, 3], Friendica[4]), mikroblogovanie (Thimbl[5], StatusNet[6]) a mnohé ďalšie.

Služba pre komentovanie webových zdrojov

Cieľom tejto práce je navrhnúť a implementovať P2P alternatívu konkrétnej inštancie UGC služby: služby pre komentovanie webových zdrojov na internete. Existujúce klient-server implementácie sú napr. sitejabber.com alebo doo.gl.

V princípe je komentovanie webových zdrojov podobné diskusnému fóru. Pre ľubovoľný webový zdroj možno v tejto službe vytvoriť diskusné vlákno, kam užívatelia publikujú príspevky týkajúce sa dotyčného zdroja (komentáre, hodnotenia, recenzie...). Pod webovým zdrojom je tu myslený ľubovoľný obsah (typicky webová stránka) dostupný na globálnom internete prostredníctvom URL identifikátora. Užívateľ má v rámci služby svoju identitu, pričom vystupuje pod pseudonymom, ktorým sú označené jeho príspevky.

Účelom služby tohto typu je umožniť diskusiu nad webovými zdrojmi, u ktorých to príslušný poskytovateľ neumožňuje[4] a zároveň zaručiť, že poskytovateľ nemôže do diskusie zasahovať inak, než z pozície bežného užívateľa.

[1] User Generated Content

[2] Viď hierarchia prispievateľov Wikipedie.

[3] Viď populárny citát “If you are not paying for it, you're not the customer; you're the product being sold.”

[4] Alebo sprístupnenie diskusie nie je z princípu možné, viď protokol FTP.

Základné vlastnosti projektu

Projekt, ktorý je predmetom tejto bakalárskej práce som nazval *Distropine*. V projekte sa pokúsim o dosiahnutie čo najviac decentralizovaného riešenia pri splnení týchto podmienok:

1. Obsah (tj. predovšetkým diskusné príspevky) bude ukladaný distribuovane medzi uzly P2P siete. Projekt zahŕňa vlastnú implementáciu DHT[8] algoritmu Kademlia[9], ktorým bude realizovaná P2P infraštruktúra a distribuovaná dátovej vrstva aplikácie.

2. Bude kladený dôraz na bezpečnú autentizáciu užívateľov, zabezpečenie integrity obsahu a ochranu pred krádežou identity. Jedná sa o vlastnosti, ktorých vynútenie je nad P2P modelom relatívne netriviálne (v porovnaní s modelom klient-server).

3. Spôsob interakcie užívateľa s aplikáciou a vzhľad GUI bude podobný tomu, na ktorý je užívateľ zvyknutý z podobných služieb na báze klient-server. Budú implementované aj ďalšie vlastnosti, ktoré umožnia reálne nasadenie aplikácie (napr. schopnosť uzlu pracovať za NAT).

4. Pripravenosť návrhu na budúce rozšírenia aplikácie v podobe zavedenia reputácie užívateľov a protiopatrení voči spamu.

Program bude implementovaný v jazyku C# platformy .NET a určený pre operačný systém MS Windows. Programovací jazyk C# som zvolil kvôli vysokej produktivite práce. Knižnice .NET Framework naviac umožňujú efektívny vývoj mnohovláknových aplikácií a prácu s asynchrónnymi udalosťami, tj. vlastnosti, ktoré sú pri vývoji sieťovej aplikácie významné.

Program bude *open source* (licencia GNU GPL [7]) a bude zverejnený na stránkach *sourceforge.net*.

Štruktúra práce

V nasledujúcej kapitole je popísaný teoretický koncept distribuovaných hašovacích tabuliek a špeciálne algoritmus Kademlia. Mierny dôraz je kladený na rozbor možných útokov voči DHT a protiopatrenia voči nim.

V kapitole 3 je popísaný dizajn riešenia. Je zdôvodnená voľba DHT ako podkladovej dátovej vrstvy a popísaný dátový model, ktorý je nad ňou budovaný. Sú diskutované a riešené aj problémy, ktoré sa v P2P prostredí stávajú netriviálnymi, ako napr. zabezpečenie integrity dátových položiek, registrácia užívateľov a ich autentizácia.

Kapitola 4 sa venuje užívateľskému rozhraniu aplikácie a jej konfigurácii.

Kapitola 5 popisuje základnú schému implementácie a rozoberá niektoré zaujímavejšie implementačné detaily.

V záverečnej kapitole sa diskutuje do akej miery sa podarilo naplniť požiadavky definované v tomto úvode a o plánoch ďalšieho vývoja.

2. Distribuované hašovacie tabuľky

V prvej časti tejto kapitoly sa pojednáva o DHT obecne. V druhej časti je opísaný algoritmus Kademlia, ktorý je použitý v aplikácii Distropine.

2.1 DHT obecne

2.1.1 Sieťové overlay

Pre definíciu konceptu DHT, bude potrebný pojem sieťového overlay.

Sieťové overlay možno vágne označiť ako ‚sieť nad sieťou'. Vzniká, keď uzly podkladovej sieťovej topológie (napr. na báze TCP/IP) medzi sebou udržiavajú sieť logických prepojení na úrovni aplikačnej vrstvy. Podkladová sieťová infraštruktúra je následne len médiom, ktorým tieto logické linky vedú.

V rámci overlay majú uzly svoju identifikáciu, ktorá môže byť nezávislá od identifikácie na podkladovej sieti. Identifikátor uzlu v overlay nazveme *nodeID*. Plní funkciu adresy, na základe ktorej sú v overlay smerované správy.

V kontexte vrstevnatého sieťového modelu vzniká na aplikačnej vrstve podkladovej siete nová smerovacia vrstva. Táto vrstva využíva abstrakciu point-to-point spojení, ktorú poskytuje protokolový zásobník podkladovej siete.

Sieťové overlay možno klasifikovať do dvoch tried podľa toho, či výsledný graf aproximuje nejakú štruktúru alebo sú jeho hrany vytvárané ad-hoc:

- neštruktúrovaný prístup. Príkladom môže byť P2P sieť Gnutella[14]. Jediným spôsobom ako v neštruktúrovanom overlay doručiť správu zatiaľ neznámemu uzlu je použitím *záplavového smerovania*. Tento prístup nezaručuje korektný výsledok a prináša neškálovateľnosť.

- štruktúrovaný prístup. Pozícia uzlu v overlay je daná jeho identifikátorom na základe nejakého predpisu. Vďaka tomu je možné vytvoriť smerovací algoritmus, ktorý výslednú štruktúru prehľadáva a dané nodeID dokáže vyhľadať efektívne alebo s určitosťou prehlásiť, že neexistuje. Príkladom štruktúrovaných overlay sú DHT.

Tam, kde to bude z kontextu zrejmé bude v ďalšom texte pojem *sieť* odkazovať na *sieťové overlay* tvorené uzlami DHT.

2.1.2 Definícia DHT

Distribuovaná hašovacia tabuľka je trieda metód distribuovaného ukladania dát. Uzly DHT tvoria štruktúrované sieťové overlay, v ktorom komunikujú na báze modelu *rovný s rovným* (P2P).

DHT svojim participantom poskytuje abstrakciu dátovej štruktúry typu hašovacia tabuľka, tj. štruktúry, ktorá zverejňuje rozhranie tvorené funkciami `PutData(key,value)` a `GetData(key)`.

Funkcia `PutData` uloží dátovú položku medzi uzly siete tak, aby ľubovoľný účastník systému poznajúci príslušný kľúč `key` dokázal hodnotu `value` získať volaním `GetData`. Je to dosiahnuté tým, že spôsob mapovania kľúčov na uzly

(resp. predpis pre distribúciu položiek) umožňuje previesť tento problém na úlohu smerovania — tj. problém nájdenia uzlu s daným nodeID v overlay. Navyše, smerovanie možno v DHT riešiť efektívne vďaka jej štruktúrovanosti, typicky so zložitosťou $O(\log n)$ smerovacích skokov[1], kde n je celkový počet uzlov overlay.

V ďalšom texte budeme pre funkcie `PutData` a `GetData` používať označenia *uloženie*, resp. *vyhľadanie* položky.

2.1.3 Adresový priestor DHT

Nazrime na DHT ako na hašovaciu tabuľku. Do DHT sa ukladajú dátové položky z univerza $\mathcal{V}$. Kľúče, pomocou ktorých sú položky v tabuľke adresované sú z priestoru $K_{\mathcal{V}}$. Označeme ďalej množinu uzlov overlay ako $\mathcal{N}$ a priestor ich identifikátorov nodeID ako $K_{\mathcal{N}}$.

Spoločným rysom všetkých implementácií DHT je nasledovný obrat: prvky priestoru kľúčov dátových položiek $K_{\mathcal{V}}$ a prvky priestoru identifikátorov uzlov $K_{\mathcal{N}}$ sú prvkami spoločného rozsahu $\mathcal{I}$. Obecne ide o priestor k-bitových binárnych reťazcov:

$$\mathcal{I} = \{0, 1\}^k$$

$$K_{\mathcal{V}} \subseteq \mathcal{I}, K_{\mathcal{N}} \subseteq \mathcal{I}$$

Konštanta k býva veľká, napr. 160, čím dostávame obrovský adresový priestor.

Rozsahu $\mathcal{I}$ budeme hovoriť *adresový priestor DHT*, jeho prvky sú DHT adresy. Pojem kľúč použijeme pri zdôraznení, že sa jedná o adresu dátovej položky, nodeID pri zdôraznení adresy uzlu.

2.1.4 Pojem vzdialenosti v adresovom priestore

Spoločnou vlastnosťou DHT algoritmov je, že uzly si prerozdeľujú dátové položky na princípe lokality, tj. uzly vo svojich databázach uchovávajú položky, ktorých kľúče sú blízke ich nodeID.

Špecifikom každého DHT algortimu je konkrétny spôsob, akým sa pojem vzdialenosti v adresovom priestore $\mathcal{I}$ zavedie. Ide teda o definovanie funkcie vzdialenosti dvoch adries $\delta(k_1, k_2)$.

Priestor $\mathcal{I}$ nemusí byť nutne metrický priestor, resp. funkcia δ nemusí byť metrikou. V prípade algoritmu *Kademlia*[9] má funkcia δ predpis $k_1 \oplus k_2$ a skutočne metrikou je, zatiaľ čo napr. algoritmus *Chord* [10] má kruhový priestor $\mathcal{I}$ a vzdialenosť kľúčov sa určuje „v smere hodinových ručičiek”, takže nie je zachovaná ani podmienka symetrie metriky: $\delta(k_1, k_2) = \delta(k_2, k_1)$.

Keď budeme ďalej hovoriť o *blízkosti* (príp. okolí) identifikátorov nodeID alebo kľúčov, príp. o blízkosti uzlov alebo dátových položiek, hovoríme vždy v kontexte funkcie vzdialenosti danej funkciou δ.

2.1.5 Blízkosť uzlov v topológii siete

Ako bolo uvedené, DHT vytvára štruktúrované overlay. Uzly si teda tvoria svoje smerovacie tabuľky tak, aby výsledok formoval konkrétnu štruktúru, ktorá býva špecifikom daného DHT algoritmu. Aj tu sa využíva princíp lokality: obecne

[1]Táto zložitosť je daná konkrétnym DHT algoritmom.

možno povedať, že uzol má najúplnejšiu informáciu o uzloch vo svojom bezprostrednom okolí a so zväčšujúcou sa vzdialenosťou sa informácia zmenšuje.

Práca [8] vymenováva rôzne štruktúry, ktoré DHT algoritmy formujú svojimi overlay sieťami. V prípade algoritmu *CAN* je to graf na d-rozmernom tore, *Pastry* aproximuje hyperkocku, *Viceroy* graf typu *butterfly network* atď.

2.1.6 Pripojenie nového uzla (bootstrap algoritmus)

Uzol musí pred pripojením do siete získať identifikátor nodeID. Najjednoduchší spôsob je nechať uzol vygenerovať si ho náhodne. To však nie je bezpečné — útočníkovi to umožňuje nastaviť nodeID svojich uzlov tak, aby sa dostali do susedstva uzlu alebo dátovej položky, na ktorú sa útočí. Následne môže realizovať tzv. *Eclipse* útok, viď 2.1.8. Preto býva nodeID výstupom kryptografickej hašovacej funkcie iného údaju, napr. IP adresy, alebo verejného kľúča asymetrickej šifry.

Ak sa uzol nepripája do siete prvýkrát, je výhodné recyklovať identifikátor z predchádzajúceho pripojenia. Ak by uzol generoval nové nodeID, jeho pozícia v adresovom priestore by sa zmenila, takže dátové položky, ktoré drží od posledného pripojenia by u neho nikto nehľadal.

Ďalšou nutnosťou pred pripojením je kontakt aspoň na jeden uzol, ktorý do siete už pripojený je (tzv. *bootstrap uzol*). Jeho adresa môže byť súčasťou konfigurácie, získaná pomocou broadcastu, príp. získaná sofistikovanejším technikami, viď [12, 13].

Pripojenie nového uzlu do DHT začne tým, že sa kontaktuje bootstrap uzol a pomocou smerovacieho algoritmu sa nájdu uzly, ktoré sú blízke nodeID nového uzla. Nový uzol týchto susedov kontaktuje, čo spôsobí zavedenie jeho kontaktu do ich smerovacích tabuliek. Následne mu môžu predať dátové položky, ktoré sú v danej časti DHT uložené.

2.1.7 Fluktuácia uzlov a protiopatrenia

Dizajn DHT algoritmu musí počítať s tým, že väčšina nových uzlov bude do P2P siete pripojená len veľmi krátko[2]. Tejto nestabilite, resp. dynamike množiny uzlov hovoríme *fluktuácia uzlov*. Uzol, ktorý je v sieti krátko, by teda nemal byť zavedený do smerovacích tabuliek veľkého počtu uzlov, aby jeho odpojenie nedestabilizovalo sieť. Zároveň by sa nemalo počítať s tým, že uzly budú pri opúšťaní siete notifikovať svoje okolie.

Protiopatrením voči strate dát spôsobenej odpojením alebo poruchou uzlu je udržiavať z každej dátovej položky niekoľko kópii distribuovaných medzi uzly a pravidelne ich replikovať. Počet kópií označme k a replikačný interval označme t_R.

Pre nastavenie týchto konštánt by malo platiť nasledujúce: pre k náhodne zvolených uzlov je pravdepodobnosť, že po uplynutí času t_R budú všetky tieto uzly nedostupné menšia ako maximálne riziko straty dát, ktoré sme ochotní v danej aplikácii akceptovať.

[2] Predpokladáme verejne prístupnú P2P sieť, napr. určenú na zdieľanie súborov, kde sa účastníci voči sieti nechovajú altruisticky.

2.1.8 Útoky na DHT

Voči DHT existuje niekoľko *generických* útokov, tj. takých ktoré možno aplikovať na ľubovoľný DHT algoritmus. Využívajú zraniteľnosti plynúce z elementárneho dizajnu DHT systému, príp. sú to útoky voči overlay sieťam obecne.

Táto sekcia poskytuje základný prehľad typov týchto útokov. Nebudú tu menované obecné útoky ako napr. (D)DoS alebo útoky, ktoré môže podniknúť útočník kontrolujúci operačný systém, kde DHT uzol beží ako proces.

V nasledujúcom prehľade budeme kvôli stručnosti nazývať napadané uzly ako *obete* a uzly pod kontrolou útočníka ako *záškodnícke uzly*.

Vyvolanie kolízie v DHT Útočník, ktorý do DHT publikuje dátovú položku, ktorá má kľúč zhodný s niektorou už existujúcou dátovou položkou môže dosiahnuť nedostupnosť pôvodnej hodnoty, resp. jej odstránenie z DHT.

Obrana voči tomuto útoku je vysvetlením veľkého adresového priestoru, ktorý je spoločným prvkom návrhu DHT algoritmov. Výpočet kľúča dátovej položky totiž zvykne byť realizovaný kryptografickou hašovacou funkciou aplikovanou na hodnotu dátovej položky. Znamená to, že kľúč nemusí byť explicitne prítomný a je možné ho kedykoľvek vypočítať z hodnoty dátovej položky. Vďaka vlastnostiam kryptografických hašovacích funkcií ako je odolnosť voči kolízii a odolnosť voči nájdeniu druhého obrazu, môže následne návrh DHT algoritmu implicitne predpokladať, že ku kolízii dátových položiek nemôže dôjsť.

Eclipse. Útok typu *Eclipse* je vykonaný keď útočník vloží záškodnícke uzly do topológie siete tak, aby smerovacie tabuľky obetí obsahovali výlučne (alebo z veľkej časti) referencie na uzly útočníka (príp. iných obetí toho istého útoku). Znamená to, že všetka komunikácia obetí so zvyškom siete prechádza cez záškodnícke uzly. Útočník môže správy prechádzajúce dnu a von monitorovať, pozmeňovať ich alebo ich úplne odfiltrovať, čim dosiahne zneviditeľnenie obetí z pohľadu zvyšku siete.

***Eclipse* útok na dátovú položku.** Tento útok sa podobá *Eclipse* útoku tým, že útočník v rámci adresového priestoru DHT umiestňuje svoje uzly do blízkosti konkrétnehu kľúča dátovej položky. Následkom toho môže získať správu danej položky výhradne do svojej kompetencie, znemožniť replikáciu a následne ju z DHT odstrániť.

Sybil. V systémoch, kde nemožno kontrolovať počet uzlov, ktoré môže mať pod kontrolou jeden subjekt, prichádza do úvahy útok typu *Sybil*. Tento útok môže ochromiť sieť v tom zmysle, že záškodnícke uzly vracajú zmätočné odpovede pri smerovaní apod. Pokiaľ sa záškodnícke uzly chovajú transparentne, môžu monitorovať všetko dianie na sieti a všetok obsah vytvorený jej užívateľmi.

Útok zvýšením *churn rate*. Aj útočník, ktorý nevlastní dostatok uzlov na zahájenie útoku typu Sybil, môže byť úspešný, pokiaľ sieť nie je odolná voči vysokej miere fluktuácie uzlov (tzv. *churn rate*).

Útočník vytvorí záškodnícke uzly, informácia o nich ovplyvnia smerovacie tabuľky iných uzlov na sieti a zverí sa im do kompetencie časť obsahu DHT.

Tieto uzly následne na príkaz útočníka zanikajú. Ostatné uzly teda majú „znečistené” smerovacie tabuľky a v prípade nedostatočnej replikácie dochádza k strate dát.

Klamlivé smerovanie. V tomto prípade záškodnícky uzol vracia nepravdivé smerovacie informácie. Nech napríklad obeť požiada záškodnícky uzol o zoznam kontaktov na uzly, ktoré sú blízko daného kľúča. Záškodnícky uzol vráti odkazy na iné záškodnícke uzly, ktoré sú danému kľúču bližšie než on sám, takže odpoveď je z pohľadu žiadateľa korektná. Akonáhle sa teda začne smerovať na základe informácií od záškodníckeho uzla, prebieha celý zvyšok cesty k hľadanému kľúču vo vnútri podsiete kontrolovanej útočníkom.

Útok na integritu dátových položiek Tento útok vyplýva z toho, že v DHT je dátová položka po publikovaní umiestnená na uzly, ktoré sú v adresovom priestore blízko jej kľúču. Jej tvorca teda nad ňou nemá kontrolu. Ak neexistuje mechanizmus na zabezpečenie integrity položky, môže ju ktorýkoľvek z uzlov nedetekovateľne pozmeniť.

Zabezpečenie integrity dátových položiek však nie je v kompetencii DHT algoritmu ale vyššej vrstvy aplikácie, ktorá zabezpečí podpisovanie dátových položiek a súvisiacu problematiku dôvery.

2.2 DHT Kademlia

Nasledujúci popis algoritmu DHT Kademlia vychádza z pôvodného článku [9].

2.2.1 Topológia

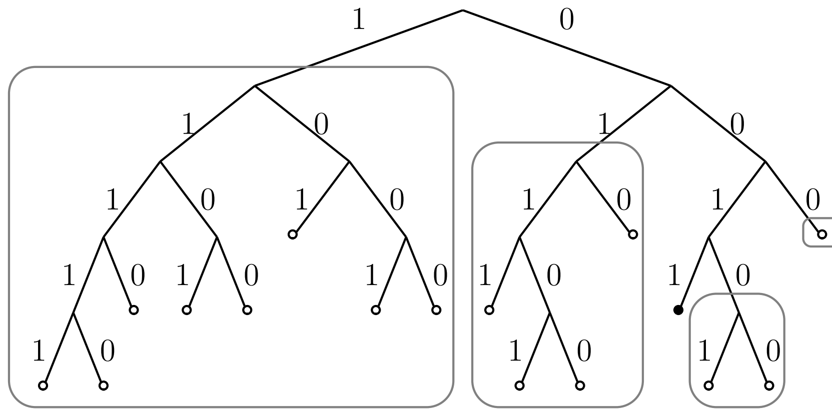
V DHT Kademlia je adresový priestor priestorom 160-bitových binárnych reťazcov, napr. výstupov hašovacej funkcie SHA-1. V tomto priestore sa pojem vzdialenosti zavádza funkciou $\delta(k_1, k_2) = k_1 \oplus k_2$.

Použitá funkcia XOR má vlastnosti symetrie, trojuholníkovej nerovnosti a tiež jednosmernosti[3]. Tieto vlastnosti prinášajú niektoré výhodné rysy viď [9] a zjednodušujú formálny dôkaz správnosti algoritmu (tj. úspešnosti smerovacieho algoritmu a vysokej pravdepodobnosti, že dáta uložené do DHT nebudú stratené ani pri vysokej fluktuácii uzlov).

Kademlia považuje uzly siete za listy binárneho stromu, kde pozícia každého uzlu je daná najkratším unikátnym prefixom jeho nodeID. Z pohľadu konkrétneho uzlu je dôležité rozdelenie stromu na tzv. *smerovacie podstromy.* Je to postupnosť zmenšujúcich sa disjunktných podstromov o ktorých platí, že neobsahujú uzol, z pohľadu ktorého sú stavané (a sú najväčšie možné s touto vlastnosťou). Strom uzlov algoritmu Kademlia a jeho rozdelenie na smerovacie podstromy ilustuje obrázok 2.1.

Z obrázku je patrné, že v danom smerovacom podstrome začínajú nodeID všetkých uzlov tým istým prefixom. Dĺžka prefixu ja daná hĺbkou koreňa smerova-

[3] Jednosmernosť znamená, že pre každý kľúč k a vzdialenosť $\triangle > 0$, existuje práve jeden kľúč k' t.ž. $\delta(k, k') = \triangle$.



Obr. 2.1: Príklad binárného strom uzlov. Sivou farbou ohraničné podstromy sú smerovacie podstromy, konštruované z pohľadu čierneho uzla (uzla s prefixom 0011). Obrázok je prebraný z článku [9].

cieho podstromu v rámci celej stromovej štruktúry. V príklade uzlu $0011\cdots$ majú smerovacie podstromy prefixy 1, 01, 000, a 0010.

Podstatou protokolu Kademlia je zaistiť, aby každý uzol mal vo svojej smerovacej tabuľke aspoň jedného reprezentanta z každého smerovacieho podstromu (ak tento podstrom obsahuje nejaký uzol).

Smerovací algoritmus

Toto usporiadanie umožňuje navrhnúť nasledovný spôsob smerovania. Nech máme nodeID uzlu, ktorý chceme vyhľadať. Prechádzame tento binárny reťazec po bitoch a zastavíme sa vtedy, keď nájdeme prefix patriaci niektorému z našich smerovacích podstromov. Ak by sme v príklade uzlu $0011\cdots$ hľadali nodeID = $101\cdots$, zastavili by sme sa už v prvom bite, tj. pri podstrome 1. Kontaktovali by sme reprezentanta, ktorého pre tento podstrom máme. Tento reprezentant by odpovedal na základe rovnakého algoritmu aplikovaného na jeho štruktúru smerovacích podstromov. Určite by nám vrátil kontakt na uzol, ktorého nodeID má minimálne prefix 10. V každej iterácii sa teda približujeme k danému uzlu minimálne o jednu úroveň hĺbky stromu, resp. o jeden bit hľadaného nodeID. Ako plynie z tohto náčrtu, konvergencia algoritmu je logaritmická voči počtu uzlov siete.

V prípade, že sa v sieti nenachádza uzol s daným nodeID, dostaneme týmto algoritmom kontakt na uzol, ktorý je najbližšie. To je presne odpoveď, ktorú potrebujeme pri vyhľadávaní dátových položiek.

2.2.2 Vnútorný stav uzlu

Smerovacia tabuľka Kademlia uzlu pozostáva z tzv. k-kapes[4], ktorých je 160 — pre každý smerovací podstrom má uzol vyhradenú jednu kapsu.

Záznamy smerovacej tabuľky sú trojice v tvare $\langle IP\ adresa, port, nodeID\rangle$. Ďalej ich budeme označavoť ako *kontakty*. Každá kapsa môže obsahovať až k

[4] Napriek újme voči slovenskej spisovnosti sa budeme pridržiavať tohto českého prekladu názvu *k-buckets*.

kontaktov, pričom konštanta k, tzv. *replikačný parameter*, tu má rovnakú sémantiku ako bolo opísané v sekcii 2.1.7. V článku [9] je navrhovaná hodnota $k = 20$.

Možno očakávať, že prvá kapsa bude väčšinou zaplnená (v prvom bite sa od aktuálneho uzlu líši priemerne polovica uzlov siete) a najnižšie kapsy budú prázdne, pretože nemožno predpokladať, že v sieti bude mnoho uzlov s nodeID takmer zhodným voči nodeID aktuálneho uzlu.

Kontakty v kapse sú zotriedené podľa času, kedy bola naposledy overená ich konektivita, pričom kontakt s najčerstvejšie overenou konektivitou je prvý. Dôkazom konektivity uzlu je prijatie ľubovoľnej RPC správy (požiadavky alebo odpovede), ktorej je odosielateľom.

V prípade keď je kapsa plná a nájde sa nový kontakt, ktorý by bolo možné do nej vložiť, vloží sa do tzv. *nahradzovacej cache* (replacement cache)[5], kde sú kontakty taktiež zotriedené podľa čerstvosti overenia ich konektivity. Kontakt z nahradzovacej cache sa využije vtedy, keď sa preukáže, že niektorý kontakt z príslušnej kapsy je nedostupný. Vtedy sa vezme prvý kontakt z nahradzovacej cache a vloží sa do kapsy ako jej posledný kontakt.

Filozofia tohto postupu spočíva v preferencii kontaktov o ktorý sa vie, že sú dlhodobo dostupné (resp. nie je dokázaný opak). Autori algoritmu túto vlastnosť zaviedli na základe výsledkov meraní na reálnych P2P sieťach pre zdieľanie súborov. Tam sa ukázalo, že väčšina uzlov ostáva pripojených do siete po veľmi krátku dobu — preto je vhodné preferovať dlho-fungujúce uzly pred uzlami novými, i keď ich dostupnosť je čerstvejšie overená.

Tento rys navyše vytvára prirodzenú ochranu algoritmu pred (D)DoS útokom ako aj útokom založeným na umelom zvyšení *churn rate* siete, viď 2.1.8.

2.2.3 Protokol

Kademlia protokol pozostáva z troch typov RPC správ:

`FIND_NODE(key)`. Návratovou hodnotou je k kontaktov, ktoré sú vzhľadom na obsah k-kapes príjemcu najbližšie k argumentu *key*. Presnejšie, ide o zoznam všetkých kontaktov ktoré má príjemca v tej kapse, do ktorej spadá (na základe prefixu) kľúč *key*. Ak príslušná kapsa neobsahuje dostatok kontaktov, získajú sa z nasledujúcich kapes, príp. sa vráti menej kontaktov.

`FIND_VALUE(key)`. Podobne ako pri `FIND_NODE` sa vracia zoznam najbližších kontaktov k argumentu. Najskôr ale príjemca overí, či nemá vo svojej lokálnej databáze dátovú položku s kľúčom *key*. Ak áno, nevracia zoznam kontaktov ale priamo dáta.

`STORE(⟨key, value⟩)`. Tento RPC príkaz obsahuje v argumente dátovú položku, ktorú príjemca skopíruje do svojej lokálnej databázy. Neodosiela sa žiadna odpoveď.

[5] Pôvodný návrh algoritmu nepoužíval nahradzovaciu cache a počítal so zasielaním PING správ. Ten istý článok [9] ale v závere uvádza nahradzovaciu cache ako možnú optimalizáciu pre zníženie množstva servisných správ, ktoré tečú sieťou.

Tieto správy sú prenášané protokolom UDP, preto je potrebné, aby bola pod RPC vrstvou implementovaná relačná vrstva. Každá relácia by mala mať iniciátorom generovaný identifikátor (*RPC ID*), ktorý je súčasťou všetkých správ zaslaných na jej účet.

Iteratívne algoritmy

Tieto tri primitívy sú základom iteratívnych algoritmov pre vyhľadanie uzlu (*node_lookup*), vyhľadanie dátovej položky (*value_lookup*) a pre uloženie dátovej položky do DHT (*iterative_store*).

Vyhľadanie najbližších uzlov k danému kľúču, *node_lookup*(*key*). Tento algoritmus na sieti vyhľadá množinu k najbližších uzlov k danému kľúču *key*. Ide o algoritmus vychádzajúci z náčrtu smerovacieho algoritmu z časti 2.2.1.

V prvej iterácii sa vyberie α kontaktov z kapsy príslušnej ku kľúču *key* (prípadne, ak tam nie je dostatok kontaktov, použijú sa nasledujúce kapsy). Následne sa týmto kontaktom paralelne a asynchrónne pošlú požiadavky `FIND_NODE` s argumentom *key*.

Konštanta α je parameter paralelizmu, ktorý je globálny v rámci systému. V článku [9] je navrhovaná hodnota $\alpha = 3$.

Zoznamy kontaktov, ktoré uzol získava z `FIND_NODE` odpovedí si zhromažďuje v zozname rozdelenom na 3 časti: üzly už kontaktované, ktoré neodpovedali; üzly už kontaktované, ktoré odpovedaliä üzly ešte nekontaktované. Posledná časť obsahuje maximálne k kontaktov, ktoré sú doposiaľ najbližšie nájdené ku kľúču *key*. Z tejto časti zoznamu je v danej iterácii náhodne vybraných α kontaktov, voči ktorým sa vyšle požiadavka `FIND_NODE`.

Posledná iterácia nastáva, keď žiaden z α kontaktov v predchádzajúcej iterácii neodpovedal v časovom limite alebo v situácii, keď z odpovedí predchádzajúcej iterácie nebol získaný žiadny nový kontakt, ktorým by došlo k zväčšeniu priblíženia ku kľúču *key*. V tejto iterácii sa vysiela `FIND_NODE` *všetkým* uzlom, ktoré zostali v časti üzly ešte nekontaktované. Výsledkom operácie je výsledná sada k najbližších kontaktov.

Vyhľadanie položky s daným kľúčom, *value_lookup*(*key*). Tento algoritmus má podobný priebeh ako predchádzajúci s tým rozdielom, že sa zasielajú RPC požiadavky typu `FIND_VALUE`. Akonáhle príde niektorá odpoveď s dátovou položkou, algoritmus končí. Ak algoritmus dobehne bez nájdenia položky, tj. skončí jeho posledná iterácia, je hľadenie prehlásené za neúspešné.

Uloženie položky do DHT, *iterative_store*$\langle$*key*, *value*$\rangle$. Uloženie dátovej položky spočíva vo vyhľadaní k uzlov, ktoré sú v sieti najbližšie ku kľúču *key*. Vyhľadanie týchto uzlov zabezpečí algoritmus *node_lookup*. Následne sa všetkým k nájdeným kontaktom položka odošle pomocou RPC `STORE`.

2.2.4 Bootstrap algoritmus

Obecný postup pri pripojení uzlu do siete bol popísaný v sekcii 2.1.6. Predpokladajme, že uzol už vlastní nodeID a kontakty na bootstrap uzly.

Kontakty si uzol vloží do svojej (zatiaľ prázdnej) štruktúry k-kapes a následne vyvolá algoritmus *node_lookup* použijúc svoje vlastné nodeID ako argument. Zoznam kontaktov, ktorý je výsledkom algoritmu sa zahodí. Pre bootrapping má význam postranný efekt tohto procesu: každá odpoveď, ktorá bola prijatá fungovala zároveň ako dôkaz konektivity odosielateľa a to spôsobilo vloženie jeho kontaktu do k-kapes.

Druhá fáza bootstrap algoritmu, má ešte zvýšiť počet uzlov v kapsách. Pre každú neprázdnu kapsu sa vygeneruje náhodné nodeID, ktoré by do nej na základe svojho prefixu patrí. Toto nodeID sa použije ako argument nového volania *node_lookup*. Aj v tomto prípade je účelom naplniť smerovacie tabuľky kontaktami na uzly, u ktorých sa overila ich dostupnosť a naopak, zaniesť informáciu o svojej prítomnosti v systéme medzi ostatné uzly. Preto sa výsledok volaní *node_lookup* zahadzuje.

Bootstrap algoritmus má aj ďalší vedľajší efekt spočívajúci v tom, že stávajúce uzly zasielajú novému uzlu (správami RPC `STORE`) dátové položky, ktoré mu na základe jeho nodeID prináležia. Tento proces prebieha nezávisle od replikácie dátových položiek (viď ďalej), takže nový uzol udržiava zodpovedajúce dáta okamžite po svojom vzniku a nie až v dobe, kedy u jednotlivých položiek dôjde k pravidelnej replikácii. Aby nedochádzalo k tomu, že uzol dostane tú istú položku od všetkých susedov, je zavedené pravidlo, že položku kopíruje uzol, ktorý je k nej v adresovom priestore najbližšie (túto informáciu dedukuje zo svojich k-kapes).

2.2.5 Replikácia

Uzol replikuje každú dátovú položku, ktorú u seba drží pravidelne raz za dobu t_R. Replikácia sa realizuje rovnako ako uloženie novej položky: algoritmom *iterative_store*.

Položku ale väčšinou udržiava až k uzlov súčasne, takže je treba docieliť, že behom intervalu t_R ju bude replikovať len jeden z nich. Preto po prijatí RPC `STORE` uzol vždy usúdi, že tým došlo k replikácii aj na zvyšných $k-1$ susedov a pre túto položku zakáže replikáciu po dobu t_R.

2.2.6 Protiopatrenia voči útokom na DHT Kademlia

V tejto sekcii sú vymenované jednotlivé typy útokov tak, ako boli definované v časti 2.1.8). Voči niektorým z nich je algoritmus prirodzene odolný vďaka svojmu dizajnu. Proti ostatným typom útokov existujú metódy ich eliminácie ako popisuje článok [11]. Všetky tu popísané metódy vychádzajú z tohto článku a (až na poslednú menovanú) sú implementované v aplikácii Distropine.

Eclipse útok na uzol a dátovú položku. Na ochranu voči tomuto útoku je prijaté nasledovné opatrenie: Každý uzol vlastní pár $\langle PK, SK\rangle$, čo sú kľúče asymetrickej podpisovacej schémy (napr. RSA). Ich generovanie má pod svojou kontrolou. Uzol svoje nodeID vypočíta ako funkciu svojho verejného kľúča: $nodeID = h(PK)$, kde h je kryptografická hašovacia funkcia.

Každá správa, ktorú uzol odošle obsahuje jeho nodeID a verejný kľúč. Zároveň je podpísaná tajným kľúčom SK.

Následkom toho nemá útočník kontrolu nad pozíciou svojho uzlu v priestore adries DHT. Útočník nie je schopný ani odcudziť nodeID iného uzlu bez znalosti príslušného tajné kľúču *SK*. Podpisovanie prenášaných správ je zároveň ochrana voči *man in the middle* útoku.

Sybil. V tomto prípade je útočníkovi skomplikovaný útok tým, že je časovo náročné generovať veľké množstvo identifikátorov nodeID.

Zachováme v platnosti vzťah pre nodeID z predchádzajúceho odseku a pridáme tzv. *kryptohádanku* (crypto-puzzle). Je to úloha, ktorej riešenie je časovo náročne ale kontrola riešenia je rýchla.

Kryptohádanka môže spočívať napr. v nájdení čísla X, kde:

$$h(X \oplus nodeID) = \underbrace{0 \cdots 00}_{c} \cdots$$

Konštanta c teda udáva dĺžku nulového prefixu pravej strany a tým aj zložitosť kryptohádanky. V priebehu existencie systému sa môže c zväčšovať, odrážajúc tak klesajúcu cenu výpočtových zdrojov. Vyriešenie kryptohádanky je problém s časovou zložitosťou $O(2^c)$, kontrola správnosti má zložitosť $O(1)$.

NodeID bude teda vždy uvádzané aj s riešením kryptohádanky, takže dvojica $\langle X, nodeID \rangle$ bude novým identifikátorom uzla.

Útok zvýšením *churn rate*. Ako sa uvádza v závere sekcie 2.2.2, algoritmus Kademlia je voči tomuto útoku prirodzene odolný kvôli jeho preferencii dlhofungujúcich uzlov.

Klamlivé smerovanie. Protiopatrenie voči tomuto útoku spočíva v úprave algoritmov **_lookup* tak, aby sa pri približovaní postupovalo po dvoch (a viacerých) disjunktných cestách. Uzol podľahne útoku až vtedy, keď všetky disjunktné cesty narazia na nejaký záškodnícky uzol, ktorý ich odkloní do podsiete útočníka.

3. Dizajn riešenia

3.1 Model aplikácie z pohľadu užívateľa

Motivácia a základná špecifikácia projektu bola definovaná v úvode tejto práce. Bolo taktiež vymenovaných niekoľko požadovaných rysov, ktorými má výsledok disponovať (napr. distribuovanosť, zabezpečenie obsahu, čo najväčšia decentralizovanosť).

Na základe týchto vstupov navrhnime model aplikácie tak ako sa bude javiť z pohľadu užívateľa. Tento model bude vynucovať podobu detailnejšieho dizajnu aplikácie.

3.1.1 Komentovanie webových stránok

Aplikácia Distropine je určená pre komentovanie internetových zdrojov obecne. Je ale prirodzené predpokladať, že typickým použitím bude komentovanie webových stránok ako špeciálneho prípadu internetového zdroja. Užívateľské rozhranie preto optimalizujme pre túto činnosť a integrujme ho priamo do webového prehliadača. Získame tým možnosť komentovať napr. aj FTP zdroje.

Užívateľ bude mať pri prehliadaní webu kedykoľvek možnosť vyvolať GUI aplikácie Distropine, ktoré zobrazí diskusné vlákno vzťahujúce sa k aktuálne prehliadanej stránke. Diskusné vlákno sa zobrazuje po stránkach s n komentármi.

3.1.2 Pojem diskusného vlákna

Diskusné vlákno aplikácie Distropine je postupnosť komentárov vzťahujúcich sa k rovnakému zdroju, pričom zdroj je daný názvom domény[1] a cestou v rámci domény. Pri zápise identifikátora URL v obecnom tvare

$$scheme://domain:port/path?query\#fragment$$

patria k rovnakému diskusnému vláknu komentáre zhodné v častiach *domain* a *path*. Reťazec v tvare *domain/path* preto nazvime *identifikátor vlákna*.

Takto nastavená definícia diskusného vlákna nie je vždy optimálna — jedná sa o prípady, kedy sa referencia na obsah nenachádza v časti *path* ale napr. v časti *query* (príkladom tohto fungovania je webová stránka `youtube.com`). Následkom toho prináležia komentáre všetkého obsahu v rámci domény do jediného diskusného vlákna. To prináša zlú čitateľnosť z pohľadu užívateľa a nevyváženú distribúciu dát medzi uzly z pohľadu DHT (ako vyplynie neskôr).

V prevažnej väčšine prípadov je však časť *query* buď prázdna alebo je využívaná na uloženie kontextu klientovej relácie s HTTP serverom a preposielanie stavu ovládacích prvkov formulárov (tzv. metóda GET). V tomto prípade začlenenie časti *query* do identifikátora vlákna nedáva zmysel.

Podobne nepriaznivý je prípad, keď si klient a HTTP server prenášajú referenciu na obsah pomocou technológie AJAX[18], takže časť *path* ani časť *query* na obsah neukazujú. URL môže byť v takých prípadoch behom celej relácie fixné (viď napr. server `vimeo.com`).

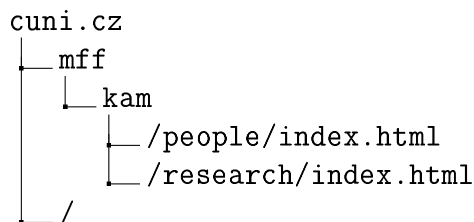
[1] Sú povolené výlučne DNS adresy verejných domén.

3.1.3 Hierarchia diskusných vlákien

Pri nazeraní na službu Distropine ako na diskusné fórum je prirodzená požiadavka na zobrazenie prehľadu existujúcich diskusných vlákien. Tie by mali byť organizované do rovnakej hierarchie ako zdroje, na ktoré sa viažu.

GUI preto bude zobrazovať nielen obsah aktuálneho diskusného vlákna (tj. komentáre) ale aj strom diskusných vlákien zakorenený v aktuálnej doméne. Po kliknutí na niektoré z vlákien aplikácia stiahne z DHT príslušné komentáre a vlákno zobrazí.

Napríklad nech aktuálne diskusné vlákno má identifikátor `cuni.cz/`. Potom pripadá do úvahy nasledovná hierarchia:



V rámci domény `cuni.cz` teda existujú tri diskusné vlákna: `cuni.cz/`, `kam.mff.cuni.cz/people/index.html` a `kam.mff.cuni.cz/research/index.html`. Je opodstatnená otázka, prečo je s časťou *path* identifikátora vlákna nakladané ako s neštrukturovaným reťazcom, keď môže byť použitá podobne ako časť *domain*, tj. mohla by generovať stromovú štruktúru. Dôvod je empirický. Málokedy ja časť *path* členená tak prehľadne ako to naznačuje toto URL:

```
edition.cnn.com/2012/04/08/world/meast/yemen-plot/index.htm
```

Vo väčšine prípadov je toto členenie z pohľadu klienta neprehľadné, nekonzistentné a zbytočne hlboké, viď napr.:

```
zpravy.idnes.cz/vyvoj-v-recku-po-volbach/zahranicni.aspx
www.czc.cz/k7-xpcpu-k7/35985/produkt
news.bbc.co.uk/2/hi/technology/8684110.stm
```

Preto nemá význam zaraďovať diskusné vlákna do hierarchie podľa časti *path*. Užitočné by to mohlo byť napr. vtedy, by sme aplikáciu Distropine optimalizovali pre komentovanie FTP zdrojov.

3.1.4 Identita užívateľa

Ako už bolo uvedené v úvode, užívateľ má v systéme Distropine vlastnú identitu. Pokúsime sa v P2P prostredí aplikácie Distropine o implementovanie klasického ponímania užívateľského účtu, ktoré je používané vo webových aplikáciách na báze klient-server

Aplikácia je po spustení v *anonymnom režime*. Užívateľ teda nie je prihlásený — je možné zobraziť ľubovoľné diskusné vlákno, no nemožno do neho prispievať.

Užívateľ má možnosť autentizovať sa. Ak ešte v systéme nemá založenú identitu, musí sa najprv registrovať. Prihlásený užívateľ môže vkladať do diskusného vlákna príspevky, ktoré sú podpísané jeho pseudonymom.

Užívateľ v tomto kontexte očakáva nasledovné garancie:

- je znemožnená krádež identity, tj. procesom registrácie je užívateľov pseudonym vyhradený výlučne jemu — nikto iný nemôže publikovať diskusné príspevky v mene iného užívateľa.

- v systéme neexistuje autorita s právomocou cenzurovať príspevky alebo autorita s možnosťou monitorovať aktivity užívateľov v systéme.

3.2 Komentár a hierarchia diskusných vlákien

3.2.1 Zobecnenie na problém súborového systému

Užívateľský model navrhnutý v sekcii 3.1 vyžaduje, aby sme nad DHT implementovali ukladanie komentárov v rámci diskusných vlákien a zároveň udržiavali hierarchiu, do ktorej sa tieto vlákna organizujú.
Pri uvažovaní nad touto úlohou sa javí ako užitočné previesť ju na obecnejší problém ukladania jednoduchého súborového systému[2] (pre účely budúcej referencie ho označme *DistropineFS*) nad DHT. Dôvodom tohto prístupu je jednoduchšie uvažovanie nad daným problém a možnosť použitia pojmového aparátu súborov a adresárov. So začiatkom sekcie 3.3 túto abstrakciu opustíme.
V zobecnení budú domény mapované na adresáre (meno adresára je menom domény), diskusné vlákna na súbory (meno súboru je časťou *path* identifikátora vlákna[3]) a komentáre na záznamy v súboroch.
Súborový záznam je dátová položka, u ktorej môžme uvažovať nasledovnú štruktúru:

$$fileRecord = FileRecord(path, publishDate, expirationDate, content)$$

kde *path* je plná cesta k súboru do ktorého záznam prináleží (napr. `cz/cuni/-mff/kam/index.html`), *publishDate* a *expirationDate* sú dátumy vzniku, resp. expirácie záznamu a pole *content* označuje dátový obsah záznamu. Táto štruktúra je abstrakciou dátovej položky diskusného komentára do tej miery, ktorá je z pohľadu jeho uloženia do hierarchie diskusných vlákien relevantná. Presný popis štruktúry komentára bude uvedený v sekcii 3.3.3.
Užívateľ voči DistropineFS vykonáva nasledovné operácie:

- Čítanie zo súboru. Súbor sa nečíta kompletný ale po stránkach o n záznamoch.

- Vloženie záznamu do súboru. Zapisuje sa výlučne v móde pridávania na koniec. Cesta, ktorá je špecifikovaná ako umiestnenie súboru (v položke *file.path*) nemusí pred zakladaním súboru existovať. Adresáre sú vytvorené podľa potreby rovnako ako súbor, ak doposiaľ neexistoval.

- Získanie obsahu adresáru, tj. zoznamu podadresárov a súborov.

[2] V zmysle stromovej štruktúry obsahujúcej adresáre a súbory.

[3] Na časť *path* identifikátora vlákna nazerajme ako na reťazec (napr. `/home/index.html`) a opomeňme, že obsahuje znaky '/', ktoré v reálnych súborových systémoch nie sú v menách súborov povolené.

Vidíme, že adresárová štruktúra v DistropineFS slúži len ako hierarchia, do ktorej sú zaraďované súbory. Adresár vzniká až vtedy keď je potrebné uložiť súbor do jeho postromu a mal by zaniknúť vo chvíli keď zaniknú všetky jeho súbory a podadresáre. Podobne je to so súbormi. Každý záznam súboru má stanovenú životnosť (viď *fileRecord.expirationDate*), po uplynutí ktorej zaniká. Súbor zaniká vtedy, keď zaniknú všetky jeho záznamy.
Pri návrhu je vhodné brať do úvahy, že operácie čítania súboru a získania obsahu adresára sú rádovo častejšie ako operácia zápisu do súboru, preto by mali byť lacné z pohľadu prístupov k podkladovému úložisku.
Nasledujúce sekcie diskutujú niekoľko alternatívnych spôsob implementácie DistropineFS nad DHT.

3.2.2 Metadáta

Základný problém pri implementácii adresárovej štruktúry nad DHT je ten, že súbory sú ukladané na adresy, ktoré sú výstupmi jednosmernej funkcie. Navyše, z dôvodu potreby vyváženej distribúcie obsahu medzi uzly by mal byť každý súbor uložený na vlastnú DHT adresu.
Príklad takého priradenia adresy súboru *file* je nasledovný:

$$key(file) = h(file.path)$$

Vidíme, že aj vtedy ak poznáme cestu k rodičovskému adresáru, tak dokým nie sú známe presné názvy potomkov (resp. podcesty k nim vedúce), nie sú odvoditeľné ich DHT adresy. Preto potrebujeme *metadáta*, zdroj informácii popisujúcich adresárovú štruktúru.
Priamočiarým riešením je uloženie metadát ako dátových položiek do DHT. Je však potrebné vziať do úvahy nasledujúce úskalia:

- Z dôvodu požadovanej decentralizácie nemôže existovať *Adresárová autorita*, ktorá by metadáta spravovala a ručila za ich konzistentnosť a aktuálnosť.
- Metadáta musia byť preto spravované kolektívne. Pri pozmenení alebo zániku referencovaných dát, príp. vzniku nových dát musí byť vyriešené, ktorý subjekt zrealizuje aktualizáciu metadát.
- Metadáta sú atraktívnym cieľom útoku. Útočník sa môže pokúsiť obmedziť k nim prístup alebo spôsobiť ich nekonzistenciu.
- Metadáta by mali byť uložené rozprestrene medzi mnoho adries a teda mnoho uzlov, pretože prístup k nim bude častý. Nie je preto prípustné uložiť všetky metadáta súborového systému na jedinú DHT adresu. Nedostatočne vyvážený je aj prístup, kedy sa na každý adresárový podstrom určitej konštantnej hĺbky (napr. hĺbky 2, tj. podstromy tvaru $x/y/*$) deleguje jedna DHT adresa (napr. $h(x/y)$), na ktorú sa ukladajú všetky metadáta tohto podstromu.

3.2.3 Získanie metadát z DNS

Strom adresárov DistropineFS je podstromom doménovej hierarchie systému DNS. Je na mieste otázka, či by nebolo možné využiť databázu DNS ako zdroj metadát vďaka ktorému by nebolo nutné udržiavať ich v DHT.

Nasledujúce body menujú dôvody, prečo nie je možné vydať sa touto cestou.

- V praxi nie je ľahké získať od DNS servera zoznam všetkých poddomén voči danej doméne, jednalo by sa o požiadavku na tzv. *zone-transfer*, ktorý DNS servery verejne neposkytujú.
- Zatiaľ čo predpokladáme, že štruktúru domén by sme poznali, tak pre danú doménu nemáme odkiaľ získať štruktúru zdrojov, ktoré obsahuje (v zmysle zdrojov adresovaných časťou *path* identifikátora URL).

  Pri tomto prístupe musia byť teda súbory ukladané na takú DHT adresu, ktorú možno odvodiť len z poznatku o rodičovskom adresári, napr. $key(file) = h(file.parentDir.path)$. Obsah všetkých súborov v adresári je teda ukladaný na jedinú DHT adresu, čo nie je optimálne vyvážené.
- Ak by sme napriek uvedeným obmedzeniam dokázali získať štruktúru poddomén alebo štruktúru zdrojov k danej doméne, jednalo by sa o len *kandidátov* na adresáre, resp. súbory. Voči týmto kandidátom by bolo nutné použiť metódu pokus-omyl: doména $\rightarrow$ prevod na DHT adresu $\rightarrow$ testovací lookup voči DHT. Toto je v rozpore s našou požiadavkou, aby časté operácie načítania komentárov a načítania adresárovej štruktúry boli nenáročné.

3.2.4 Hašovanie zachovávajúce lokalitu

Diskutujme ďalšiu alternatívu ako eliminovať potrebu udržiavania metadát v DHT. Pri generovaní adresy súboru nahraďme kryptografickú hašovaciu funkciu h, takou funkciou, aby súbory vzájomne blízke v adresárovej hierarchii boli hašované na kľúče blízke v zmysle metriky použitého DHT algoritmu.

Hašovacia funkcia

Uvažujme XOR metriku algoritmu Kademlia. Jednoduchý príklad hľadanej funkcie môže byť založený na konkatenovaní hašovaných reťazcov podľa adresárovej hierarchie:

$$key(file) = h(dir_0)|_{i_0} + h(dir_1)|_{i_1} + ... + h(filename)|_{i_k}$$

kde $file.path = dir_0/dir_1/.../dir_{k-1}/filename$, operátor + označuje konkatenáciu binárnych reťazcov, pre hodnoty i_j platí $\sum_{j=0}^{k} i_j = 160$ a zápis $s|_x$ označuje orezanie binárneho reťazca s na x-bitov. Výstup tejto funkcie je teda prvkom 160-bitového adresového priestor DHT Kademlia.
Vidíme, že dva súbory ležiace v spoločnom podstrome adresáru $dir_0/.../dir_m$ $(m < k)$ sú hašované na adresy, ktoré zdieľajú spoločný prefix dĺžky $\sum_{j=0}^{m} i_j$ bitov.

Intervalové dotazy nad DHT

Základnou otázkou tejto metódy je to, ako využiť princíp prenosu lokality u danej hašovacej funkcie za účelom extrakcie metadát z DHT. Všimnime si, že vyhľadanie súborov patriacich do určitého adresárového podstromu znamená pri tejto metóde dotaz na získanie dátových položiek patriacich do určitého intervalu

priestoru kľúčov. Odpoveďou by teda bola podpora intervalových dotazov nad DHT.
DHT algoritmy poskytujú rozhranie štandardnej hašovacej tabuľky, tj. umožňujú vyhľadávanie položiek jedine na základe plne známeho kľúča (*exact-matching*). Nad DHT vrstvou však možno vybudovať efektívne dátové štruktúry, ktoré intervalové dotazovanie umožnia. Zástupcami týchto štruktúr sú Prefixové hašovacie stromy [17] a Distribuované segmentové stromy [16].

Jednoduchý algoritmus získania potomkov adresáru

Pre účely tejto sekcie však postačí popis primitívnejšej techniky, ktorá sa bez uvedených štruktúr zaobíde, no nie je efektívna a škálovateľná.
Úlohou je teda na základe známej cesty $path = dir_0/dir_1/.../dir_m$ získať zoznam jej súborov a podadresárov. Vygenerujme spoločný prefix kľúču, ktorý zdieľajú všetky súbory v podstrome cesty $path$. Jedná sa o binárny reťazec $k' = h(dir_0)|_{i_0} + ... + h(dir_m)|_{i_m}$.
Vyhľadávajúci uzol sa pokúsi pokryť celý interval kľúčov s prefixom k', a to na minimálny počet DHT dotazov. Využije sa to, že na základe obsahu svojej smerovacej tabuľky je DHT uzol schopný odhadnúť celkový počet uzlov systému. Z toho môže vypočítať očakávané vzdialenosti medzi uzlami (vychádzajúc z predpokladu ich rovnomerného rozloženia v priestore adries) a následne môže odvodiť granularitu s akou je vhodné generovať nástrely kľúčov z prehľadávaného intervalu. DHT uzlom v blízkosti týchto kľúčov následne zasiela požiadavku na vrátenie zoznamu súborov, ktoré udržiavajú vo svojich lokálnych databázach.
Ak je cieľom získať len zoznam bezprostredných potomkov cesty $path$ (a nie zoznam všetkých súborov podstromu), potom za každý podadresár $path/sub$ stačí získať referenciu na jediný súbor ležiaci kdekoľvek v jeho podstrome. Tento súbor poslúži ako svedok existencie adresáru sub.

Nevýhody

Zrejmou nevýhodou definície funkcie key je to, že segmenty, z ktorých pozostáva DHT kľúč sú stanovené napevno a teda predpokladajú určitú rovnomernosť v košatosti adresárového stromu na jeho jednotlivých úrovniach. Táto metóda má však aj dva zásadnejšie problémy.
Predpokladajme situáciu, kedy je počet uzlov DHT menší než 2^i a máme množinu M dátových položiek, ktorých kľúče majú spoločný prefix dĺžky i-bitov. Vzhľadom na to, že uzly sú v adresovom priestore rozmiestnené rovnomerne, priemerná vzdialenosť medzi nimi je zaokrúhlene[4] 2^{160-i}. Zároveň platí, že vzdialenosť medzi ľubovoľnými dvoma kľúčmi dátových položiek z M je menšia než 2^{160-i}.
Pre predstavu teda možno konštatovať, že všetky dátové položky z M sa v priemernom prípade „vojdú do priestoru medzi dvoma uzlami DHT". Preto je pravdepodobné, že k všetkým dátovým položkám z množiny M bude pristupované tak, akoby mali ekvivalentný kľúč: pre každý uzol DHT platí, že vo svojej lokálnej databáze buď udržiava všetky položky z M alebo žiadnu položku z M.
Podľa tohto pozorovania by segmenty, z ktorých sa konštruuje kľúč nemali byť dlhé. Akákoľvek hodnota kľúča v dolných 130 bitoch totiž nemá v reálnych P2P

[4] Presne: $2^{160}/(2^i + 1)$

aplikáciách[5] vplyv na množinu uzlov, ktoré danú položku udržiavajú.
Druhý problém spočíva v tom, že vytvorená hašovacia funkcia nemá kryptografické vlastnosti. Útočníkovi sa rádovo zjednodušuje problém nájdenia druhého alebo prvého obrazu k danému kľúču, pretože obrazy jednotlivých segmentov, z ktorých je kľúč konkatenovaný možno hľadať samostatne. Zložitosť nájdenia obrazu pre celý kľúč je následne súčtom zložitostí nájdenia obrazov jednotlivých segmentov, z ktorých každý je z príliš malého priestoru. Voľba krátkych segmentov teda zjednodušuje útok, ktorý bol v časti 2.1.8 označený ako *Eclipse* útok na dátovú položku.

3.2.5 Metóda s referenčnými záznamami

Nasledujúca metóda implementácie DistropineFS nad DHT je nasadená v súčasnej implementácii aplikácie Distropine.

Definícia

Pri tejto metóde sú metadáta ukladané explicitne do DHT. Adresárový strom je reprezentovaný dvojicami rodič-potomok, ktoré sú do DHT ukladané ako dátové položky nazvané *referenčné záznamy*.
Referenčný záznam je nositeľom informácie, že na danej ceste ($path$) v adresárovej štruktúre sa nachádza súbor alebo adresár určitého mena ($childName$):

$$refRecord = RefRecord(path, childName, expirationDate)$$

kde $expirationDate$ je dátum expirácie referenčného záznamu (viď ďalej).
Pre ukladanie referenčných záznamov do DHT zaveďme rovnaký prístup ako pri ukladaní súborov a ich záznamov. Kolekciu referenčných záznamov vzťahujúcich sa k rovnakej ceste budeme ďalej nazývať *adresár*. Referenčné záznamy tvoriace adresár budeme ukladať na spoločnú DHT adresu, ktorá bude funkciou cesty.
Priradenie DHT adresy pre súbor a adresár má teda nasledujúci predpis:

$$key(file) = h(file.path)$$
$$key(dir) = h(dir.path)$$

Kryptografická hašovacia funkcia h nám okrem iného garantuje rovnomernosť uloženia dát (súborov) a metadát (adresárov) medzi uzly DHT.

Základný princíp

So zavedením referenčných záznamov vzniká medzi DHT adresami stromová štruktúra referencií. Hlavný zreteľ pri návrhu metód budovania a udržovania tejto štruktúry sa musí sústrediť na zaistenie jej konzistencie.
Zodpovednosť za existenciu a aktuálnosť referenčného záznamu delegujeme na potomka, ktorý je týmto záznamom referencovaný. Keď u súboru alebo adresáru (ktorý nie je koreňovým adresárom) dôjde k udalosti[6], je z jeho adresy zaslaná

[5] Tj. za „reálne" sú tu (s veľkou toleranciou) považované P2P siete s menej než $2^{30} \doteq 10^9$ uzlami.

[6] Pod udalosťou zatiaľ uvažujme vznik položky, neskôr pridáme udalosť súvisiacu s predĺžením jej životnosti.

notifikačná správa na adresu rodičovského adresára. Ak rodičovský adresár dosiaľ neexistoval, tak vzniká v dôsledku toho, že vzniká jeho prvý referenčný záznam. Tento model dobre zodpovedá princípu budovania štruktúry zdola-nahor, ktorý bol popísaný pri definícii DistropineFS.
Po prijatí notifikačnej správy z adresy potomka, leží na strane rodiča zodpovednosť za overenie pravdivosti jej obsahu. V možnosti ľahko overiť pravdivosť referenčného záznamu spočíva odolnosť týchto položiek voči útoku na konzistenciu štruktúry. Referenčný záznam síce obsahuje informáciu ktorú je ťažké získať iným spôsobom no možno ju ľahko overiť:

- dotazom voči DNS, ktorý potvrdí existenciu domény príslušnej k adresáru potomka,
- testovacím výberom z DHT, ktorý overí či potomok, skutočne existuje.

Po úspešnom overení pravdivosti notifikačnej správy sa v adresári vytvorí alebo aktualizuje príslušný referenčný záznam. Ďalej celý proces prebieha rekurzívne, takže ak adresár práve vznikol, alebo došlo k udalosti v súvislosti s predĺžením jeho životnosti, odosiela sa notifikačná správu na adresu nadradeného adresára.
Alternatívnou metódou voči tomuto kaskádovému prístupu by bolo, keby zodpovednosť za vytvorenie potrebnej adresárovej štruktúry niesol autor prvotného súborového záznamu v dôsledku ktorého vznikol nový súbor a príslušná hierarchia adresárov. Tento prístup by sa však spoliehal na to, že autor si túto povinnosť skutočne splní. V opačnom prípade by vznikali súbory a adresáre nedostupné z hlavného stromu a vytváralo by to priestor záškodníckym užívateľom. Preto je vhodné, že štruktúra ktorú sme navrhli vykonáva popísanú kaskádu notifikácii bez toho, aby mal na to vplyv spúšťač tohto procesu.

Notifikačné správy na úrovni DHT uzlov

V predchádzajúcej sekcii bolo používané (a bude používané aj naďalej) zjednodušenie *„z adresy potomka X sa posiela notifikačná správu na adresu rodiča Y"*. Tento proces v skutočnosti prebieha medzi uzlami spravujúcimi DHT adresy $x = h(X.path)$ a $y = h(Y.path)$.
V algoritme Kademlia je dátová položka udržiavaná k najbližšími uzlami voči jej kľúču. Je potrebné, aby všetky uzly spravujúce adresu y zodpovedajúcim spôsobom u seba aktualizovali alebo vytvorili referenčný záznam. Na druhej strane nie je nutné, aby notifikačnú správu vysielalo všetkých k uzlov spravujúcich adresu x. Stačí, aby ich bol dostatočný počet na to, aby bolo takmer isté, že správa bude odoslaná aspoň jedným z nich.
Bol zvolený autonómny prístup. V okolí adresy x sa vyberie k^* ($k^* < k$, napr. $k^* = 3$) najbližších uzlov. Uzly za účelom výberu týchto reprezentantov nepotrebujú komunikovať, sú schopné vyvodiť to zo svojich smerovacích tabuliek. Každý z k^* reprezentantov vyšle notifikačnú správa na adresu y. Správa sa vysiela ako dátová položka pomocou procedúry $iterative_store(y)$ (viď 2.2.3). Následkom toho, každý z k uzlov spravujúcich adresu y dostane notifikačnú správu najviac k^*-krát.

Životnosť referenčných záznamov

Ako je definované v 3.2.1, adresáre v systéme DistropineFS nie sú odstraňované explicitne ale mali by zanikať automaticky vtedy, keď zanikne ich obsah — súbory a podadresáre. Platí tiež, že životnosť súborov a následne adresárov nie je fixne daná pri ich vzniku, môže sa predlžovať. U súboru je to vtedy, keď je doň pridaný nový záznam, u ktorého je dátum expirácie neskorší než u ostatných záznamov. U adresára vtedy, keď sa predĺži životnosť niektorého jeho potomka.
Metódou ako implementovať toto chovanie by mohol byť *pooling* rodiča voči potomkovi, kedy by rodič pravidelne kontroloval či potomok, na ktorého vlastní referenčný záznam aj naďalej existuje. Ak by potomok nebol nájdený, rodič by odstránil príslušný referenčný záznam. Odstránením všetkých referenčných záznamov by rodič zanikol.
Základný princíp metódy referenčných záznamov (tak už bol popísaný) uvádza, že rodič nikdy proaktívne nekontaktuje potomka, ale potomok kontaktuje rodiča keď uňho dôjde k nejakej udalosti. Pokúsme sa navrhnúť riešenie v duchu tohto princípu.
Zaveďme hodnotu r_i a pomenujme ju *expiračná rezerva* referenčného záznamu v hĺbke i. Nech platí vzťah $r_0 = r, r_{i+1} = r_i/2$. Konštantu r nastavme napr. na 1 deň.
Nech dátum ukončenia životnosti potomka je d. Potomok rodičovi oznamuje tento dátum v notifikačnej správe, ktorú mu zasiela pri svojom vzniku. Rodič si v príslušnom referenčnom zázname, v poli *expirationDate*, zaznamená tento dátum posunutý o expiračnú rezervu, takže výsledkom je hodnota $d + r_i$, kde i je rodičova hĺbka v adresárovej štruktúre. Keď tento dátum nastane, referenčný záznam na potomka je u rodiča odstránený.
Nasledujúci príklad ilustruje vznik súboru `index.html` na ceste `cz/cuni/mff/kam/`. Predpokladajme na začiatku prázdnu adresárová štruktúru. Dátum expirácie súboru nech je d. Hodnoty na pravej strane označujú dátum expirácie príslušnej položky (všetky hodnoty sú v dňoch).

```
cz ...................... (d + 7/8 + 1 = d + 15/8)
└─cuni .................. (d + 3/8 + 1/2 = d + 7/8)
  └─mff ................. (d + 1/8 + 1/4 = d + 3/8)
    └─kam ............... (d + 1/8)
      └─index.html ...... (d)
```

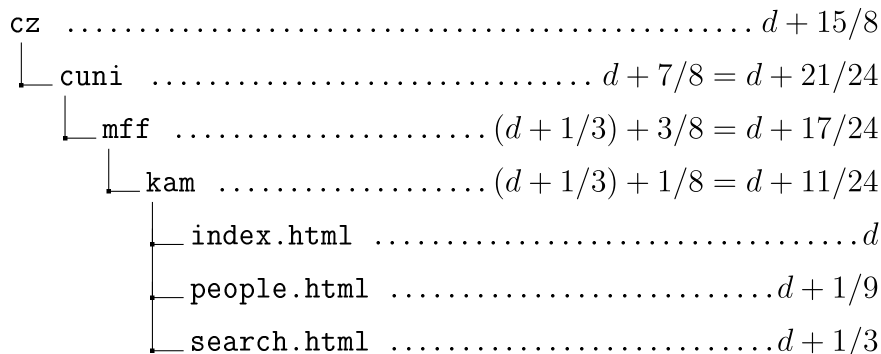
Týmto sme popísali vznik nového potomka. Teraz predpokladajme, že došlo k predĺženiu životnosti už existujúceho potomka. Konkrétne u potomka, ktorý je v hĺbke $i + 1$, na dátum d', $d' > d$. Ak

A. $d' \geq d + r_i$: V čase $d + r_i$ by došlo k situácii, že rodič by odstránil referenčný záznam na potomka, ktorému zatiaľ neskončila životnosť. Potomok tomu zamedzí tým, že odošle notifikačnú správu s hodnotou d'.

B. $d' < d + r_i$: Nehrozí, žeby rodič v budúcnosti odstránil referenčný záznam na potomka, ktorý ešte nezanikol. Potomok nepotrebuje kontaktovať rodiča.

Nech v adresárovej štruktúre z predchádzajúceho príkladu, v adresári `cz/cuni/mff/kam`, vznikne nový súbor `people.html`, ktorého dátum expirácie je $d' =$

$d + 1/9$. Jedná sa o prípad B, pretože $d + 1/8 > d'$. Notifikačná správa sa preto neposiela.
Nech je v tom istom adresári vytvorený súbor `search.html` ktorého dátum expirácie je $d' = d + 1/3$. Jedná sa o prípad A, pretože $d + 1/8 < d'$. Posiela sa teda notifikačná správa na adresu adresára `cz/cuni/mff/kam`.
V adresári `cz/cuni/mff/kam` dochádza k predĺženiu životnosti referenčného záznamu ukazujúceho na potomka `search.html`. V dôsledku toho dochádza aj k predĺženiu životnosti adresára samotného (expirácia adresára je daná jeho referenčným záznamom s najneskoršou expiráciou), a to na čas $d' + 1/8 = d + 11/24$.
Z dôvodu predĺženia životnosti sa zvažuje notifikácia rodiča, adresára `cz/cuni/-mff`. Dochádza tu k situácii A, pretože $d + 3/8 < d' + 1/8$. Na adresu adresára `cz/cuni/mff` sa teda vysiela notifikácia.
V adresári `cz/cuni/mff` je aktualizovaný príslušný referenčný záznam a je teda posunutá expirácia celého adresára. Tentokrát dochádza k situácii B, pretože expiračná rezerva u adresáru `cz/cuni` je väčšia než toto predĺženie životnosti.



Predpokladajme, že by bola expiračná rezerva nulová, resp. vôbec by sme ju nezaviedli. Kedykoľvek by bol do niektorého súboru pridaný záznam, ktorý by mal najneskorší dátum expirácie z celého adresárového stromu, došlo by k propagácii notifikačných správ až ku koreňovému adresáru. To by neúnosne zaťažovalo adresáre na vyšších úrovniach.
Popísaný mechanizmus nám teda zabezpečuje, že notifikačné správy o predĺžení životnosti budú od každého potomka prichádzať najviac jedenkrát za dobu expiračnej rezervy platnej pre danú adresárovú hĺbku. Túto garanciu dostávame za cenu toho, že po obmedzenú dobu môžu v adresárovej štruktúre existovať cesty, ktoré nevedú k súboru. Tu sa však ukazuje význam exponenciálne inverzného vzťahu medzi hodnotou expiračnej rezervy a hĺbkou adresárovej štruktúry. Po zániku posledného súboru v rámci podstromu hĺbky i potrvá najdlhšie $2^{1-i} \cdot r$, kým zaniknú adresáre celého podstromu.

3.3 Identita užívateľa

3.3.1 Požadované vlastnosti

V projekte Distropine si kladieme za úlohu simulovať vlastnosti diskusného fóra typu 'webový server' v prostredí P2P siete postavenej na distribuovanej hašovacej tabuľke. Na jednej strane sa od systému požaduje maximálna decentralizácia, anonymita tvorcov obsahu, nemožnosť monitorovania aktivít na sieti z jediného miesta atď.

Na druhej strane chceme eliminovať mieru anarchie tým, že užívateľ má v systéme svoju idenitu. Malo by ísť o identitu v klasickom ponímaní užívateľského účtu:

1. užívateľ v systéme vystupuje pod svojim pseudonymom, ktorý si zvolil pri registrácii — týmto pseudonymom sú označené komentáre, ktorý do systému vkladá

2. útočníkovi je znemožnená krádež identity iného užívateľa, tj. útočník nie je schopný

   - generovať komentáre označené pseudonymom obete,
   - pozmeňovať nejaké už existujúce komentáre vytvorené obeťou

3. identita užívateľa, pod ktorou publikuje obsah do P2P siete nie je zviazaná s uzlom, prostredníctvom ktorého sa v danej chvíli do siete pripája.

Pri zavádzaní týchto požiadaviek musí dizajn aplikácie vyriešiť nasledovné problémy:

- V DHT je obsah po vytvorení uložený distribuovane a v niekoľkých replikách medzi uzlami. Pôvodný autor teda od momentu publikácie komentára nemá žiadnu kontrolu nad jeho prípadným pozmeňovaním. V tejto situácii, keď nemáme možnosť *zamedziť* pozmeňovaniu dát, môžme urobiť takúto zmenu *detekovateľnú*.

- Z uvedeného zoznamu požiadaviek plynie aj to, že je potrebné zaviesť nejaký druh *autentizácie* — užívateľ musí dokázať nejakú znalosť alebo schopnosť a až potom je mu umožnené vystupovať pod želaným pseudonymom. Je však nutné vyhnúť sa zavádzaniu akéhokoľvek centrálneho prvku, voči ktorému by autentizácia prebiehala — takýto prvok (fakticky autentizačný server) by porušoval náš záväzok decentralizácie, okrem iného by mal aj príliš veľkú znalosť o aktivitách užívateľov siete, teda aj schopnosť monitorovať ich.

3.3.2 Nasadenie PKI a CA

Riešenie požiadaviek predchádzajúcej sekcie spočíva v nasadení kryptografie. V rámci P2P siete sa nasadí PKI[15] (Public Key Infrastructure). Následne bude možné pomocu digitálneho podpisovania chrániť integritu dátových položiek DHT, zabezpečiť nepopierateľnosť a autentizáciu.
Využitím kryptografie sa požiadavka *útočníkovi je znemožnená krádež identity* napĺňa tým, že pre krádež identity je nutné investovať obrovské výpočtové a časové zdroje.
Zabezpečíme teda, aby každý užívateľ vlastnil verejný a tajný kľúč podpisovacej schémy. Dátovú položku, ktorú užívateľ vkladá do DHT podpíše svojim tajným kľúčom.
Pri overovaní platnosti podpisu dátovej položky bude overujúci subjekt potrebovať verejný kľúč užívateľa, ktorý sa vydáva za autora. Autenticitu tohto verejného kľúča, resp. jeho väzbu s pseudonymom bude dokazovať *certifikát*. Ako popisuje literatúra [15], do úvahy pripadajú dva modely:

- PKI s certifikačnou autoritou (CA).

- PKI na báze siete dôvery (web of trust).

Pri návrhu aplikácie Distropine bol zvolený prvý prístup. V úvode sme vytýčili požiadavku vytvoriť riešenie, ktoré je maximálne decentralizované za predpokladu dosiahnutia funkcionality a bezpečnosti, ktorá je štandardná pre riešenia typu klient-server. Prečo sme teda neuprednostnili plne decentralizovanú variantu?

Registrácia. Preferujeme jednoduchú registrácie do siete. Užívateľ nepotrebuje byť pozývaný. Od začiatku dôveruje jedinej entite, tj. CA a vďaka prenosu dôvery následne všetkým certifikátom — môže teda overiť podpis každého diskusného príspevku od ľubovoľného užívateľa P2P siete.

Údržba dôvery. Pre užívateľa aplikácie na diskutovanie webových stránok by mala byť bezpečnostná vrstva plne transparentná. Nemožno od užívateľa očakávať, že bude ochotný venovať sa rozširovaniu sady verejných kľúčov, ktorým dôveruje apod.

Certifikáty. Po načítaní komentárov diskusného vlákna nie je prípustné začať na DHT vyhľadávanie certifikátov jednotlivých autorov komentárov. Certifikát autora musí byť teda súčasťou každého komentára. Mala by to byť dátová položka čo najmenšej veľkosti. Ak by certifikát obsahoval podpisy viacerých užívateľov, ktorí dôverujú danému autorovi, táto podmienka by nebola splnená.
Riešenie na báze CA sa teda v danej aplikácii ukazuje ako nutný kompromis. V rámci neho sa budeme snažiť, aby bola CA potrebná výlučne pri procese vystavovania certifikátov, nemala právomoci cenzurovať obsah alebo monitorovať právomoci užívateľov. Pokúsime sa zaviesť takú formu redundancie, aby registrácia nových užívateľov nezávisela od behu jediného CA uzlu v rámci siete.

3.3.3 Dátové štruktúry certifikátu a komentára

Certifikát

Certifikát je dátová položka, ktorá má sémantiku potvrdenia väzby medzi určitým pseudonymom *nickname* a určitým verejným kľúčom PK. Toto potvrdenie sa vydáva na presne stanovené obdobie a je podpísané certifikačnou autoritou.
Registrácia je žiadosť voči CA o vydanie certifikátu. Užívateľ v rámci registrácie predkladá požadovaný pseudonym, verejný kľúč a dôkaz, že k tomuto verejnému kľúču vlastní tajný kľúč. CA musí garantovať, že nevystaví na jeden pseudonym dva certifikáty s rôznymi verejnými kľúčmi, ktorých platnosť by sa prekrývala.
V rámci záväzku, že u CA budeme minimalizovať moc a potenciál monitorovať užívateľov sa pokúsme zredukovať informácie, ktoré sa CA pri registrácii dozvedá. Môžeme zamedziť tomu, aby sa CA dozvedela pseudonym užívateľa, ktorého registruje. Dosiahneme to tým, že certifikát nebude obsahovať reťazec *nickname*, ale jeho kryptograficky hašovanú hodnotu $h(\mathit{nickname})$. Pseudonym v otvorenom tvare sa bude nachádzať výlučne v komentároch. Overovateľ validity komentára teda skontroluje, či zahašovaná hodnota v certifikáte zodpovedá otvorenej hodnote v komentári.

Výsledná podoba dátovej štruktúry certifikátu je nasledovná:

$$\begin{aligned} hashnick &= h(nickname), \\ content &= CertificateContent(hashnick, PK_U, validFrom, validTo, cmtDurability), \\ certificate &= Certificate(content, sign_{SK_{CA}}(content)) \end{aligned}$$

kde *nickname* je pseudonym užívateľa (vlastníka certifikátu), PK_U je jeho verejný kľúč, dátumy *validFrom* a *validTo* sú v UTC a vymedzujú dobu platnosti certifikátu, *cmtDurability* je životnosť, ktorú budú mať komentáre publikované vlastníkom certifikátu. Tieto údaje sú podpísané tajným kľúčom CA (SK_{CA}), čím vzniká výsledný certifikát *certificate*.
Certifikát nie je dátová položka, ktorá sa ukladá priamo na DHT. Vždy je súčasťou inej položky, napr. komentára.

Komentár

Ako už bolo popísané užívateľ prikladá svoj certifikát ku každému komentáru, ktorý vytvára. Túto dvojicu následne podpisuje svojim tajným kľúčom a publikuje do DHT. Definitívna štruktúra komentára je nasledovná:

$$\begin{aligned} content &= CommentContent(nickname, url, body, publishDate, certificate), \\ comment &= Comment(content, sign_{SK_U}(content)) \end{aligned}$$

kde *nickname* je pseudonym autora v otvorenom tvare, *url* je URL webovej stránky, ku ktorej sa komentár vzťahuje, *body* je text diskusného príspevku, *publishDate* je dátum vytvorenia komentára v UTC a *certificate* je certifikát autora. Uvedené údaje sú podpísané autorovým tajným kľúčom SK_U, čím vzniká výsledný komentár *comment*.
Subjekt overujúci platnosť komentára postupuje v týchto krokoch:

1. základné kontroly: validita URL, komentár nie je z budúcnosti (*publishDate*), komentáru neuplynula životnosť (*publishDate* a *cmtDurability*),
2. $h(nickname) = hashnick$,
3. overenie validity certifikátu: aktuálny dátum patrí do intervalu platnosti certifikátu, podpis CA je valídny (voči známemu PK_{CA}),
4. overenie platnosti autorovho podpisu (voči PK_U, ktorý získa z certifikátu).

Životnosť komentára

Dátová štruktúra komentára ktorú sme definovali v tejto sekcii sa od definície súborového záznamu zo sekcie 3.2.1 odlišuje oi. v tom, že neobsahuje dátum expirácie, položku *expirationDate*. Životnosť dátovej položky samozrejme nemôže byť ľubovoľne volená jej autorom. Túto hodnotu preto stanovuje CA a udáva ju v certifikáte daného užívateľa (položka *cmtDurability*). Dátum expirácie komentára je teda dátum publikácie (položka *publishDate*) posunutým o životnosť.
Doba, na ktorú nastavuje CA životnosť komentárov je nastaviteľná u CA. V aplikáciách, keď je sieť Distropine zamýšľaná ako diskusný *chat*, môže byť táto položka nastavená na niekoľko hodín, v iných situáciách kde by sa jednalo napr.

o recenzie, môže byť životnosť komentárov nastavená na neobmedzenú dobu. To, že životnosť komentára nie je globálny parameter systému navyše umožňuje v budúcnosti implementovať logiku na základe ktorej bude CA udeľovať rôzne hodnoty životnosti komentára pre rôznych užívateľov.

3.3.4 Identifikačné karty

Užívateľ si pri registrácii vygeneroval verejný a tajný kľúč (PK_U a SK_U) a od CA získal certifikát (spoločný výraz: *autentizačné dáta*). Tieto údaje je potrebné uchovávať medzi užívateľovými prihláseniami do systému, pretože bez nich nedokáže vytvárať komentáre.
V časti 3.3.1 sme si uviedli požiadavku, aby bola identita užívateľa nezávislá od uzlu, z ktorého sa do P2P siete pripája. To znamená, že autentizačné dáta nemôžu byť súčasťou konfigurácie konkrétneho uzlu. Je potrebné vytvoriť autentizačný mechanizmus, v ktorom užívateľ poznajúci svoj pseudonym a heslo získa svoje autentizačné dáta z ľubovoľného uzlu systému.
Elegantným riešením je uložiť tieto dáta priamo do DHT. Vytvorme za týmto účelom novú dátovú položku, *identifikačnú kartu* (IDC), ktorá bude kontajnerom autentizačných dát určitého užívateľa. Pre zaistenie požadovanej autentizácie typu pseudonym-heslo potrebujeme, aby DHT adresa IDC bola funkciou užívateľovho pseudonymu. Priebeh autentizácie je potom nasledovný: užívateľ sa pripojí k DHT uzlu, ktorý je zatiaľ v anonymnom režime, zadá svoj pseudonym, v DHT sa vyhľadá príslušná IDC a autentizačné dáta sa z nej získajú pomocou daného hesla.
Jediným kritickým údajom v IDC, ktorý potrebujeme chrániť heslom je užívateľov tajný kľúč SK_U. Zvyšné autentizačné dáta môžu byť prítomné v otvorenej podobe. IDC musí byť zároveň podpísaná svojím autorom, aby bol ľubovoľný uzol schopný z IDC určiť, či nie je narušená jej integrita. Certifikát teda v IDC nie je len vo funkcii autentizačného údaja, ale je nutný aj pre overenie autorovho podpisu samotnej IDC.
IDC je teda nasledovná dátová štruktúra:

$$\begin{aligned} encrypted &= E_{K(pwd,salt)}(SK_U), \\ privatePackage &= PrivatePackage(certificate, encrypted, salt), \\ identityCard &= IdentityCard(privatePackage, sign_{SK_U}(privatePackage)) \end{aligned}$$

$$key(identityCard) = certificate.hashnick$$

kde

- *pwd* je heslo, pomocou ktorého sa odvodí kľúč pre symetrický šifrovací algoritmus E.
- *salt* je soľ, tj. náhodné dáta pre techniku solenia hesiel.
- *encrypted* je utajená časť IDC: obsahuje zašifrovaný tajný kľúč užívateľa.
- *certificate* je certifikát, ktorý užívateľovi udelila CA.
- *privatePackage* je vnútorná časť IDC, ktorá je podpisovaná.

- *identityCard* je samotná IDC v tvare, v ktorom bude uložená do DHT.

Všimnime si, že uzol držiaci IDC nedokáže o jej vlastníkovi získať takmer žiadne informácie, vrátane jeho pseudonymu.

3.3.5 Expirácia certifikátu a jeho obnovenie

Ako už bolo uvedené, platnosť certifikátu je časovo obmedzená dátumom expirácie. Po tomto dátume sú komentáre, ku ktorým je certifikát priložený, považované za neplatné.
Platnosť certifikátu určuje aj platnosť identifikačnej karty, ktorá ho obsahuje. Znamená to, že po expirácii certifikátu sú z DHT odstránené nielen komentáre ale aj IDC daného užívateľa a ten sa nemôže autentizovať. Tým je fakticky zo systému odstránený.
Certifikát, ktorému sa blíži doba expirácie[7] je potrebné obnoviť. Kedykoľvek, keď sa užívateľ v tomto období prihlási do P2P siete, jeho uzol zareaguje na expirujúci certifikát v jeho IDC tým, že sa pokúsi kontaktovať CA a požiadať o *obnovenie certifikátu*. Znamená to, že CA vydá nový certifikát, ktorého platnosť začína momentom prijatia požiadavky o obnovenie.
Tu sa ponúka priestor na implementovanie logiky, ktorá by certifikačnej autorite umožňovala mať kontrolu nad užívateľmi aj potom, čo ich zaregistruje.
Inštancia CA má nastaviteľnú dobu životnosti, ktorú priraďuje certifikátom pri ich prvom vydaní, tj. pri registrácii. Pri každej ďalšej požiadavke o obnovenie daného certifikátu, mu CA priradí dlhšiu životnosť[8]. To, že užívateľ požiadal o obnovenie certifikátu po uplynutí väčšiny jeho životnosti dáva opodstatnenie predpokladu, že na P2P sieti zostane aktívny aj naďalej.

3.3.6 CA ako užívateľ

Zo zatiaľ uvedeného by sa mohlo zdať, že CA je špeciálne konfigurovaný DHT uzol, ktorý vo svojej konfigurácii drží predovšetkým tajný kľúč SK_{CA} pre podpisovanie certifikátov. Vďaka zavedeniu systému identifikačných kariet však existuje elegantnejšie riešenie.
Prehlásime CA za užívateľa systému. Štandardným užívateľom bude v tom zmysle, že svoje autentizačné údaje bude mať uložené v IDC, ktorá je voľne uložená na DHT. Neštandardným užívateľom bude v tom zmysle, že jeho verejný kľúč poznajú všetky uzly systému a jeho certifikát je podpísaný ním samým (self-signed).
Keď sa na ľubovoľný uzol systému prihlási užívateľ CA (tj. stiahne sa z DHT jeho IDC a zadá sa správne heslo), uzol to detekuje a spustí procesy určené pre odbavovanie agendy CA.
Úlohu uchovať v tajnosti tajný kľúč SK_{CA} sme teda previedli na úlohu utajenia príslušného hesla k IDC. Vzhľadom na odolnosť šifrovacích metód použitých pre zabezpečenie IDC sme neznížili mieru zabezpečenia SK_{CA}. Získali sme možnosť ľahko delegovať ľubovoľný uzol systému do funkcie CA.
Najväčšou výhodou však je, že vo funkcii CA môže byť v jednej chvíli delegovaných niekoľko uzlov súčasne (tzv. *inštancií CA*). To prináša škálovateľnosť a zvýšenú redundanciu.

[7] Presnejšie, ubehnú 2/3 jeho životnosti.
[8] V súčasnej implementácii o 50% dlhšiu životnosť.

Problémom, ktorý je potrebné vyriešiť je to, ako sa ostatné DHT uzly dozvedia, ktoré uzly (tj. na akých IP alebo DHT adresách) sú momentálne inštanciami CA.

Inštancie CA na zvláštnych adresách DHT

Relatívne priamočiarím riešením by bolo, aby inštancie CA boli uzly so zvláštnym umiestnením v adresovom priestore DHT, takže všetky ostatné uzly systému by vedeli, kde ich hľadať. Napr. adresy by boli z určitej známej množiny alebo by sa nachádzali v blízkosti vopred známej adresy.
Metódy založené na tomto princípe nie sú použiteľné. Na DHT vrstve sme v časti 2.2.6 zvolili takú metódu generovania DHT adries pre uzly, že uzol nedokáže ovplyvniť svoju pozíciu v adresovom priestore DHT. Adresa uzla (tj. jeho nodeID) je v tvare $h(PK)$, tj. je výstupom kryptografickej hašovacej funkcie z verejného kľúča, ku ktorému musí uzol vlastniť kľúč tajný a musí mať vyriešenú tzv. kryptohádanku (viď 2.2.6, útok Eclipse a Sybil). Tento prístup sme nasadili ako protiopatrenie voči niektorým typom útokov voči DHT.

Koncept *Message of the day.*

Zaveďme novú dátovú položku MOTD (*Message of the day*). Tieto položky by bola generovaná a podpisovaná výlučne inštanciami CA a to v pravidelných intervaloch (napr. 1 hod). Adekvátna tomu by bola ich krátka životnosť.
Položky MOTD by boli prostriedkom pre hromadnú notifikáciu uzlov systému zo strany CA. V momentálne preberanom kontexte je dôležité, že každá položka MOTD by obsahovala DHT adresu uzlu, ktorý ju vygeneroval. Ak by bolo v systéme niekoľko inštancií CA, potom by na jednej adrese ležalo niekoľko položiek MOTD (podobne, ako leží viacero komentárov na jednej DHT adrese vlákna). Ďalším príkladom obsahu MOTD môže byť CRL, tj. zoznam revokovaných certifikátov, viď nižšie.
Štruktúra dátovej položky MOTD:

$$\begin{aligned} content &= MotdContent(publishDate, expirationDate, caNodeID, data), \\ motd &= Motd(content, sign_{SK_{CA}}(content)) \end{aligned}$$

kde dátumy *publishDate* a *expirationDate* sú v UTC a určujú dobu platnosti položky, *caNodeID* je DHT adresa inštancie CA, ktorá danú položku vygenerovala, *data* je priestor na ďalšie dáta (napr. CRL).
Je potrebné vyriešiť otázku priradenia DHT kľúča položkám MOTD. Ak by sa zvolila konštantná hodnota[9], je treba počítať s tým, že by sa príslušná DHT adresa stala atraktívnym cieľom útokov a tiež s tým, že pri narastajúcom počte uzlov systému by boli uzly spravujúce túto adresu nadmerne zaťažené.
Hľadajme teda iné metódy.

Kešovanie. Položky MOTD by získali zvláštny štatút kešovanej dátovej položky. Ostatné dátové položky sú medzi DHT uzlami udržiavané striktne tak, aby sa vyskytovali v k replikách a nie v omnoho väčšom počte. Pre kešovanú dátovú položku toto obmedzenie neplatí a ľubovoľný uzol, ktorý ju získa ju udržiava do

[9] Tento prístup je implementovaný v súčasnej verzii DHT Kademlia.

doby jej expirácie. Uzol ju môže získať tak, že ju proaktívne vyhľadal (operáciou *value_lookup*) alebo tak, že sa kedysi nachádzal v jej blízkosti, ale vplyvom zvýšenia počtu uzlov v danej oblasti adresového priestoru DHT už nepatrí medzi jej k najbližších uzlov.
Tento princíp je účinný, pretože lookup dotazy v DHT prebiehajú na princípe približovania sa k danej adrese. Na druhú stranu, vlastnosťou kešovania je, že by boli najviac rozšírené najstaršie (zatiaľ neexpirované) položky, čo nie je v kontexte MOTD optimálne.

Redundantné uloženie. Pri tejto metóde by dátové položky MOTD boli ukladané redundantne na viacero DHT adries, pričom ich počet (tj. miera redundancie) by bol funkciou počtu uzlov celého DHT systému. Pre túto funkciu by mohol postačovať nasledovný triviálny vzťah: $f(n) = \lfloor n/m \rfloor + 1$, kde napr. $m = 1000$.
Túto metódu možno implementovať vďaka tomu, že každý uzol DHT Kademlia dokáže z obsahu svojej smerovacej tabuľky (tj. k-kapes) odhadnúť celkový počet uzlov tvoriacich DHT. Predpokladom správnosti týchto odhadov je to, že uzly majú DHT adresy pridelené rovnomerne, takže uzol môže predpokladať, že sa nachádza v časti DHT s priemernou hustotou uzlov.
Tento odhad počtu uzlov systému by urobila inštancia CA pred uložením novej položky MOTD na DHT. Ak by došla k odhadu n, potom by pre každé i, $i \in \{1, .., f(n)\}$, uložila do DHT jednu kópiu na adresu $h(i)$.
Ľubovoľný uzol, hľadajúci položky MOTD by urobil podobný odhad počtu uzlov systému, čím by došiel k číslu n'. Potom by vygeneroval náhodné číslo i, $i \in \{1, \ldots, f(n')\}$. Dátovú položku by následne vyhľadával na adrese $h(i)$. Ak by neuspel, znamená to, že jeho odhad bol nadhodnotený, alebo že sa žiadna položka MOTD v DHT nenachádza. Preto by zopakoval postup pre $n' = \lfloor n'/2 \rfloor$ (ak $n' \geq 2$).
Maximálny počet pokusov by však mal byť konštantný. Schopnosť odhadnúť počet uzlov systému je dostatočne dobrá nato, že v systémoch s veľkým počtom uzlov $\bar{n}$, $\bar{n} \gg m$, nemá zmysel deliť interval až do stavu $n' = 1$, pretože v prípade neexistencie položky MOTD by došlo k preťaženiu uzlov na *nízkych* adresách $h(0), h(1), \ldots$

3.4 Plánované rozšírenia

V predchádzajúcich sekciách boli popísané riešenia, ktoré sú implementované v aktuálnej verzii projektu Distropine. Ostáva niekoľko dôležitých úloh, ktoré nie sú implementované, no návrh a dizajn aplikácie s nimi počíta. Sú popísané v nasledujúcich sekciách.

3.4.1 Zavedenie reputácie užívateľov

CA „odmeňuje" užívateľov, ktorí sú registrovaní v systéme dlhšiu dobu tým, že ich certifikáty majú dlhšiu životnosť (viď 3.3.5). Z každého komentára teda možno určiť, nakoľko je jeho autor stabilným užívateľom. Už vďaka tejto elementárnej informácii by bolo možné v GUI zvýrazňovať príspevky dlhodobých užívateľov, príp. dať krátkodobým užívateľom len obmedzený priestor v danom diskusnom vlákne. Uzly prerozdeľujúce obmedzenú kapacitu svojej lokálnej databázy by

pri potrebe uvoľniť miesto prioritne odstraňovali komentáre krátkodobých užívateľov.
Ďalším krokom ako zaviesť reputačný systém v systéme Distropine by bolo umožniť užívateľom pozitívne alebo negatívne ohodnocovanie komentárov. V diskusných fórach typu klient-server je to štandardná funkcionalita.
Toto ohodnotenie príspevku by znamenalo, že hodnotiaci užívateľ by na DHT uložil špeciálnu dátovú položku, *hodnotenie*. Tá by obsahovala hodnotiacu známku komentára (napr. +/-1 alebo nahlásenie spamu), referenciu na hodnotený komentár a certifikát hodnotiaceho užívateľa. Celá dátová položka by bola podpísaná. Hodnotenia by sa v DHT ukladali na rovnakú adresu ako IDC hodnoteného užívateľa, tj. $h(nickname)$.
Pri obnovovaní certifikátu by teda CA stiahla z DHT hodnotenia príslušného užívateľa. Výsledné reputačné skóre by spočítala na základe známok, ktoré by boli vážené reputáciami hodnotiacich užívateľov. Výsledok by CA uložila do obnoveného certifikátu. V prípade zlej reputácie by mohla vydanie nového certifikátu odmietnuť.
Vyššie uvedený prepočet reputácie je síce priamočiary, ale príliš zraniteľný voči viacerým typom útokov (resp. spôsobov ako ho oklamať za účelom ovplyvnenia svojho reputačného skóre). Zavedenie reputácie (obzvlášť v P2P systémoch) nie je triviálny problém a pred nasadením by vyžadoval ďalšie oboznámenie sa s touto problematikou.

3.4.2 Revokácia

Definícia v kontexte systému Distropine

Revokácia certifikátu znamená jeho zneplatnenie v situácii, keď hrozí, že bol odcudzený k nemu príslušný tajný podpisový kľúč[10]. Znamená to, že (potenciálne) existuje útočník, ktorý je schopný podpisovať sa v mene užívateľa. Bez mechanizmu revokácie by útočník mohol využívať prenesenú dôveru, ktorú zabezpečuje certifikát a to až do doby jeho expirácie.
Po revokovaní certifikátu je potrebné vygenerovať nové podpisové kľúče a požiadať CA o vydanie nového certifikátu, ktorý zviaže pôvodnú identitu a nový verejný kľúč. V tomto procese musí byť ošetrené, ako sa voči CA autentizuje pôvodný majiteľ certifikátu (tj. obeť potenciálneho útoku), aby nemohlo dôjsť k tomu, že o revokáciu prvý požiada útočník, získa od CA nový certifikát a tým sa jeho falošná identita zlegitimizuje.

Popis riešenia

V aktuálnej verzii projektu *nie je* implementovaný žiadny mechanizmus pre revokáciu certifikátov. Je pritom samozrejmé, že akákoľvek implementácia PKI bez tohto mechanizmu má už z princípu veľkú bezpečnostnú trhlinu. Prejdime si teda spôsoby ako by mohla byť revokácia v tomto projekte riešená.
Dôvod na revokáciu má užívateľ systému Distropine vtedy, keď existuje možnosť, že bolo odhalené heslo, ktoré v jeho identifikačnej karte chráni jeho tajný podpisový kľúč. V tejto situácii nestačí vygenerovať novú IDC s pozmeneným heslom.

[10] Iným prípadom je situácia, kedy útočník získal od CA podpísaný certifikát na identitu, ktorá mu nepatrí. Tento a ďalšie dôvody revokácie sú však mimo náš aktuálny záujem.

Môže byť neskoro, pretože útočník už mohol kľúč SK_U získať a IDC už nepotrebuje. Navyše v DHT nikdy nie je isté, že sa v budúcnosti nepripojí uzol, ktorý vo svojej lokálnej databáze obsahuje starú verziu IDC (tj. pred zmenou hesla). Z týchto dôvodov aplikácia Distropine neobsahuje možnosť zmeny hesla[11], i keď z implementačného hľadiska by sa jednalo o triviálnu úlohu.
Pri riziku odcudzenia tajného kľúča musí byť teda užívateľov kľúčový pár zmenený a jeho pôvodný certifikát zneplatnený. Za týmto účelom musí existovať tzv. *zoznam revokovaných certifikátov* (CRL) a ten musí byť prístupný všetkým uzlom systému. Súčasťou kontroly validity certifikátu musí byť okrem overenia podpisu CA aj kontrola, či certifikát nie je referencovaný v CRL.
Záznam v CRL môže obsahovať aj dátum predpokladaného úniku tajného kľúča. V takom prípade by neboli zneplatňované komentáre, ktoré vznikli pred týmto dátumom. Revokačný záznam ostáva v CRL až do doby expirácie príslušného certifikátu.
V súlade so záväzkom minimalizovať schopnosť CA cenzurovať obsah siete jej musíme znemožniť revokovať certifikáty samovoľne. Preto bude valídny revokačný záznam podpísaný revokujúcim užívateľom.
Navrhneme dve metódy zavedenia revokácie. Budú sa líšiť v tom, či je CA pri procese revokácie kontaktovaná alebo nie. Čo sa týka spôsobu autentizácie držiteľa certifikátu, je v oboch prípadoch predpokladaná existencia tzv. revokačného hesla pwd_2. Predpokladá sa, že toto heslo útočník nepozná.
Nebolo by vhodné ak by si pwd_2 musel užívateľ pamätať (pravdepodobne by ho zvolil totožné s hlavným IDC heslom). Lepšia varianta je taká, že pri registrácii je pwd_2 vygenerované užívateľovým uzlom ako náhodný reťazec. Užívateľ následne zadá svoju e-mailovú adresu a jeho uzol mu toto heslo prepošle.

Metóda s kontaktovaním CA. CA pri registrácii vygeneruje tzv. *priepustku*. Priepustka vznikne tak, že CA na nejaké známe dáta (napr. na certifikát užívateľa) aplikuje kryptografickú operáciu, ktorú protistrana nedokáže zopakovať, napr. šifrovanie alebo aplikáciu MAC funkcie s kľúčom známym len CA. CA priepustku vráti užívateľovi a ten ju vloží do svojej IDC v zašifrovanej forme použijúc pwd_2.

Revokácia:

1. Užívateľ sa voči CA autentizuje priepustkou a pôvodným vzorom, z ktorého vznikla. Okrem toho si vygeneruje nový pár podpisových kľúčov (PK'_U a SK'_U). Odošle CA štruktúru *signedCrrContent*, ktorá obsahuje jeho pôvodný certifikát (*oldCert*), nový verejný kľúč a aktuálny dátum:

$$crrContent = CrrContent(oldCert, PK'_U, date),$$
$$signedCrrContent = SignedCrrContent(crrContent, sign_{SK'_U}(crrContent))$$

   Podpisom štruktúry podáva užívateľ dôkaz, že súhlasí s revokáciou a zároveň dokazuje, že pre verejný kľúč PK'_U skutočne vlastní tajný kľúč SK'_U.

[11] Prístup v súlade s princípom: „radšej žiadna bezpečnosť, než bezpečnosť zdanlivá".

2. CA si overí platnosť priepustky, tj. znova na predlohu aplikuje svoju kryptografickú operáciu. Ak dôjde k zhode, overí validitu zaslanej položky (platnosť podpisu, certifikátu...) a následne ju podpíše. Tým vzniká úplný revokačný záznam (crr), ktorý vloží do DHT na adresu CRL.

$$crr = Crr(signedCrrContent, sign_{SK_{CA}}(signedCrrContent)),$$

CA podpíše a odošle nový certifikát, kde zviaže pseudonym použitý v pôvodnom certifikáte ($oldCert$) a nový verejný kľúč (PK'_U).

Metóda bez kontaktovania CA. Pri tejto metóde užívateľ už pri registrácii žiada o dva certifikáty vystavené na ten istý pseudonym: *hlavný* a *rezervný* certifikát.
CA oba certifikáty podpisuje. Navyše posiela aj *revokačný záznam*, ktorý je ňou podpísaný a referencuje hlavný certifikát.
Dáta „pre prípad revokácie" teda tvorí: rezervný certifikát podpísaný CA, k nemu príslušný tajný podpisový kľúč a revokačný záznam podpísaný CA. Tieto dáta sa zašifrujú heslom pwd_2 a vložia sa do IDC.
V momente, keď uzol potrebuje revokovať, má všetky potrebné údaje vo svojej IDC za predpokladu, že pozná heslo pwd_2. Rezervný certifikát a príslušný tajný kľúč prehlási za svoju novú identitu. Do CRL sám publikuje revokačný záznam, ktorý navyše opatrí dátumom revokácie, svojim rezervným certifikátom a celé to podpíše. Následne na DHT vloží aj novú IDC, v ktorej už samozrejme chýba rezervný certifikát.
Nevýhodou tohto riešenia je, že revokáciu možno v rámci danej životnosti certifikátu vykonať najviac raz. Samozrejme, tento postup možno rozšíriť tak, aby sa udržiaval viac než jeden rezervný certifikát.

4. Užívateľské rozhranie

Aplikácia Distropine je distribuované riešenie problému, ktorý je typicky riešený modelom klient-server. Jedným z cieľov pri jej návrhu bolo odtieniť užívateľa od tohto faktu a poskytnúť mu službu, na ktorú je navyknutý. Vedľa bezpečnosti a ochrany identity je v tomto ohľade dôležité aj užívateľsky-prívetivé GUI.
Z tohto dôvodu nemá aplikácia štandardné formulárové GUI. Najprirodzenejším riešením je integrácia GUI priamo do rozhrania webového prehliadača, takže užívateľ môže komentovať stránky z toho istého prostredia, v rámci ktorého stránky prehliada. O GUI pojednáva sekcia 4.1.
Na druhej strane je potrebné zachovať možnosť nízkoúrovňového prístupu k aplikácii. Pri ladení P2P aplikácií je výhodné v rámci jednej bežiacej inštancie programu simulovať sieť mnohých uzlov bežiacich paralelne a nezávisle na sebe. V prípade DHT je výhodné sledovať smerovacie tabuľky uzlov, distribúciu dátových položiek medzi uzly, replikáciu atď. Pre sprostredkovanie tejto funkcionality zaveďme druhé užívateľské rozhranie, ktoré bude textové. Toto rozhranie bude ďalej označované ako *konzolové*. Pojednáva o ňom sekcia 4.2.

4.1 GUI

4.1.1 Rozšírenie webového prehliadača

Integrácia aplikácie do prehliadača je umožnená pomocou systému tzv. rozšírení (*extensions*, *add-ons*), ktorý majú prakticky všetky súčasné webové prehliadače. Pre aplikáciu Distropine bol zvolený prehliadač *Google Chrome* práve kvôli jednoduchosti tvorby jeho rozšírení a jeho vzrastajúcej popularite.
Koncept rozšírení so sebou nesie nutnosť implementovať celé GUI ako HTML stránku a jej funkcionalitu programovať v jazyku JavaScript. Prepojenie GUI a hlavného programu je realizované tak, že hlavný program prevádzkuje HTTP server a JavaScriptový kód tento lokálny server dotazuje podľa potreby prístupom *Ajax*[1].
Použitie technológie *Ajax* umožňuje napríklad to, že pri zaslaní dotazu na komentáre nemusí stránka čakať na dokončenie celej DHT operácie. Komentáre sa postupne pridávajú do zoznamu tak, ako prichádzajú z DHT. Pohodlné je tiež to, že program na pozadí v pravidelných intervaloch dotazuje DHT na prípadné prírastky medzi komentármi. Vďaka tomu, že GUI realizuje voči hlavnému programu *pooling*, sa obsah diskusného vlákna obnovuje bez zásahu užívateľa. To vytvára dojem podobný dynamickým uživateľským rozhraniam moderných UGC služieb typu klient-server.

4.1.2 Dizajn

GUI aplikácie znázorňuje obr. 4.1.

[1] Ajax[18] (Asynchronous JavaScript and XML): asynchrónne dotazovanie HTTP servera z JavaScript kódu, ktoré umožňuje vytvoriť interaktívne rozhranie bez nutnosti znovunačítania celej stránky pri každom dotaze.

- Napravo adresového riadku prehliadača sa nachádza ikona aplikácie, po stlačení ktorej sa zobrazí plávajúce okno.
- V prvom riadku okna je uvedený pseudonym aktuálne prihláseného užívateľa (sonia) a odkaz pre jeho odhlásenie.
- Nasleduje zobrazenie štruktúry domén a v nej organizovaných diskusných vlákien. Kliknutie na názov domény vyvolá načítanie zoznamu jej poddomén a diskusných vlákien z DHT. Načítanie prebieha asynchrónne podobne ako načítavanie komentárov. Po jeho ukončení sa adresár príslušnej domény rozbalí.
- Kliknutie na názov vlákna vyvolá načítanie príslušnej webovej stránky v prehliadači. Okno aplikácie Distropine zmizne. Po opätovnom kliknutí na ikonu Distropine sa zobrazí užívateľské rozhranie zobrazujúce zmenené diskusné vlákno.
- Každý komentár obsahuje pseudonym užívateľa, čas vzniku a samotný text príspevku.
- V spodnej časti okna sa nachádza pole pre vloženie textu nového diskusného príspevku.

Uvedený obrázok popisuje program v stave, keď je prihlásený užívateľ. V prípade, že užívateľ prihlásený nie je, je program v *anonymnom režime*, kedy sú zobrazené odkazy k registrácii nového užívateľa a k prihláseniu existujúceho užívateľa. V tomto režime nie je možné vkladať komentáre.

4.1.3 Postup: Registrácia nového užívateľa

Dvojica obrázkov 4.2, 4.3 popisuje priebeh registrácie nového užívateľa. GUI je na začiatku v anonymnom režime. Užívateľ klikne na 'Register' a vloží želané užívateľské meno a heslo. Program na pozadí postupne nachádza na DHT zoznam inštancií CA a pokúša sa ich kontaktovať až kým u niektorej neuspeje. Ak registrácia prebehne v poriadku (CA vystavila certifikát atď.), je nový užívateľ prihlásený a môže vytvárať komentáre.

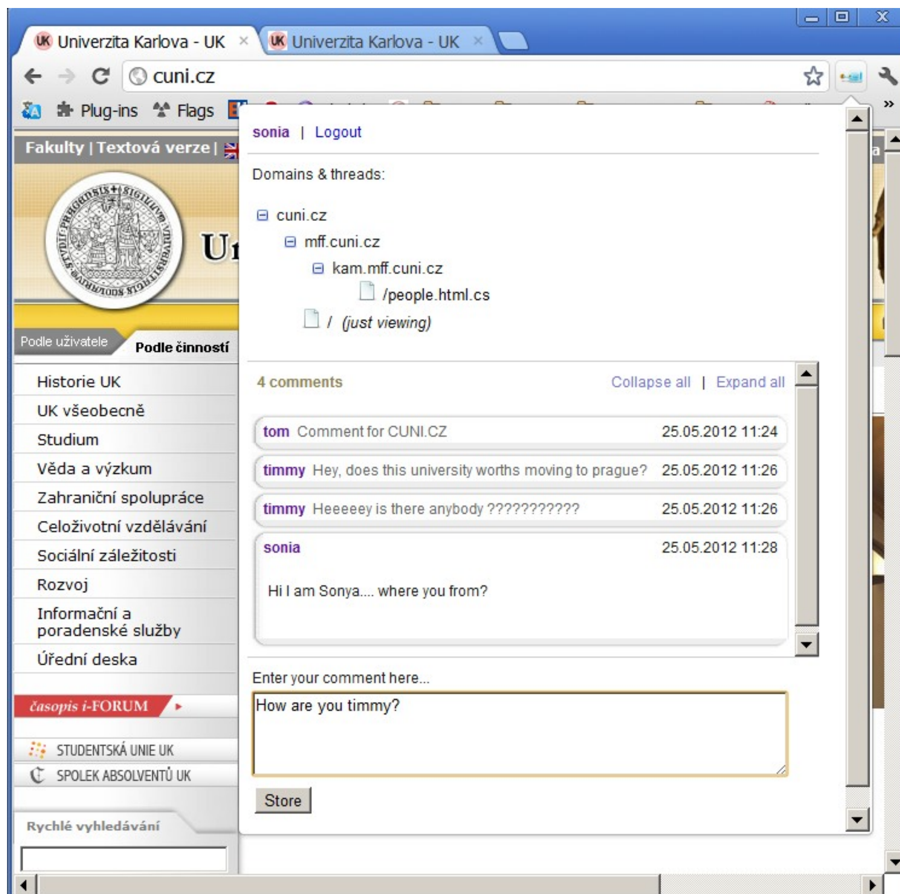
4.2 Konzolové rozhranie, konfigurácia

Hlavný program aplikácie Distropine reprezentuje ikona, ktorá je umiestnená v notifikačnej oblasti hlavného panelu Windows. Kliknutím na ňu sa zobrazí kontextové menu. Prostredníctvom neho možno program vypnúť alebo zobraziť (resp. skryť) konzolové prostredie.

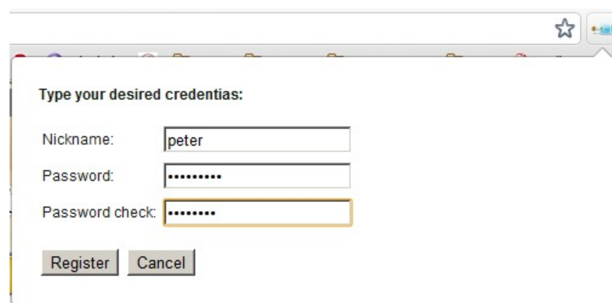
4.2.1 Prehľad funkcionality konzolového rozhrania

Uveďme základný prehľad toho, čo konzolové prostredie umožňuje:

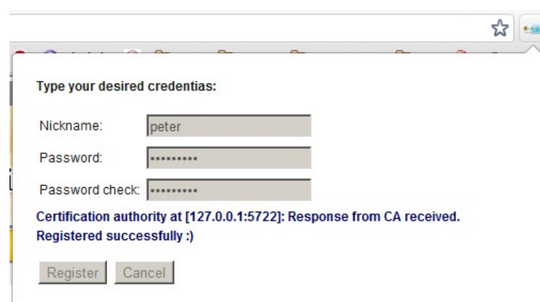
Všetko, čo je možné z GUI. Je možné dotázať DHT na komentáre a referenčné záznamy danej domény, vložiť do DHT nový komentár (`scmt`), prihlásiť (`lin`) a odhlásiť sa (`lout`) a zaregistrovať nového užívateľa (`reg`).



Obr. 4.1: GUI aplikácie Distropine integrované v Google Chrome



Obr. 4.2: Registrácia. Zadanie mena a hesla



Obr. 4.3: Registrácia. Bola nájdená CA, ktorá vystavila certikát

Beh viacerých inštancií. Program po spustení obsahuje jediný DHT uzol, označme ho n_0. Ďalšie uzly možno vytvoriť príkazom `new`, ktorý berie počet uzlov ako argument.
Ako je uvedené v úvode kapitoly, uzly bežia paralelne a nezávisle na sebe. Ich vzájomná komunikácia prebieha cez TCP adresu *localhost*.
V rámci konzoly je vždy jeden uzol aktuálny (nastavuje sa príkazom `i`). Voči tomuto uzlu sa vykonávajú ostatné príkazy. Daný príkaz však možno zavolať aj hromadne voči všetkým bežiacim uzlom (príkaz `all`).

Vnútorný stav uzlu. Je možné zobraziť obsah najdôležitejších dátových štruktúr uzlu: jeho smerovaciu tabuľku, tj. štruktúru k-kapes (príkaz `b`), obsah databázy dátových položiek (`lht`) a zoznam aktívnych RPC relácií (`sessions`).

Zmeny parametrov programu alebo konkrétneho uzlu. Príkazy z tejto sady umožňujú zmeniť replikačný interval dátových položiek (príkaz `rep`, viď 2.2.5), parameter paralelizmu algoritmu Kademlia (príkaz `alpha`, viď 2.2.3) a nastavenie úrovne logovania do logovacieho súboru (`loglevel`).

4.2.2 Konfigurácia programu

Konfigurácia programu Distropine sa nachádza v XML súbore `config.xml` a vzťahuje sa na všetky inštancie uzlov, ktoré v programe bežia.
Konfigurovateľné sú nasledovné parametre:

- verejný kľúč CA.
- zoznam DNS alebo IP adries bootstrap uzlov.
- konštanta k, tj. replikačný parameter algoritmu Kademlia.

Pre účely nasledujúceho príkladu predpokladajme, že konfiguračný súbor neexistuje, resp. je prázdny. V tom prípade založí konfiguračný súbor, do ktorého vygeneruje predvolenú konfiguráciu a použije ju.
V predvolenej konfigurácii je parameter replikácie k nastavený na hodnotu 7. Bootstrap uzol je nastavený na lokálnu adresu a UDP port 5721. Toto je zároveň port, na ktorý bude pripojený uzol n_0. To znamená, že všetky ostatné uzly v rámci bežiacej inštancie programu budú tento uzol využívať ako bootstrap uzol.
V prípade, keď v konfigurácii nie je uvedený verejný kľúč CA (to je aj prípad predvolenej konfigurácie), sa predpokladá, že je zakladaná nová sieť Distropine a toto je jej prvý uzol. V takom prípade je pri štarte aplikácie automaticky vytváraná nová identita CA. Generujú sa náhodné podpisovacie kľúče a užívateľ je v konzole vyzvaný na zadanie pseudonymu a hesla pre užívateľa CA. Na základe toho sa vytvára *self-signed* certifikát a identifikačná karta. Do konfiguračného súboru aplikácie sa zapisuje nový verejný kľúč CA.
Môžme predpokladať, že pri štarte z predvolenej konfigurácie neobsahujú DHT uzly vo svojich lokálnych úložiskách žiadne dátové položky. Aj keby nejaké obsahovali z predchádzajúceho behu aplikácie, tak v dôsledku vytvorenia novej identity CA, budú považované za neplatné všetky staré certifikáty a tým aj dátové položky.

4.2.3 Príklad

V nasledujúcom príklade bude demonštrovaná práca s konzolovým rozhraním. Program bude spustený s predvolenou konfiguráciou, bude vytvorená testovacia Distropine sieť, na ktorej sa spustí CA a voči nej bude registrovaný nový užívateľ. Do DHT následne vložíme na jeho účet nový komentár.

Spustenie programu

Po spustení programu je užívateľ vyzvaný na zadanie pseudonymu a hesla. Predpokladajme, že užívateľ pre CA zvolil pseudonym 'ca'. Následne sa vytvára uzol n_0 a zobrazí sa príkazový riadok.
Zadaním príkazu `info` sa zobrazia základné informácie o aktuálnom uzle:

```
0.0.0.0:5721/(A69C)
My nodeID long = 00000000A69C2F3A 3CC11F13443496FD 2548208FF4E6E3FC
Actual MyIdentity = []
GUI HTTP server at 127.0.0.1:5721
```

Z tohto možno výčítať, že uzol n_0 prijíma RPC správy na porte UDP 5721. Údaj za znakom '/' označuje jeho nodeID v skrátenej forme (horných 16 bitov) — týmto zápisom sú identifikované uzly aj vo výpisoch ostatných príkazov. Na druhom riadku je vypísané nodeID v úplnom tvare.
Tretí riadok uvádza, že momentálne nie je prihlásený žiaden užívateľ, tj. uzol je v anonymnom režime.
Posledný riadok informuje, že HTTP server pre komunikáciu s GUI beží na porte TCP 5721. V JavaScriptovom kóde HTML stránky GUI je tento port fixne nastavený. Takže aj vtedy, keď bude v programe súčasne vytvorených niekoľko uzlov, klient GUI sa bude pripájať vždy len na HTTP server uzlu n_0.

Generovanie ďalších inštancií uzlov

Zavolaním príkazu:

```
new 30
```

pridáme do testovacieho systému ďalších 30 uzlov (dostávame uzly n_0 – n_{30}). Dlhé čakanie na dokončenie tejto operácie je dôsledkom generovania kryptohádanky, ktorá je bezpečnostným protiopatrením voči *Sybil* útoku na DHT (viď 2.2.6).
Po vytvorení uzlov sa na každom z nich spúšťa bootstrap algoritmus. Ako už bolo uvedené, všetky uzly majú nastavený bootstrap uzol na n_0.
Príkazom `i 10` konzolu napojíme na uzol n_{10} a príkazom `b` zobrazíme obsah jeho štruktúry k-kapes. Tá môže byť zatiaľ prázdna, ak sa na uzle ešte nespustil bootstrap algoritmus. Inak bude výstup podobný nasledujúcemu:

```
#154 (1 contacts, 0 in cache)
        loc:5722/(1984)
#156 (1 contacts, 0 in cache)
        loc:5747/(09CE)
#157 (2 contacts, 0 in cache)
        loc:5736/(3A90)
```

```
        loc:5737/(A725)
#158 (7 contacts, 7 in cache)
        loc:5734/(09E6)
        loc:5721/(E3FC)
        loc:5751/(7823)
        ...
#159 (7 contacts, 5 in cache)
        loc:5735/(AC42)
        loc:5733/(44EA)
        loc:5723/(C6E1)
        ...
```

Je to zoznam všetkých neprázdnych kapes a k nim príslušných kontaktov (zoznamy sú skrátené). Vidíme napríklad, že k aktuálnemu uzlu je v adresovom priestore DHT najbližší uzol na porte 5722 (uzol n_1), ktorý s ním zdieľa adresový prefix dĺžky 5 bitov. Je to vidieť z toho, že sa kontakt na uzol n_1 nachádza v šiestej kapse, tj. kapse č. 154.
Pri každej kapse je uvedený aj počet kontaktov, ktoré sú momentálne v jej nahradzovacej cache (viď 2.2.2).

Prihlásenie CA

Príkaz `lht` zobrazuje výpis zoznamu dátových položiek, ktoré vo svojej lokálnej databáze drží aktuálny uzol. Zavolaním prikazu `all lht` dostávame vypis dátových položiek cez jednotlivé uzly.
Jediná dátová položka, ktorá sa v DHT zatiaľ nachádza je IDC certifikačnej autority, ktorá sa vytvorila pri spustení programu:

```
IDC nickhash=(13B0), [24.11.11 03:47]-[31.12.99 23:59]
```

Bezprostredne po vytvorení nových uzlov je táto dátová položka držaná len uzlom n_0. Postupne sa na uzloch spúšťa bootstrap algoritmus a dochádza k replikácii, takže položka by sa mala rozšíriť medzi uzly a nakoniec by sa jej výskyt mal stabilizovať na siedmich replikách.
Zmeňme teraz aktuálny uzol na taký, ktorý zatiaľ neudržiava žiadnu dátovú položku, nech to je napr. uzol n_{10} (voláme teda `i 10`). Pokúsime sa z tohto uzlu autentizovať do systému Distopine ako užívateľ 'ca'. Použijeme nasledovný príkaz:

```
lin ca heslo
```

Uzol n_{10} musí na DHT najprv vyhľadať príslušnú IDC, z ktorej po aplikovaní hesla získa certifikát a tajný kľúč. Po úspešnej autentizácii uzol rozpozná, že sa jedná o *self-signed* certifikát dôverovanej CA. Preberá teda funkciu inštancie CA. Ukladá na DHT dátovú položku MOTD, čim zverejňuje pre ostatné uzly informáciou, že je pripravený prijímať požiadavky o registráciu nových užívateľov. Uloženie položky MOTD je možné overiť zavolaním príkazu `all lht`.

Registrácia nového užívateľa

Zmeňme znovu aktuálny uzol, napr. na n_{20}. Pokúsme sa z tohto uzla zaregistrovať nového užívateľa s pseudonymom 'tom' a heslom '1234'. Do príkazového riadku

zadajme nasledovné:

```
reg tom 1234
```

Po vložení tohto príkazu uzol n_{20} najprv vyhľadáva na DHT dátové položky MOTD, z ktorých sa dozvie DHT adresu inštancie CA. Následne na základe známej DHT adresy zisťuje algoritmom *node_lookup* príslušnú TCP/IP adresu, ktorú potom kontaktuje s požiadavkou o registráciu. Inštancia CA, uzol n_{10}, po prijatí požiadavky uskutočňuje testovacie hľadanie identifikačnej karty užívateľa s rovnakým pseudonymom. Ak sa výskyt takejto IDC nepotvrdí, nič nebráni vo vystavení nového certifikátu aktuálnemu žiadateľovi.

Pridanie komentára

Aktuálnym užívateľom je teda užívateľ 'tom'. Vytvorme v jeho mene nový komentár:

```
scmt kam.mff.cuni.cz/people.html
PRVY KOMENTAR.
```

Ak hneď po vytvorení komentára budeme opakovane v krátkych intervaloch kontrolovať obsah DHT príkazom `all lht`, uvidíme postupné vznikanie dátových položiek, predovšetkým propagáciu adresárových referencií v smere zdola-hore. Dátové položky sa teda budú objavovať v tomto poradí:

```
CMT: kam.mff.cuni.cz/people.html     | tom    | PRVY%20KOMENTAR.
DIR: kam.mff.cuni.cz -> kam.mff.cuni.cz/people.html 20.05. 15:42
DIR: mff.cuni.cz -> kam.mff.cuni.cz                 20.05. 17:12
DIR: cuni.cz -> mff.cuni.cz                         20.05. 20:12
DIR: cz -> cuni.cz                                  21.05. 02:12
DIR: '' -> cz                                       21.05. 14:12
```

Poznamenajme, že tieto položky majú rozdielne DHT kľúče, preto pravdepodobne nebude existovať uzol, ktorý by všetky z nich udržiaval vo svojej lokálnej databáze. Na výpise si ešte všimnime dátumy na pravej strane u riadkov zobrazujúcich adresárové referencie. Jedná sa o ich dátumy expirácie. Rozdiely medzi hodnotami nad sebou sú rovné expiračnej rezerve v príslušnej hĺbke adresárovej hierarchie (viď 3.2.5, pričom $r_0 = 12$ *hod*).

Kontrola z GUI

Dostupnosť nového komentára a vytvorenie príslušnej adresárovej štruktúry možno tiež skontrolovať priamo z GUI: V Google Chrome zadáme URL `cuni.cz` a po kliknutí na ikonu Distropine prejdeme adresárovou štruktúrou až na poddoménu `kam.mff.cuni.cz`. Vidíme, že pri každom rozbalení adresára sa čaká na prijatie referenčných záznamov podadresárov z DHT.

5. Štruktúra aplikácie a detaily implementácie

Táto kapitola popisuje štruktúru modulov, z ktorých pozostáva aplikácia Distropine. Ďalej sú tu rozobrané tie implementačné rozhodnutia, ktorých podoba nie je diktovaná doteraz navrhnutým dizajnom aplikácie. Naopak, v tejto kapitole neuvádzame popis tých častí programu, u ktorých je implementácia do veľkej miery vynútená návrhom dizajnu, tak ako bol popísaný v kapitole 3 alebo dizajnom DHT vrstvy popísanej v časti 2.2.

5.1 Dekompozícia problému

Pri návrhu aplikácie Distropine sa ukázalo ako užitočné dekomponovať jej funkcionalitu do niekoľkých vrstiev, kde každá vrstva je relatívne nezávislým modulom, resp. úrovňou abstrakcie nad ktorou možno uvažovať samostatne. Táto dekompozícia je aplikovaná pri rozdelení programu do menných priestorov a tried a následne pri rozdelenie kódu do súborov a adresárov.
Pri vrstevnatom dizajne je zachovaný princíp, že tok dát, udalostí a volaní prebieha výlučne medzi susednými vrstvami aplikácie, pričom vždy nižšia vrstva poskytuje svoje služby vyššej vrstve. Vďaka tomu je každej vrstve skrytá zložitosť aplikácie na nižších aj vyšších úrovniach.
Aplikácia pozostáva zo štyroch vrstiev:

- L0. Vrstva L0 zabezpečuje sieťovú komunikáciu nad protokolom UDP a tento protokol prekrýva jednoduchou relačnou vrstvou (trieda `Session`), ktorú publikuje vrstve L1. Z pohľadu sieťového overlay (ktoré leží na L1) je L0 vrstvu podkladovej sieťovej infraštuktúry.
- L1. Vrstva L1 je vrstvou DHT, obsahuje implementáciu algoritmu Kademlia — jej dizajn je popísaný v časti 2.2.
- L2. Návrh vrstvy L2 je základným výsledkom tejto práce a je obsahom kapitoly 3. L2 je vrstvou aplikačnej logiky služby pre komentovanie webových zdrojov. Na tejto úrovni vystupujú koncepty ako *komentár* alebo *užívateľ* a nad nimi operácie vyhľadania komentárov, autentizácia, registrácia užívateľov atď.
- L3. Najvyššie v zásobníku vrstiev sa nachádza vrstva pre styk s užívateľským rozhraním, vrstva L3. Táto vrstva sprostredkuváva komunikáciu s GUI, ktoré je popísané v sekcii 4.1.

5.2 Implementácia vrstevnatého dizajnu

Každá vrstva má v programe Distropine vlastný menný priestor. Ten obsahuje tri hlavné kategórie tried: triedu `Layer`, triedy kontextov a triedy paketov.

5.2.1 Trieda `Layer`

Trieda `Layer` existuje pre každú vrstvu v jedinej inštancii. Nejedná sa však o singleton v pravom zmysle slova, pretože v programe je možné súčasne udržiavať niekoľko paralelných zásobníkov vrstiev, k čomu dochádza vtedy, keď je v jednej bežiacej inštancii programu simulovaná P2P sieť viacerých uzlov.
Objekt triedy `Layer` na vrstve i je vlastníkom objektu triedy `Layer` na vrstve $i-1$ (pre $i > 0$). Tento objekt nižšej vrstvy sa vytvára v konštruktore, takže konštrukciou triedy `Layer` na vrstve L3 dochádza ku kaskádovému vzniku celého zásobníka vrstiev.
Konštrukciou sa ešte neiniciujú žiadne sieťové prenosy. Za týmto účelom má každá trieda `Layer` metódu `Bootstrap`. Kde opäť dochádza ku kaskádovým volaniam, najprv sa vždy realizuje bootstrap nižšej vrstvy, až následne bootstrap aktuálnej vrstvy.

5.2.2 Akcie

Zaveďme pojdem *akcia vrstvy* a označme ním algoritmus vzťahujúci sa k danej vrstve, ktorý v konečnom dôsledu znamená komunikáciu s iným uzlom alebo uzlami systému. Jedná sa o procedúru, ktorá je v kontexte dizajnu danej vrstvy primitívou, využívanou vyššou vrstvou na implementáciu vlastných algoritmov. Pre vrstvu L1 sa jedná napr. o príkazy `FIND_*`, `STORE`, `lookup_*`. Pre vrstvu L2 sa jedná o procedúry pre zverejnenie komentáru, získanie referenčných záznamov pre danú domému apod.
Volania akcií vrstvy možno riešiť dvoma prístupmi:

Akcia ako blokačná operácia

Pri tomto prístupe je akcia z pohľadu volajúceho kódu funkciou, ktorej sú dodané vstupné argumenty, je vykonaná a vracia výsledok. Pri volaní akcie dochádza na úrovni vrstvy L0 ku komunikácii s iným uzlom, a teda je treba čakať na príchod datagramu s odpoveďou. Na úrovni vrstvy L0 preto musí byť pri tejto metóde implementovaný pooling, ktorý zabezpečí zablokovanie behu programu, kým nie prijatá odpoveď (alebo nevyprší časový limit). Nevýhodou je, že ak chceme spustiť viacero akcií súčasne, je nutné pre každú z nich vytvoriť vlastné vlákno a zablokovať beh programu až do ukončenia všetkých vlákien. Túto potrebu by sme mali pri iteratívnych algoritmoch vrstvy L1 (napr. `lookup_*`), kedy je potrebné súčasne zaslať RPC správu viacerým uzlom. Jedná sa o neefektívne riešenie.

Asynchrónny prístup

Pri tomto prístupe je akcia objektom, ktorý budeme označovať *kontext*. Kontext má metódu `Start`, udalosť `OnFinished` a dátový člen `Result`. Kontext je skonštruovaný s argumentami volania príslušnej akcie. Volajúci kód vkladá do udalosti `OnFinished` svoju obslužnú rutinu, ktorá bude zavolaná po skončení akcie. Akcia je odštartovaná metódou `Start`. Interne dochádza k volaniu akcií podkladovej vrstvy (ak sme na vrstve L0, dochádza k odosielaniu paketu). Odštartovenie nie je blokačná operácie a po návrate z nej nie je akcia ukončená. Kontext oznamuje ukončenie akcie vyvolaním udalosti `OnFinished`. Obslužná rutina

prevezme výsledok akcie z dátového člena `Response` a pokračuje programom, ktorý by pri blokačnom prístupe nasledoval bezprostredne za zavolaním akcie.
Vidíme, že je možné vyvolať viac akcií súčasne, no ak ich obslužné rutiny manipulujú so zdieľanými dátami, je potrebné, aby implementovali zamykanie.
Ako už bolo naznačené, tento prístup je použitý v aplikácii Distropine. Triedy kontextu dostávajú vo svojich konštruktoroch okrem argumentov akcie aj referenciu na objekt `Layer` príslušnej vrstvy. Kontexty tak môžu pristupovať k dátam spoločným pre celú vrstvu, ako napr. identifikácia uzla, identita prihláseného užívateľa apod.
Zavedením kontextov sme sa v celej aplikácii vyhli použitiu poolingu. Výnimkou je komnikácia GUI klienta a HTTP servera vrstvy L3. Tu je to nevyhnutné, pretože neexistuje spôsob ako by HTTP server informoval prehliadač, resp. JavaScriptový kód o ukončení nejakej akcie.

5.2.3 Pakety

Ďalšou kategóriou tried, ktoré sa vyskytujú na každej vrstve sú *pakety*. Jedná sa o dátové obálky, ktoré sú prenášané na rozhraní vrstiev. Je tu aplikovaný princíp zapuzdrovania, kedy pri odosielaní správy, každá vrstva zabalí paket vyššej vrstvy do svojho paketu a zároveň doň pridá dáta, ktoré zasiela na tú istú vrstvu protistrany.
Pri deserializácii ľubovoľného objektu nie je zaručené, že výsledok je v konzistentom, resp. valídnom stave. Každý paket má preto metódu `CheckValidity`, ktorou skontroluje vlastnú validitu a rekurzívne validitu všektých paketov, ktoré zapuzdruje. Táto kontrola sa robí bezprostredne po prijatí a deserializácii paketu na vrstve L0, tzn. že vo zvyšku programu možno predpokladať, že každý paket je vo valídnom stave.

5.3 Triedy ležiace mimo zásobník vrstiev

V programe existuje niekoľko tried ležiacich mimo zásobník vrstiev. Ich funkcionalita, príp. dáta v nich uložené, sú zdieľané naprieč vrstvami. Nesledujúce podsekcie menujú najdôležitejšie z týchto tried.

5.3.1 Trieda `CommonCrypto`

Statická trieda `CommonCrypto` je modul zapuzdrujúci funkcionalitu šifrovania, podpisovania, hašovania a tiež serializácie objektov. Vďaka tomu nemusia byť vo zvyšku programu referencované konkrétne kryptografické algoritmy. Trieda zverejňuje len generické operácie ako `Encrypt`, `Sign`, `VerifySignature`, `Hash`, `GetRandomData`, `Serialize` apod.

Kryptografické operácie

V aplikácii sú použité implementácie kryptografických algoritmov, ktoré sú súčasťou knižnice .NET Framework. Tabuľka 5.1 obsahuje základný prehľad použitých algoritmov a ich parametrov.

Operácia	Algoritmus	Príklad použitia
Hašovanie	SHA-1	Výpočet kľúča dátovej položky a nodeID uzla.
Symetrické šifrovanie	AES, kľúč: 128 bitov, mód: CBC, výplň: PKCS7	Zabezpečenie tajného podpisového kľúča v IDC.
Odvodenie šifrovacieho kľúča z hesla	PBKDF2, s generátorom pseudonáhodných čísel založeným na HMACSHA1.	Vytvorenie tajného kľúča, ktorým sa šifruje IDC zo známeho hesla a soli.
Podpisová schéma	RSA, 1024-bitov, hašovanie: SHA-1	Podpisovanie na L0 (podpisovnie paketov), na L2 (podpisovanie komentárov, certifikátov).

Tabuľka 5.1: Prehľad konkrétnych implementácií kryptografických operácií realizovaných triedou `CommonCrypto`

Serializácia

Serializácia (a deserializácia) objektov je nutná v dvoch situáciách:

- pri odosielaní (prijímaní) paketu vrstvy L0. Tým, že paket obsahuje vnorené pakety vyšších vrstiev, dochádza k (de)serializácii celej správy.
- pri podpisovaní objektov (resp. overovaní podpisov). Napr. vtedy, keď autor komentáru podpisuje jeho vnútornú časť, aby ho spolu s podpisom zabalil do výsledného komentáru. Vtedy je potrebné objekt najprv serializovať a na výsledné dáta aplikovať podpisovú operáciu.

Na serializáciu sú použité tzv. dátové kontrakty. Pri tomto prístupe nemusí byť serializovaný objekt deserializovaný voči svojmu pôvodnénu typu. Stačí aby sa jednalo o typ s rovnakým dátovým kontraktom, pretože dátový kontrakt je len abstraktný popis dátových členov a ich typov. To umožňuje oi. ľahšie dosiahnuteľnú kompatibilitu rôznych verzií toho istého softvéru.
Výsledkom serializácie sú XML dáta kódované v binárnom tvare, ktorý je použitý za účelom redukcie veľkosti výsledných správ a zrýchlenia ich spracovania. Tento formát je otvorený, takže nekladie prekážku prenositeľnosti aplikácie.

5.3.2 Trieda `Consts`

Trieda obsahujúca globálne parametre systému. Niektoré z nich sú konštanté, iné sú hodnoty nastavené konfiguráciou, príp. nastaviteľné prostredníctvom konzolového rozhrania.

5.3.3 Trieda `MyLogger`

Logovací systém je pri vývoji P2P aplikácie významný ladiaci nástroj. Pri simulovaní systémov s viacerými súčasne bežiacimi uzlami umožňuje sledovať ich vzájomnú interakciu, zmeny ich vnútorného stavu, presné zdroje výnimiek atď.

Túto logiku implementuje trieda `MyLogger`. Jej inštanciu možno vytvoriť z ktorejkoľvek triedy programu a následne voči nej volať metódy ako `Trace` a `Log`. Logovacie hlášky sú v triede `MyLogger` filtrované voči kritériám nastaviteľným v konzolovom rozhraní a následne zapisované do spoločného logovacieho súboru. Medzi kritériá patrí napr. voľba horného a dolného limitu na vrstvu aplikácie, z ktorej sa bude logovať alebo povolenie logovania len určitému uzlu.
Ako podkladový logovací systém je zvolená knižnica NLog[1]. Jej prednosťami sú vysoká výkonnosť a konfigurovateľnosť. Projekt NLog je licencovaný tzv. „Zjednodušenou BSD licenciou", ktorá je kompatibilná s licenciou GNU GPL využívanou aplikáciou Distropine.

5.4 LHT

Každý DHT uzol obsahuje lokálnu databázu, v ktorej udržiava podmnožinu dát, ktorá mu prináleží na základe jeho pozície v priestore kľúčov. Túto funkcionalitu u každého uzlu zapuzdruje trieda LHT[2]. Objekt tejto triedy je vlastnený objektom triedy `Layer` na vrstve L2.

5.4.1 Perzistentné uloženie dát

V aplikácii Distropine je na jednej strane implementácia lokálnej databázy uzlu sekundárnym problémom, ktorý sa snažíme riešiť jednoducho a nie nutne optimálne. Na druhej strane, uzol môže mať v rámci DHT vo svojej zodpovednosti aj relatívne veľké množstvo dát. V takom prípade nie je vhodné, aby boli všetky dátové položky počas behu aplikácie udržiavané v operačnej pamäti. LHT teda musí obsahovať perzistentné úložisko dát, voči ktorému je možné klásť dotazy. Na základe týchto faktorov by bola vlastná implementácia takého systému zbytočne pracná preto sa pokúsime použiť nejaké už existujúce riešenie.
V kontexte aplikácie Distropine od vrstvy pre perzistenté uloženie dát očakáva predovšetkým maximálne zapúzdrenie jej vnútornej logiky tak, aby sa jej nemusel prispôsobovať dizajn aplikácie. Pri tejto podmienke je nasadenie relačnej databázy nevhodné, nakoľko vyžaduje udržiavanie prekladovej medzivrstvy medzi relačným a objektovým modelom. Naopak, ako vhodná sa ukazuje paradigma objektových databáz a to vďaka tomu, že databáza a vonkajší kód používajú rovnaký dátový model.
Ďalšími podmienkami v kontexte aplikácie Distropine sú: kompatibilita s licenciou GNU GPL, možnosť fungovania ako knižnice referencovanej v kóde (tj. nie samostatný server) a použiteľnosťou s platformou .NET. Týmto podmienkam vyhovuje niekoľko projektov, nakoniec bola zvolená knižnica db4o[3].
Db4o má vysokú mieru integrácie do jazyka C#, predovšetkým vďaka podpore technológie LINQ[4]. Pre ilustráciu tohto tvrdenia uveďme ukážku dotazovania voči databáze:

```
var resultset = db
```

[1] `http://nlog-project.org/`

[2] Skratka od *Local Hash Table*, slúži len pre odlíšenie od DHT. Ako sa ukáže, táto trieda interne neukladá dátové položky v hašovacej tabuľke.

[3] `http://www.db4o.com/`

[4] Language Integrated Query, deklaritívny dotazovací jazyk včlenený do jazyka C#.

```
.Query<LhtComment>()
.Where(cmt => cmt.PublishDate < dateMaxTreshold)
.OrderByDescending(cmt => cmt.PublishDate)
.Take(maxCount)
.Select(cmt => cmt.InnerContent)
.ToArray();
```

Vidíme, že dotaz je silne typizovaný (funkcia je generická) a filtračné kritéria sú špecifikované pomocou lambda-výrazov namiesto štandardného spôsobu, kedy je dotaz zadaný v reťazci, ktorý je argumentom funkcie. To uľahčuje prípadný refaktoring.
Nevýhodou je tu nižšia výkonnosť a to aj napriek tomu, že db4o podporuje vytváranie indexov a ďalšie optimalizácie (ktoré sú v aplikácii Distropine využité).

5.5 Rozhranie program-GUI

5.5.1 Podrobne o *rozšírení* prehliadača Chrome

Na vrstve L3 sa nachádza HTTP server pre komunikáciu s GUI klientom aplikácie Distropine, ktorý je *rozšírením*[5] prehliadača Chrome. Spôsob tejto komunikácie je popísaný v časti 4.1.
Zdôraznime len, že komunikácia medzi prehliadačom a HTTP serverom v tomto prípade nefunguje štandardným spôsobom, kedy prehliadač požiada o HTML stránku, obrázky, štýly atď. Všetky tieto súbory sú zabalené do komprimovaného a podpísaného súboru `Distropine.crx`. Aplikácia Distropine pri svojom spustení vyhľadáva lokálnu inštaláciu preliadača Google Chrome, kopíruje tento súbor do príslušného adresára a registruje *rozšírenie* v konfiguračných súboroch. Pri nasledovnom spustení programu Chrome sa objaví ikona *rozšírenia* vedľa adresového riadku. Súbory HTML stránky sú teda behom celej existencie *rozšírenia* fixné.

5.5.2 HTTP server

HTTP server na vrstve L3 je využívaný výlučne na odpovedanie AJAX dotazov. To znamená, že medzi *rozšírením* (resp. jeho JavaScript kódom) a serverom sú prenášené požiadavky a odpovede vo formáte JSON[6] zabalené do správ protokolu HTTP. Úlohou skriptu *rozšírenia* je teda posielať serveru dotazy na základe užívateľových akcií a na základe odpovedí od servru upravovať obsah HTML stranky (modifikovaním jej DOM[7] modelu).

5.5.3 Medzipamäť adresárových záznamov a komentárov

Na vrstve L3 je medzi HTTP servrom a vrstvou L2 uložená tzv. medzipamäť adresárových záznamov a komentárov (ďalej medzipamäť). Na ľavej strane tejto

[5]Slovo 'rozšírenie' v zmysle *browser extension* uvádzame pre zvýraznenie kurzívou, kvôli jeho viacznačnému významu.
[6]JavaScript Object Notation, Formát pre serializáciu objektov používaný s technológiou AJAX.
[7] Document Object Model, Objektová reprezentácia HTML kódu, ktorou prehliadač sprístupňuje JavaScriptu zobrazenú stránku.

medzipamäte sa nachádza HTTP server a požiadavky zo strany *rozšírenia* a na pravej strane je vrstva L2 a DHT.
Vždy, keď je v prehliadači otvorené okno *rozšírenia*, tak sa voči HTTP servru v pravidelných intervaloch (napr. 2 s) vysielajú požiadavky `GetComments(url)` a `GetDirReferences(url.domain)`, tj. požiadavky o najaktuálnejšie komentáre a adresárové referencie pre danú URL. Tieto požiadavky sú spracovávané ľavou stranou medzipamäte, ktorá vždy poskytne všekty komentáre a adresárové referencie, ktoré pre dané URL obsahuje.
Medzipamäť si pri každej URL, pre ktorú obsahuje nejaké dátové položky uchováva dátum, k akému je tento obsah aktuálny, tj. dátum, kedy bol uskutočnený posledný lookup voči vrstve L2 (a sprostredkovane voči DHT) pre toto URL. Ak medzipamäť zaregistruje na ľavej strane dotaz o také URL, ktoré nebolo dlhú dobu (alebo ešte nikdy) obnovené, potom pravá strana požiada vrstu L2 o aktualizáciu (použije príslušný kontext pre získanie komentárov alebo adresárových referencií).
Ľavá strana je od tohto procesu odtienená, až na indikáciu, že príslušná bunka medzipamäte je v procese obnovenia a môžu do nej pribudnúť dátové položky. V GUI sa táto informáciu využije na zobrazenie animácie načítavania obsahu. V prípade, že pribudnú v medzipamäti dátové položky, kód *rozšírenia* ich zobrazí v krátkej dobe vďaka poolingu, ktorý ustavične realizuje.

5.5.4 Uloženie stavu GUI

Medzi jednotlivými otvoreniami okna *rozšírenia* neexistuje žiaden kontext. Pri každom zavretí okna je teda stratený všetok stav, ktorý užívateľ v GUI vytvoril, napr. nastavenie ovládacích prvkov, expandovanie adresárovej štruktúry apod. Z tohoto dôvodu *rozšírenie* pri každej zmene serializuje svoj stav a odosiela ho HTTP serveru prikladajúc kľúč, ktorým je aktuálne URL. Server si ukladá dvojice (URL : stav-GUI). Keď je neskôr okno *rozšírenia* otvorené, je zistené aktuálne URL, je načítaný príslušný stav a GUI je nastavené do tvaru v akom ho užívateľ zanechal.

Záver

Záverečné hodnotenie projektu Disropine — jeho prínosu a úspešnosti naplnenia požiadaviek stanovených v Úvode — rozdeľme do dvoch oblastí: užívateľskej a akademickej.

Užívateľská oblasť

V užívateľskej oblasti sme dosiahli vytvorenie plnohodnotnej služby pre komentovanie internetových zdrojov. Dôležitým v tejto oblasti je jednoduché a intuitívne užívateľské rozhranie integrované do webového prehliadača. Jeho prínos spočíva v tom, že skrýva komplexnosť aplikácie, ktorá sa nachádza za ním a je postavené na šablóne webového diskusného fóra typu klient-server. Nazdávam sa, že toto obalenie P2P paradigmy do klient-server formy je nutnou požiadavkou užívateľského prijatia tohto typu aplikácie.
Problémom pri návrhu v tejto oblasti bola nekonzistentnosť v adresovaní obsahu webových stránok. Tvorcovia tu využívajú tri hlavné smery: adresovanie identifikátorom URL, časťou *path* alebo *query* a transparentné adresovanie pomocou technológie AJAX. Následkom toho môže byť použiteľnosť aplikácie u niektorých stránok znížená.

Akademická oblasť

Nazerajme na zadanie stanovené v úvode tejto práce ako na neformálny príklad optimalizačnej úlohy. Sada obmedzujúcich podmienok je nasledovná:

- riešenie je implementáciou diskusného fóra.
- obsah je ukladaný distribuovane metódou DHT.
- je zavedený pojem identity užívateľa, je zamedzená krádež identity.
- dátové položky majú zabezpečenú integritu.

Maximalizovanou cieľovou funkciou tejto optimalizačnej úlohy je miera decentralizácie riešenia.
V riešení, ktoré predkladám, nebol dosiahnutý ideálny výsledok: plne decentralizovaný P2P systém. Bol zavedený koncept certifikačnej autority — prvku systému, ktorého verejný kľúč je známy a dôverovaný medzi všetkými uzlami siete. Mechanizmus prenosu dôvery pomocou certifikátov umožnil zabezpečiť apriórnu dôveru medzi užívateľmi siete, resp. dôveru v autenticitu obsahu, ktorý ľubovoľný užívateľ vytvoril a podpísal.
Nazdávam sa, že v tomto type systému, kde užívatelia neustanovujú akékoľvek väzby je zavedenie spoločne dôverovaného prvku nutnosťou. V tomto sa teda služba diskusného fóra odlišuje od iných UGC služieb, u ktorých je v súčasnosti snaha vytvárať ich P2P alternatívy; napr. sociálnych sietí alebo chatovacích služieb.
Ďalším rozdielom, predovšetkým voči problému P2P sociálnej siete, je to, že v prípade diskusného fóra nie je potrebné zavádzať autorizáciu užívateľov voči obsahu. Obsah diskusného fóra je *broadcastového* typu, tj. je prístupný len na čítanie a to všetkým participantom systému (aj neutentizovaným).

Eliminácia nevýhod centrálneho prvku. Existencia centrálneho prvku so sebou nesie typické riziká a nevýhody ako napr. zavedenie jediného bodu zlyhania, umožnenie monitorovať a cenzurovať činnosť užívateľov, problematické škálovanie systému.
Tieto faktory som sa snažil eliminovať tým, že funkcia CA spočíva výlučne vo vystavovaní a obnovení certifikátov, preto výpadok CA neohrozí elementárny chod systému. CA nemá prehľad o aktivitách užívateľov, nemá moc cenzurovať obsah a ani samovoľne revokovať certifikáty. Navyše je možný súčasný beh viacerých inštancií CA v rámci P2P siete, čím je obmedzený faktor jediného bodu zlyhania a problematického škálovania.
Pri návrhu sa vyskytli viaceré problémy, u ktorých by bol centralizovaný prístup jednoduchší, no podarilo sa ich riešiť decentralizovane. Ide napr. o autentizáciu užívateľov alebo udržiavania metadát súvisiacich s adresárou štruktúrou diskusných vlákien.

Ďalší vývoj

Smery, ktorými sa môže vývoj aplikácie Distropine ďalej uberať je možné rozdeliť do niekoľkých vetiev:

- *sociálny aspekt*: zavedenie užívateľského profilu, vzájomné hodnotenie užívateľov, hodnotenia diskusných vlákien atď.
- *dostupnosť*: lepšie riešený bootstrapping DHT, podpora IPv6, rozšírenie množiny webových prehliadačov, s ktorými je GUI klient kompatibilný.
- *decentralizovanosť a zabezpečenie*: reputácia užívateľov, dynamicky vytvárané delegované certifikačné autority (certifikáty podpisované užívateľmi s vysokou reputáciou), ochrana voči spamu.

Zoznam použitej literatúry

[1] Leucio Antonio CUTILLO, Refik MOLVA, Melek O¨NEN. *Safebook: A Distributed Privacy Preserving Online Social Network*. EURECOM, Sophia Antipolis, France.

[2] http://diasporaproject.org/.

[3] The Growth of Diaspora – A Decentralized Online Social Network in the Wild. Ames BIELENBERG, Lara HELM. Global Internet Symposium 2012.

[4] http://friendica.com/

[5] http://www.thimbl.net/

[6] http://status.net/

[7] http://www.gnu.org/copyleft/gpl.html

[8] Sameh EL-ANSARY, Seif HARIDI. *An Overview of Structured P2P Overlay Networks*. Swedish Institute of Computer Science (SICS). Royal Institute of Technology - IMIT/KTH, Sweden. Citeseer (citeseerx.ist.psu.edu), 2004.

[9] Petar MAYMOUNKOV, David MAZIERES. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. New York University. Peer-to-peer systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA. Springer, 2002. ISBN: 3540441794.

[10] Ion STOICA, Robert MORRIS, David KARGER, M. Frans KAASHOEK, Hari BALAKRISHNAN. *Chord: A scalable peer-to-peer lookup service for internet applications*. Proceedings of the ACM SIGCOMM '01 Conference, San Diego, California, August 2001.

[11] Ingmar BAUMGART, Sebastian MIES. *S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing*. Institute of Telematics, Universität at Karlsruhe (TH). International Conference on Parallel and Distributed Systems, 2007. ISBN: 978-1-4244-1889-3.

[12] J. DINGER, O. WALDHORST. *Decentralized Bootstrapping of P2P Systems: A Practical View*. Proc. 8th IFIP TC6 Int. Conf. on Networking. May 2009, pp. 703–715.

[13] Roland BLESS, Oliver P. WALDHORST, Christoph MAYER, Hans WIPPEL. *Decentralized and Autonomous Bootstrapping for IPv6-based Peer-to-Peer Networks*. Institute of Telematics, Universit¨at Karlsruhe (TH), Zirkel 2, D–76128 Karlsruhe, Germany.

[14] Jordan RITTER. *Why Gnutella Can't Scale. No, Really.* 2001.

[15] Carlise ADAMS, Steve LLOYD. Understanding PKI. Concepts, Standards and Deployment Considerations.

[16] Changxi ZHENG, Guobin SHEN, Shipeng LI, Scott SHENKER *Distributed Segment Tree: Support of Range Query and Cover Query over DHT.* Microsoft Research Asia, Beijing, 100080, P.R.China.

[17] Sriram RAMABHADRAN et al. *Prefix Hash Tree. An Indexing Data Structure over Distributed Hash Tables.* University of California, San Diego

[18] Jesse James GARRETT. *Ajax: A New Approach to Web Applications.*