České Vysoké Učení Technické v Praze
Fakulta Elektrotechnická
Katedra Kybernetiky

Bakalářská práce

Peer-to-Peer zálohovací systém založený na BitTorrent protokolu

Maxat Mansurov

Vedoucí práce: *Ing. Miroslav Uller*

Studijní program: Otevřená Informatika (bakalářský)
Obor: Informatika a počítačové vědy

25. května 2017

Poděkování

Rád bych poděkoval mému vedoucímu práce Ing. Miroslavu Ullerovi za nadnesení tématu, které mě zaujalo, a za jeho konzultace a rady k vypracování práce. Dík samozřejmě patří i původnímu autoru zadání RNDr. Petru Pudlákovi, Ph.D., který měl velmi cenné komentáře. Svým rodičům a prarodičům za to, že mi umožnili studium a vždy mě podporovali. V poslední řadě nesmím zapomenout poděkovat všem vyučujícím, se kterými jsem se během svého studia setkal, za jejich vstřícnost a za veliké množství vědomostí, které mi umožnili poznat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. května 2017 .....

Abstrakt

Cílem této práce je navrhnout a implementovat prototyp distribuovaného Peer-to-Peer systému pro zálohování dat s použitím protokolu BitTorrent. Návrh systému by měl umožňovat uživatelům přispět svým lokálním volným místem na disku do systému výměnou za spolehlivé úložiště jejich dat u jiných uživatelů. Výsledkem práce je funkční prototyp takového systému, který implementuje základní myšlenky a ukazuje koncepci dalšího rozvoje.

Klíčová slova: Online zálohování; Síť; Distribuovaný; Peer-to-peer; DHT; BitTorrent.

Abstract

The aim of this work is to design and implement a prototype of a distributed Peer-to-Peer backup system with the usage of the BitTorrent protocol. The system design should let the users to contribute their local free space in exchange of reliable remote storage, which is a virtually co-allocated space on other users' devices. The result is a functional prototype of such system, which implements the basic idea and shows a concept for further development.

Key words: Online backup; Network; Distributed; Peer-to-peer; DHT; BitTorrent.
Title translation: Peer-to-peer backup system based on BitTorrent.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Motivace

V oblasti informačních technologií (stejně jako v jiných oblastech) nikdy nelze zajistit 100% spolehlivost. Hardware a software lze nahradit docela jednoduše, avšak data, které uživatel vytváří často i několik let, jsou často nenahraditelná, proto je nutné řešit také jejich zálohování.

Možná rizika ztráty dat mohou být následující:

- Chyba nebo poškození hardwaru.
- Chyba softwaru.
- Počítačové viry a škodlivý software.
- Chyba uživatele.
- Přírodní vlivy.
- ...

Proces zálohování dat je velice důležitý pro obyčejné uživatele osobních počítačů i pro velké firmy. Klasické způsoby zálohování dat ovšem mohou být finančně velmi náročné, jelikož vyžadují regulární rozšiřování kapacity nosičů zálohovaných dat buďto „fyzicky“ (zakoupení nových magnetických pásek, pevných/flash/optických disků apod.), nebo „virtuálně“ (příplatek v on-line zálohovacích službách jako *Dropbox*, *iCloud*, *Google Drive* apod.). Zatímco velké firmy a organizace si mohou dovolit používat takové způsoby zálohování dat, malé firmy a obyčejní uživatelé osobních počítačů s tím mívají potíže. Jiným důležitým pozorováním je, že kapacita moderních pevných disků mnohonásobně převyšuje potřeby většiny uživatelů osobních počítačů. Jedním z možných řešení problému se zálohováním se proto jeví vybudování sítě počítačů, ve které si jednotliví uživatelé budou navzájem poskytovat nevyužité

místo na svých pevných discích pro účely ukládání záloh. Vzhledem k tomu, že rychlost předávání dat prostřednictvím Internetu neustále roste, taková síť se může stát docela levnou a efektivní alternativou klasickým způsobům zálohování dat.

1.2 Cíl práce

Hlavní cíl práce je navrhnout architekturu systému pro Peer-to-Peer (P2P) distribuované zálohování s využitím protokolu BitTorrent. Na základě tohoto návrhu vytvořit funkční prototyp systému. Zhodnotit kvalitu návrhu a implementace provedením zálohování v simulovaném P2P prostředí s použitím reálných dat.

1.3 Scénáře použití

Typický scénář zálohování je následující: každý uživatel zazálohuje svá data, zašifruje je a případně je digitálně podepíše a publikuje je do P2P sítě. Současně nabídne určitý diskový prostor ostatním účastníkům pro uložení jejich dat. Následně si uživatelé automaticky přepošlou data aby jejich data byla rovnoměrně rozložena v síti a mohla být stažena jejich vlastníkem v případě ztráty dat. Pro uživatele by bylo také velice pohodlné, kdyby mohl svá data verzovat, neboli mít přístup ke všem změnám u každého souboru.

1.4 Struktura textu

V tomto odstavci si představíme strukturu této práce. V první části se podíváme na několik již existujících P2P systémů zálohování dat. Popíšeme si je z hlediska sdílení, synchronizace, bezpečnosti a férovosti. Ve druhé části se zaměříme na návrh našeho systému BTVault, kde detailně rozebereme všechny jeho součásti. Následující kapitola se bude věnovat implementaci prototypu systému. Zaměříme se zde na popis jednotlivých modulů a komunikací v rámci celkového systému. V závěru si zhodnotíme cíle práce a co se nám podařilo splnit a navrhneme, jak se dá práce dále rozšiřovat. V příloze je možné nalézt uživatelskou příručku k prototypu (instalace, nastavení a spuštění).

Kapitola 2

Přehled existujících peer-to-peer zálohovacích systémů

V této kapitole si předvedeme několik již existujících P2P systémů zálohování dat, popíšeme jejich hlavní vlastnosti a provedeme jejich podrobnou analýzu. Při analýze se primárně zaměříme na způsoby implementace těchto částí programů:

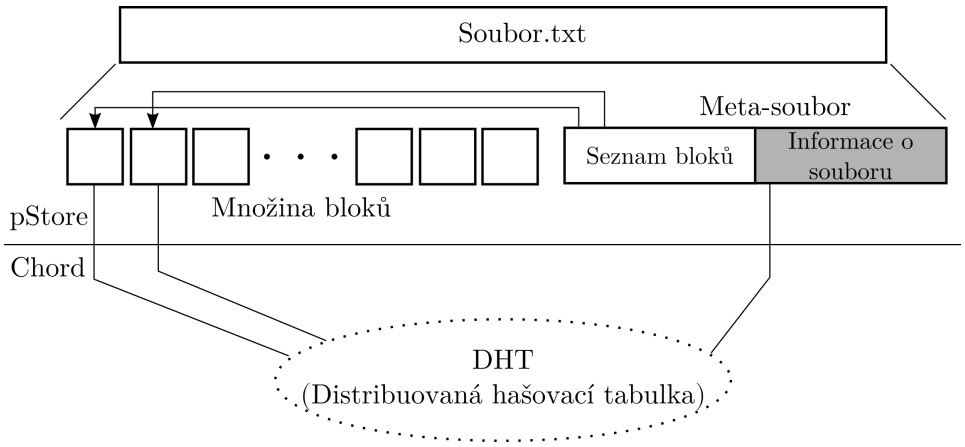
- Zálohování dat.
- Replikace dat.
- Distribuce dat mezi uzly.
- Férovost a bezpečnost systému.

2.1 pStore

Úvod

Projekt pStore [1] byl vyvinut v roce 2001 na MIT [1]. Tento projekt vznikl jako snaha zkombinovat výsledky provedených výzkumů ve sféře distribuovaných P2P sítí a technik pro inkrementální zálohování dat. Jelikož byl ale pStore původně myšlen pouze jako výzkumný projekt a byl vytvořen jedině za účelem ověření koncepce, v dnešní době se reálně nikde nepoužívá, a dokonce ani autoři projektu nemají v plánech nějak dál ho rozvíjet či podporovat.

[1] projekt byl vyvinut v rámci kurzu MIT 6.824: Distributed Computer Systems



Obrázek 2.1: Zálohovací proces v pStore

Zálohování dat

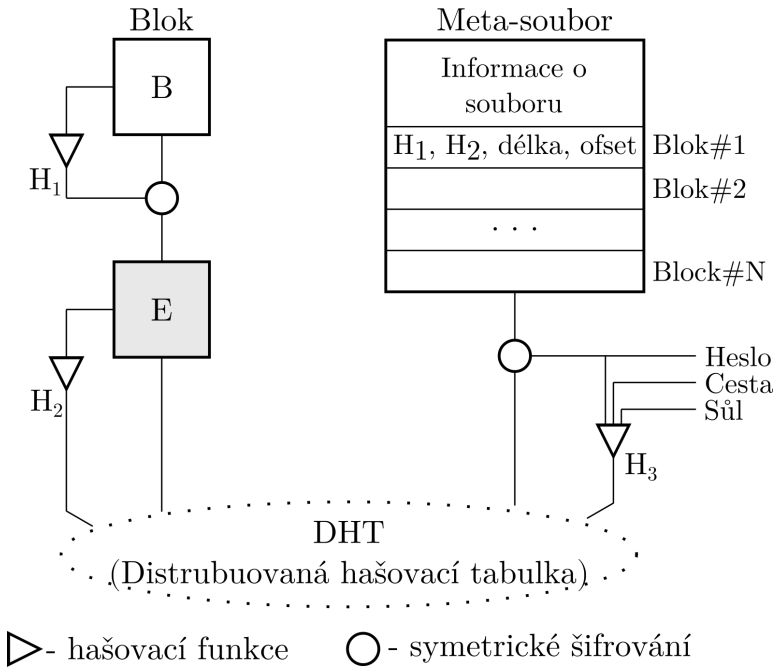
Systém pStore se primárně zabývá implementaci procesu zálohování dat, a veškerou P2P funkcionalitu přenechává systému Chord [2]. Při zálohování určitého souboru pStore postupuje následovně:

1. Rozdělí soubor na množinu bloků fixní velikosti (výjimkou je poslední blok, který může mít menší velikost).

2. Vytvoří metadata, která budou obsahovat veškerou informaci potřebnou pro rekonstrukci daného souboru z odpovídající množiny jeho bloků.

3. Uloží bloky a metadata do distribuované hašovací tabulky (distribuci do jednotlivých uzlů nechává na starosti systému Chord).

Celý proces zálohování souboru je zobrazen na Obrázku 2.1. Před vložením do distribuované hašovací tabulky (DHT) se všechny bloky a metadata šifrují. K zašifrování bloků se používá algoritmus konvergentního šifrování [3], a samotné šifrování každého bloku probíhá v několika krocích. Nejdříve se z obsahu bloku vygeneruje pomocí hašovací funkce (např. SHA-2) hash H_1, která se použije jako symetrický šifrovací klíč k jeho zašifrování. Pak se z obsahu již zašifrovaného bloku E vygeneruje další hash H_2, která bude sloužit jako identifikační číslo tohoto bloku v systému. Potom se dvojice $\langle H_2, E \rangle$ vkládá do DHT. Pak je informace o bloku spolu s H_1 a H_2 je zapsána do metadat. Metadata jsou zašifrována pomocí symetrického šifrovacího klíče odvozeného z uživatelských přihlašovacích údajů. Jako identifikační číslo pro metadata slouží hash H_3, vygenerovaná z řetězce vzniklého zřetězením uživatelského hesla, absolutní cesty do souboru a předem dané soli. Proces vytvoření a šifrování bloků je zobrazen na Obrázku 2.2.



Obrázek 2.2: Proces vytvoření a šifrování bloků v pStore

Distribuce dat

Použití konvergentního šifrování v pStore zaručuje, že všichni uživatelé systému při zálohování shodného souboru dostanou stejnou množinu zašifrovaných bloků, tedy že všechny požadavky o uložení totožných bloků budou směřované stejným uzlům v DHT. A proto, pokud nějaký uzel obdrží žádost o uložení bloku identického tomu který již uložený má, bude tu žádost jednoduše ignorovat a odpoví zprávou znamenající úspěšné uložení bloku. Takové chování systému garantuje, že všechny bloky budou uloženy v systému pouze jednou a že vždy dojde ke sdílení totožných bloků mezi uživateli.

Replikace

Replikace dat je jednoduchý, ale velmi efektivní způsob jak zvýšit celkovou spolehlivost systémů distribuce dat. V pStore replikace dat probíhá tak, že pStore ukládá do DHT několik kopií každého bloku. pStore vytváří identifikační čísla bloků vygenerováním hashe z H_2 spojeného se speciální kryptografickou „solí“. pStore používá několik předem definovaných a dobře známých „solí“. Například, pro vytvoření identifikačního čísla i-té kopie nějakého bloku, pStore zřetězí „sůl“ pro i-tou kopii s identifikačním číslem tohoto bloku H_2 a vygeneruje z výsledného řetězce pomocí jednostranné hašovací funkce hash. Takto získaná hash se pak použije jako klíč při vkládání kopie do DHT. Tímto způsobem se dá vytvářet v podstatě libovolný počet replik.

Férovost a bezpečnost

Bohužel, pStore nemá implementované žádné techniky, které by mohly zaručit férovost vztahů mezi uživateli. Nic nebrání uživatelům v přetěžování systému obrovským množstvím svých dat nebo v nějakých jiných způsobech zneužití systému. Toto je pravděpodobně spojeno s tím, že hlavním cílem vývojářů pStore bylo pouze ověření technologie, nikoliv vytvoření prakticky použitelného programu.

2.2 Cooperative Internet Backup Scheme

Úvod

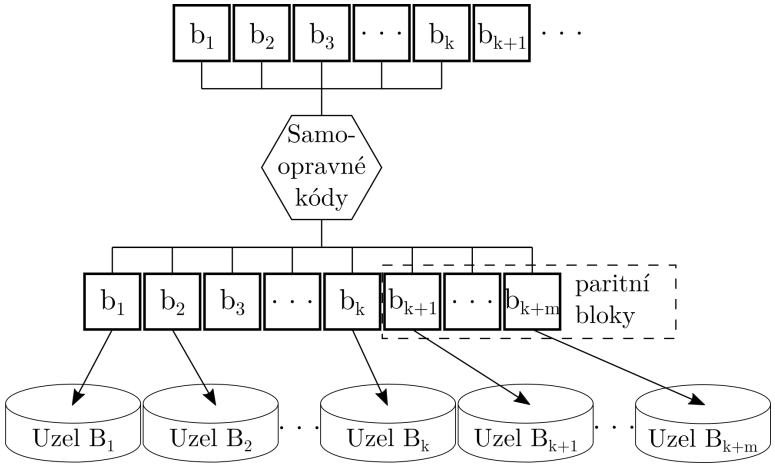
A Cooperative Internet Backup System [4] (CIBS) je poněkud zvláštním zálohovacím systémem, jelikož nabízí uživatelům pouze prostor („virtuální“ disk) pro ukládání svých dat, a o nic jiného se nestará. Systém klade velký důraz na zajištění férových vztahů mezi uživateli sítě. Hlavní myšlenkou CIBS je, že se výměna volného místa mezi uživateli provádí symetricky. Při vytváření partnerských vztahů si mezi sebou uživatelé sdílejí informace o tom kdy bývají obvykle k dispozici, kolik místa budou potřebovat a v jakých časových úsecích očekávají dostupnost svých dat. Jakmile se uživatelům podaří dohodnout na provozních podmínkách, mohou začít ukládat svá data v partnerských uzlech.

Zálohování

Na rozdíl od pStore se v CIBS partnerské uzly hledají pomocí centrálního serveru. Zálohování dat se v CIBS probíhá následovně: uzel hledající partnery pošle na server inicializační dotaz, obsahující popis základních vlastností požadovaného partnera. Server uchovává informaci o všech uzlech sítě ve své databázi a ve své odpovědi pošle seznam potenciálních kandidátů. Po obdržení odpovědi od serverů, uzel začne postupně kontaktovat uzly uvedené v seznamu a bude se snažit navázat vhodné partnerské vztahy.

Replikace

CIBS se snaží vybudovat „virtuální disk“ z jednotlivých uzlů sítě, a proto ke zvýšení spolehlivosti disku používá blokové samoopravné kódy (*erasure codes*) (viz. Obrázek 2.3). Stejné kódy se například používají v některých RAID metodách. Tyto samoopravné kódy dokáži detekovat chybějící bloky a obnovit jejich obsah z informace uchovávané v jiných v blocích. Efektivita těchto kódů je přímo úměrná počtu použitých fragmentů. Na jednu stranu toto znamená, že každý uzel může mít hodně partnerů a tím snížit nároky na místo, na druhou stranu by toto ovšem vedlo ke zvýšení celkové sítové zátěže a latence, jelikož uzel bude potřebovat více fragmentů k obnovení zálohy. A



Obrázek 2.3: „Virtuální disk“ s použitím samoopravných kódů

právě proto není vůbec jednoduché rozhodnout o tom, kolik fragmentů je potřeba použít, a závisí to spíše na tom, zda jsme ochotni obětovat průchodnost systému nebo diskový prostor.

Férovost a bezpečnost

Zajištění férových vztahů mezi uživateli je hlavní prioritou CIBS, proto navrhuje několik zajímavých technik pro vypořádání s černými pasažéry a zlomyslnými uživateli. Jednou z takových technik je klasifikace partnerů. V CIBS každý uzel je klasifikován bud jako „dobrý“, nebo jako „špatný“. Za „špatné“ jsou považovány uzly které:

- Ztrácejí data svých partnerů.
- Periodicky porušují své sliby tím, že nebývají k dispozici ve slíbenou dobu.

Partneři se mohou navzájem kontrolovat periodickým posíláním žádostí o zaslání náhodně vybraných bloků. Pokud se nějaký uzel ukáže být „špatným“, pak budou veškerá jeho data okamžitě smazána z disků všech jeho partnerů a informace o něm se dostane na černou listinu. Použití černé listiny zaručuje, že v budoucnu již nebude docházet ke kontaktu se „špatnými“ partnery.

2.3 Pastiche

Úvod

Pastiche [5] je zajímavou kombinací myšlenek použitých v systémech pStore a CIBS. Pastiche byl vyvinut vědci z Michiganské Univerzity v roce 2002.

Zálohování

Zálohování dat v Pastiche probíhá stejně jako v pStore. Každý soubor je rozdělen do několika bloků. Pak nasleduje šifrování každého bloku konvergentním šifrováním, aby bylo možné sdílet stejné bloky mezi více uživatele. Pro každý ze zálohovaných souboru se vygenerují metadata obsahující informaci o všech blocích a o původní struktuře souboru. Tyto bloky a metadata se pak předávají partnerským uzlům.

Distribuce dat

Způsob distribuce dat v Pastiche se již více podobá způsobům použitým v CIBS. K nalezení partnerských uzlů v Pastiche se používají dvě DHT. Za nejvhodnější partnery jsou považovány ty uzly, které mají největší překryv v datech s daným uživatelem. Potom k vytvoření plné zálohy svých dat na disku partnera každému uživateli stačí poslat svému partnerovi pouze malou množinu u něj chybějících bloků. Nalezení partnerů v Pastiche se uskutečňuje za pomocí centrálního serveru.

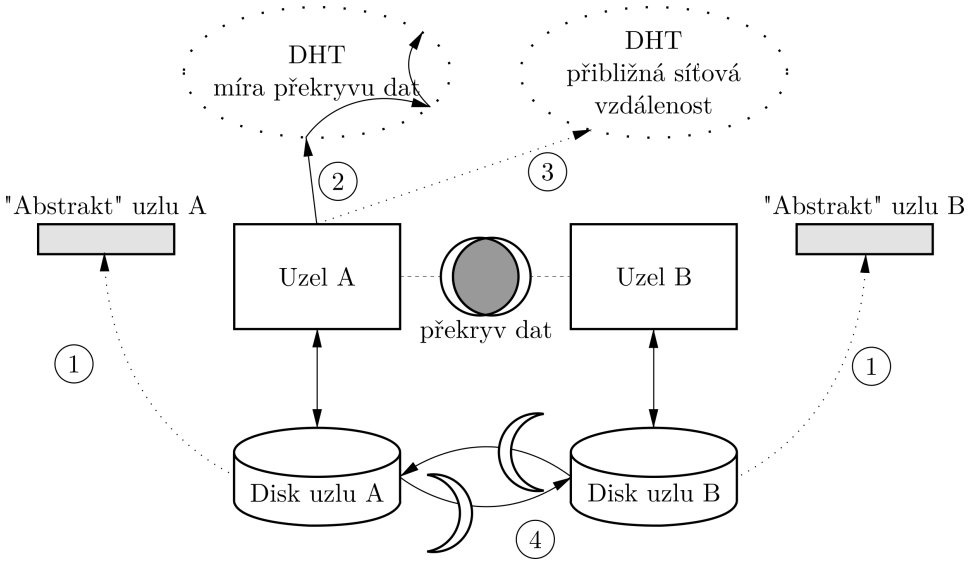
Každý nový uzel sítě se přidává do dvou DHT. První DHT je uspořádána podle odhadu sítové vzdálenosti mezi uzly, a druhá je organizována podle odhadu míry překryvu dat mezi jednotlivými uzly. Druhá DHT se používá pouze v případech, kdy se nepodaří najít dostatečný počet partnerů v první tabulce. Proces nalezení partnerů v Pastiche je zobrazen na Obrázku 2.4 a funguje následovně:

1. Nejprve hostitelský uzel vypočítává tzv. „abstrakt“ svého souborového systému. „Abstrakt“ je složen z kontrolních součtů několika náhodně vybraných souborů malé velikosti, a slouží jako „otisk“ dat daného uzlu.

2. Pak si hostitelský uzel začíná podle vypočteného „abstraktu“ vyhledávat partnery v první DHT. Při hledání se používá tzv. princip *lighthouse sweep*[2]. Hlavní myšlenka tohoto principu spočívá v tom, že hostitelský uzel odešle žádost o partnerství několika náhodně vybraným uzlům. Tato žádost obsahuje identifikační číslo hostitelského uzlu spolu s jeho „abstraktem“. Uzly, které obdrží podobnou žádost, spočítají míru podobnosti obsahu jejich souborového systému s „abstraktem“ uvedeným v žádosti a odešlou výslednou hodnotu zpět žádajícímu uzlu. Uzly s dostatečně velkou mírou podobnosti jsou přidány mezi partnery hostitelského uzlu.

3. Pokud se hostitelskému uzlu nepodaří najít potřebný počet partnerů ve 2. kroku, bude použita druhá DHT. Ovšem zde už není použít princip

[2]Princip *lighthouse sweep* je podrobně vysvětlen v [5] v Sekci 3.2 (Overlays: Finding a Set of Buddies)



Obrázek 2.4: Zálohovací proces v Pastiche

lighthouse sweep, místo toho jsou žádosti o partnerství poslány rovnou uzlům s největší mírou překryvu dat s hostitelským uzlem.

4. Jakmile bude potřebný počet partnerů nalezen, hostitelský uzel zahájí proces výměny dat se svými partnery.

Férovost a bezpečnost

Pastiche má stejný bezpečnostní model jako pStore. Jednotlivé bloky se šifrují za použití konvergentního šifrování. Dešifrovací symetrické klíče všech zašifrovaných bloků jsou zapsány do odpovídajících metadat. Metadata jsou zašifrována pomocí symetrického šifrovacího klíče odvozeného z uživatelského hesla.

Co se týče férovosti, Pastiche je na tom stejně jako CIBS. Pastiche také používá jedinou černou listinu, do které se postupně přidávají „špatné“ uzly. Uzly kontrolují své partnery periodickým posíláním žádostí o zaslání náhodně vybraných bloků. Uzly nesplňující svoje povinnosti jsou označeny jako „špatné“ a jsou přidány do černé listiny.

2.4 Resilio (BitTorrent Sync)

Popis

Resilio [6] je proprietární program vyvinutý ve firmě BitTorrent, Inc., který slouží pro P2P synchronizaci dat mezi několika zařízení, a používá BitTorrent

protokol. Resilio sice nikdy nebyl myšlen jako program pro zálohování dat, ale přesto jsme se rozhodli o něm tady zmínit, jelikož má několik zajímavých vlastností, které umožňují používat jej jako poměrně dobrý zálohovací nástroj. Hlavní nevýhoda Resilia spočívá ovšem v tom, že nejsou zveřejněny jeho zdrojové kódy. Vzhledem k tomu, že se Resilio prezentuje hlavně bezpečím, je to poměrně vážný nedostatek. Tvůrci Resilia ovšem tvrdí, že je docela možné, že někdy v budoucnu zdrojové kódy zveřejněny budou.

Synchronizace

Synchronizace v Resilio probíhá na bázi adresářů. Pokud uživatel chce sdílet nějaký adresář na svém počítači s jiným zařízením, musí ten adresář nejdříve přidat přes webové rozhraní. Informace o sdílení jsou v šifrované podobě odeslána na *tracker*. Zařízení, které se k adresáři chce připojit, musí znát 32-bitový klíč (*secret*).
Pro každý sdílený adresář jsou vygenerovány klíče dvou typů:

1. Pro neomezený přístup - klíč dává všem jeho držitelům právo pro čtení/zápis z/do synchronizovaného adresáře, a obsah adresáře je obousměrně synchronizovatelný.

2. Pouze pro čtení - obsah adresáře se synchronizuje pouze jednostranně.

Klíče je možné sdílet buď v textové reprezentaci nebo v podobě QR kódu. Pak každé zařízení, kterému uživatel poslal klíč, podle informací na *trackeru* dohledá a spojí se uživatelským počítačem, a začne synchronizovat.

Bezpečnost

Resilio má možnost šifrování synchronizovaných dat. Obsah každého adresáře se šifruje pomocí symetrického šifrovacího klíče odvozeného z přístupových klíčů vygenerovaných pro tento konkrétní adresář. Použití 32-bitových klíčů výrazně snižuje pravděpodobnost jejich uhodnutí.

Kapitola 3

Návrh systému BTVault

Tato kapitola popisuje návrh zálohovacího systému BTVault. Napříč celou kapitolou budeme dodržovat následující terminologii:

Uživatel člověk používající systém.

Klient front-end systému používaný uživatelem.

Uzel počítač, na kterém běží instance BTVaultu.

Smlouva dvoustranná dohoda mezi uzly sítě.

Partner jeden z účastníků smlouvy.

3.1 Základní rysy aplikace

3.1.1 Zálohování

Uživatel může pomocí klienta vybrat data, které je potřeba zazálohovat. Vybraná data jsou zašifrována a rozdělena do fragmentů. Pro úspěšné zálohování vybraných dat musí uzel najít dostatečné množství „úložného prostoru“ v Peer-to-Peer (P2P) síti. Uzel získává „úložný prostor“ obchodováním volným prostorem na svém pevném disku s ostatními uzly P2P sítě. Uživatel určuje horní limit pro množství diskového prostoru dostupného k obchodování. Jakmile se uzlu podaří získat potřebné množství „úložného prostoru“, dojde k distribuci vytvořených fragmentů. Celý proces zálohování je mnohem detailněji popsán v Sekci 3.6.

3.1.2 Obnovení zálohy

Po úspěšném vytvoření zálohy si uživatel může pomocí klienta nechat zobrazit veškeré uložené soubory spolu s jim přiřazenými unikátními identifikačními čísly. Uživatel si muže také prohlédnout všechny dostupné verze jednotlivých souborů. Tímto způsobem je uživatel schopen specifikovat přesně

která data je potřeba obnovit. Proces obnovení každého vybraného souboru probíhá následovně: klient nejdříve určí umístění jednotlivých fragmentů patřících danému souboru, pak tyto fragmenty stáhne, dešifruje a složí je dohromady. Celý proces znovuobnovení dat je mnohem detailněji popsán v Sekci 3.7.

3.1.3 Optimální obchodování

Pro zaručení férovosti vztahů mezi uzly sítě se v BTVaultu používá jednoduchý reputační systém, který by měl chránit systém proti zneužití černými pasažéry a zlomyslnými uzly. Uzly sítě během obchodování vždy berou v úvahu reputaci svých potenciálních partnerů, což se pak odráží v kapacitě nabízeného diskového prostoru. Při sjednávaní partnerství mezi dvěma uzly se případný reputační rozdíl kompenzuje poskytnutím dodatečného diskového prostoru uzlem s nižší reputaci. Tímto způsobem máme garantováno, že kvalita služeb poskytovaných uzlům sítě je přímo úměrná celkové přínosnosti těchto uzlů v rámci systému. Detailnější popis reputačního systému a celého procesu obchodování lze najít v Sekci 3.2.

3.1.4 Bezpečnost

Jelikož se uživatelská data uchovávají na pevných discích vzdálených počítačů, je důležité zaručit jejich zabezpečení a omezit k nim přístup nežádoucím osobám. Z tohoto důvodu se veškerá uživatelská data před odesláním šifrují, a to takovým způsobem, že dešifrovat je schopen pouze jejich majitel. Více informací o bezpečnostním modelu najdete v Sekci 3.4.

3.1.5 Dostupnost dat

Aby byli uživatelé systému schopni vytvářet pravidelné zálohy a poskytovat zálohovací služby ostatním členům sítě, musí být přítomni v systému poměrně často. Pokud uživatel není dostupen již delší dobu, jeho partneři se mohou rozhodnout jej potrestat smazáním jeho záloh. Dále BTVault používá mechanismus pro ověření toho, že všechny uzly opravdu splňuji své povinnosti a uchovávají svěřená jím data. Za nesplnění nebo porušení povinností uzlům hrozí trest ze strany jejich partnerů. Více informací o bezpečnostním modelu najdete v Sekci 3.5.

3.2 Optimální obchodování

3.2.1 Reputační systém

Reputace odráží kvalitu daného uzlu z hlediska ostatních uzlů sítě. Kvalitou uzlu v daném případě myslíme míru toho, jak často je daný uzel *on-line* tj.

je připojen k Internetu a poskytuje zálohovací služby. Informace o reputacích všech uzlů P2P sítě se uchovává na centrálním serveru a je dostupná k nahlédnutí všem uživatelům systému.

Uzel může odvodit reputaci jiných uzlů dvěma způsoby: buď přímým pozorováním, nebo tázáním centrálního serveru. Začneme prvním případem, kdy uzel i odhaduje reputaci uzlu j na základě vlastních zkušeností. Dejme tomu, že uzel j uchovává u sebe na disku zálohu uzlu i. Pak může uzel i periodicky posílat uzlu j různé zkoušky (*challenges*), čímž ověří, že uzel j je *on-line* a že opravdu uchovává u sebe potřebnou zálohu. Záznamy o úspěšnosti těchto zkoušek budou zapsány do lokální databáze uzlu i. Pokud celkový počet interakci mezi uzly i a j označíme jako t_{ij} a počet úspěšných interakci označíme jako s_{ij}, můžeme spočítat reputaci uzlu j z hlediska uzlu i podle vzorce:

$$d_{ij} = \frac{s_{ij}}{t_{ij}} \tag{3.1}$$

Uzel i může také získat reputaci uzlu j od serveru. Tento způsob se používá v případech, kdy se uzly i a j setkávají poprvé. Všechny uzly periodicky posílají na server informaci o svých zkušenostech s ostatními uzly sítě. Například, uzel a může poslat informaci o svém partnerovi b ve formátu $\langle d_{ab}, t_{ab}\rangle$. Pak je server schopen dopočítat reputaci uzlu j z informací získaných od jeho partnerů P podle následujícího vzorce:

$$e_j = \begin{cases} \frac{\sum_{k\in P} e_k d_{kj} t_{kj}}{\sum_{k\in P} t_{kj}}, & \text{pokud } \sum_{k\in P} t_{kj} \geq 10 \\ 0.5, & \text{jinak (t.j. uzel } j \text{ je nový)} \end{cases} \tag{3.2}$$

Uzel i pak může z d_{ij} a e_j spočítat celkovou reputaci uzlu j podle vzorce:

$$r_{ij} = \alpha d_{ij} + (1-\alpha)e_j, \text{ kde } \alpha = \min\{\frac{t_{ij}}{n}, 1\}, \tag{3.3}$$

kde n je minimální počet interakcí potřebných k tomu, aby uzel i byl schopen samostatně soudit o chování uzlu j. Číslo n musí být dostatečně velké, aby uzel i měl možnost získat dostatečné zkušenosti s uzlem j (momentálně $n = 100$).

3.2.2 Optimální obchodní politika

Předpokládejme, že se uzel A s reputací r_A chce stát partnerem uzlu B s reputací r_B. Potřebujeme zjistit, jakou velikost prostoru s_B musí získat uzel A od uzlu B v případě, že uzel A nabízí prostor velikostí s_A. Pokud mají uzly A a B stejnou reputaci (tj. $r_A = r_B$), oba uzly obdrží prostory stejné velikosti s_A. Pravděpodobnost toho, že některý z uzlu není *on-line* je $1 - r_A$. Pravděpodobnost toho, že žádný z uzlů není *on-line*, je rovna $(1 - r_A)^2$. Odtud můžeme odvodit, že dostupnost zálohy je v tomto případě rovna

$$R_{(r_A=r_B)} = 1 - (1 - r_A)^2. \tag{3.4}$$

Dostupností zálohy zde myslíme pravděpodobnost toho, že v daný okamžik je požadovaná záloha dostupná buď z uzlu A, nebo z uzlu B.

Pokud má uzel B nižší reputaci než uzel A (tj. $r_A > r_B$), uzel A musí jako kompenzaci dostat vetší prostor pro uložení zálohy. V takovém případě $s_B = ks_A$, kde $k \geq 1$. Dostupnost zálohy se bude rovnat

$$R_{(r_A > r_B)} = 1 - (1 - r_A)(1 - r_B)^k. \tag{3.5}$$

Jelikož potřebujeme zaručit aby $R_{(r_A = r_B)} = R_{(r_A > r_B)}$, dostáváme následující rovnici:

$$1 - (1 - r_A)^2 = 1 - (1 - r_A)(1 - r_B)^k. \tag{3.6}$$

Odkud po vyřešení pro neznámé k dostáváme

$$k = \log_{(1-r_B)}(1 - r_A). \tag{3.7}$$

Optimální obchodní politika pro uzel A vypadá tudíž následovně:

$$s_B = s_A \log_{(1-r_B)}(1 - r_A), \tag{3.8}$$

Vzorec 3.8 lze také zapsat jako:

$$\frac{s_B}{s_A} = \frac{\log(1 - r_A)}{\log(1 - r_B)}. \tag{3.9}$$

3.2.3 Obchodní schéma

Pro uložení svých záloh každý uzel potřebuje získat určité množství „úložného prostoru“ v síti. Uživatele mohou přispět svým lokálním volným místem na disku do systému výměnou za spolehlivé úložiště jejich dat u jiných uživatelů. Přičemž poměr mezi množstvím poskytnutého prostoru a získaného prostoru závisí na reputaci daných uživatelů a určuje se podle vzorce 3.8.

Uživatelé specifikují, jakou část volného prostoru na pevných discích jsou ochotni poskytnout ostatním členům sítě a uzly si pak mezi sebou obchodují diskovým prostorem v souladu s obchodním protokolem. Podle tohoto protokolu se každé obchodní jednání uskutečňuje mezi dvěma uzly P2P sítě. Výsledkem úspěšně ukončeného jednání je *smlouva*, která specifikuje kapacitu diskového prostoru alokovaného každou stranou pro potřeby jiné strany. Mezi dvěma uzly může být uzavřeno nekonečné množství podobných smluv.

Například, pokud má uzel A sjednanou smlouvu c s jiným uzlem B, uzel A má právo použit diskový prostor poskytnutý uzlem B v rámci smlouvy c. Potom může uzel A může poslat uzlu B svoje data d ve tvaru $\langle d, c \rangle$, čímž naznačí použití smlouvy c. V tomto případě budou data d propojena se smlouvou c.

Smlouvy se vždycky uzavírají na dobu určitou s možností prodloužení. Ukončením platnosti smlouvy se ukončuje i povinnost stran uchovávat data spojená s touto smlouvou.

3.3 Organizace dat v síti

3.3.1 Správa záloh a metadat

V systému BTVault spravujeme data dvou typu: samotná uživatelská data a speciální meta-soubory sloužící k nalezení zazálohovaných dat v síti. Jak samotná data, tak i metadata se ukládají do sítě. Podstatným rozdílem je ovšem to, že data se uchovávají přímo v souborových systémech partnerských uzlů, kdežto metadata se vkládají do distribuované hašovací tabulky (DHT) tvořené všemi uzly sítě.

Hlavní výhodou takového přístupu je minimalizace síťové komunikace mezi uzly a vyhnutí se režijním nákladům spojeným s častou migraci dat. Pokud bychom totiž do DHT ukládali kromě metadat ještě i data samotná, jak to dělá například pStore, museli bychom pak přeposílat stovky megabajtů pokaždé když dojde ke vzniku/zániku nějakého uzlu [7]. Na druhou stranu je velikost metadat většinou velice malá (v poměru se samotnými daty skoro zanedbatelná), takže si můžeme dovolit použití DHT pro jejich ukládání. Kromě toho uživatelé systému potřebují mít neustálý přístup ke svým metadatům, protože metadata obsahují informaci o všech uživatelských zálohách a vztazích s jinými členy sítě. Uložením metadat do DHT jí zároveň přenecháváme veškeré starosti spojené se zajištěním non-stop dostupností těchto dat (jejich replikaci a případnou migraci).

3.3.2 Organizace metadat

BTVault udržuje několik typu metadat. Uživatelé systému potřebují mít informaci o zazálohovaných souborech a umístění fragmentů patřících těmto souborům. Uzly také potřebují mít neustálý přístup k informaci o uzavřených smlouvách s jinými uzly sítě. Jak již bylo vysvětleno výše, metadata obsahující tyto informace se ukládají do DHT. Tato subsekce popisuje jak jsou tyto metadata strukturovaná a v jaké podobě se uchovávají uvnitř DHT.

Struktura metadat

Meta-data jsou strukturována do několika vrstev. Zavedení více vrstev umožňuje uživatelům vytahovat potřebnou informaci z DHT postupně, po malých částech. Kdybychom totiž veškerou uživatelskou meta-informaci uchovávali v jednom velkém meta-souboru, museli bychom pak vytahovat/vkládat z/do DHT celý meta-soubor, bez ohledu na to, jakou velkou část z té meta-informace uživatel vlastně potřebuje. Mimo jiné, vícevrstvá struktura metadat umožňuje jejich sdílení mezi uzly.

Je třeba ovšem poznamenat, že podobné strukturování má i určité nevýhody. Hlavní nevýhodou je zvýšení počtu vyhledávání v DHT potřebných pro přístup k informacím obsazených v nižších vrstvách. Tento problém

se částečně řeší lokálním cachováním metadat přímo v souborových systémech uzlů. Jiným nedostatkem vícevrstvé struktury je potřeba zajištění konzistence metadat napříč všemi vrstvami. Jinými slovy, vnesení jakýchkoliv úprav v jedné vrstvě často vynucuje aktualizaci metadat i na vyšších vrstvách. Je patrné, že závažnost těchto problémů roste přímo úměrně s počtem zavedených vrstev.

Ve snaze minimalizovat výše zmíněné nevýhody jsme v BTVaultu zavedli čtyřvrstvé strukturování metadat. První vrstva uchovává dva druhy uživatelské informace: seznam zazálohovaných souborů (*file list*) a seznam smluv (*contract list*). Ve druhé vrstvě udržujeme seznamy bloků souborů (*file block lists*) a uživatelské smlouvy (*contracts*). Do třetí vrstvy vkládáme seznamy fragmentů bloků (*block fragment lists*). Čtvrtá vrstva obsahuje seznamy držitelů fragmentů (*holder lists*). Obecný přehled takového strukturování metadat je zobrazen na Obrázku 3.1. Veškerá metadata se ukládají do DHT v podobě dvojic klíč-hodnota $\langle k, v \rangle$. Dále následuje detailnější popis jednotlivých druhů metadat:

File list (FL)
obsahuje výčet všech uživatelem zazálohovaných souborů v podobě jejich absolutních cest. Do DHT se vkládá v následující podobě [1]:

$$\langle k, v \rangle = \langle \texttt{sha-2}(K' \parallel U \parallel \text{sůl}),\ IV \parallel \texttt{aes-gcm}(FL, \texttt{hkdf}(K'), IV) \rangle.$$

Contract list (CL)
obsahuje seznam identifikačních čísel všech uživatelských smluv. Do DHT se vkládá v následující podobě:

$$\langle k, v \rangle = \langle \texttt{sha-2}(K' \parallel U \parallel \text{sůl}),\ IV \parallel \texttt{aes-gcm}(CL, \texttt{hkdf}(K'), IV) \rangle.$$

File block list (FBL)
obsahuje následující informace o souboru: název, velikost, přístupová práva a seznam dostupných verzí. Každá verze se skládá z data vytvoření zálohy a seznamu s informací (hash, umístění uvnitř souboru

[1] Použitá notace:

U – uživatelské jméno
K – veřejný klíč uživatele
K' – soukromý klíč uživatele
IV – náhodně vygenerovaný inicializační vektor
$\vec{0}$ – nulový inicializační vektor $(0, \ldots, 0)$
$\parallel$ – binární operace spojení řetězců
$\texttt{sha-2}(d)$ – kryptografická hashovací funkce
$\texttt{aes-gcm}(d, k, iv)$ – bloková šifra AES v režimu provozu GCM
$\texttt{aes-ctr}(d, k, iv)$ – bloková šifra AES v režimu provozu CTR
$\texttt{hkdf}(k)$ – funkce pro odvození šifrovácích klíčů založena na HMAC

a velikost) o všech blocích patřících dané verzi souboru. Do DHT se vkládá v následující podobě:

$$\langle k, v \rangle = \langle \texttt{sha-2}(K' \parallel \text{soubor}),\ IV \parallel \texttt{aes-gcm}(FBL, \texttt{hkdf}(K'), IV) \rangle.$$

Contract (C)
obsahuje následující informace o smlouvě: identifikační čísla obou stan, sjednané kapacity vyměněných diskových prostorů, seznamy fragmentů spojených s konkretní smlouvou. Do DHT se vkládá v následující podobě:

$$\langle k, v \rangle = \langle \texttt{sha-2}(K' \parallel \text{smlouva}),\ IV \parallel \texttt{aes-gcm}(C, \texttt{hkdf}(K'), IV) \rangle.$$

Block fragment list (BFL)
obsahuje seznam s informací (hash, umístění uvnitř bloku) o všech fragmentech patřících danému bloku. Do DHT se vkládá v následující podobě:

$$\langle k, v \rangle = \langle \texttt{sha-2}(\texttt{sha-2}(\text{blok})),\ \texttt{aes-ctr}(BFL, \texttt{sha-2}(\text{blok}), \vec{0}) \rangle.$$

Holder list (HL)
obsahuje seznam s identifikačními čísly všech držitelů daného fragmentů. Do DHT se vkládá v následující podobě:

$$\langle k, v \rangle = \langle \texttt{sha-2}(\texttt{sha-2}(\text{fragment})),\ \texttt{aes-ctr}(HL, \texttt{sha-2}(\text{fragment}), \vec{0}) \rangle.$$

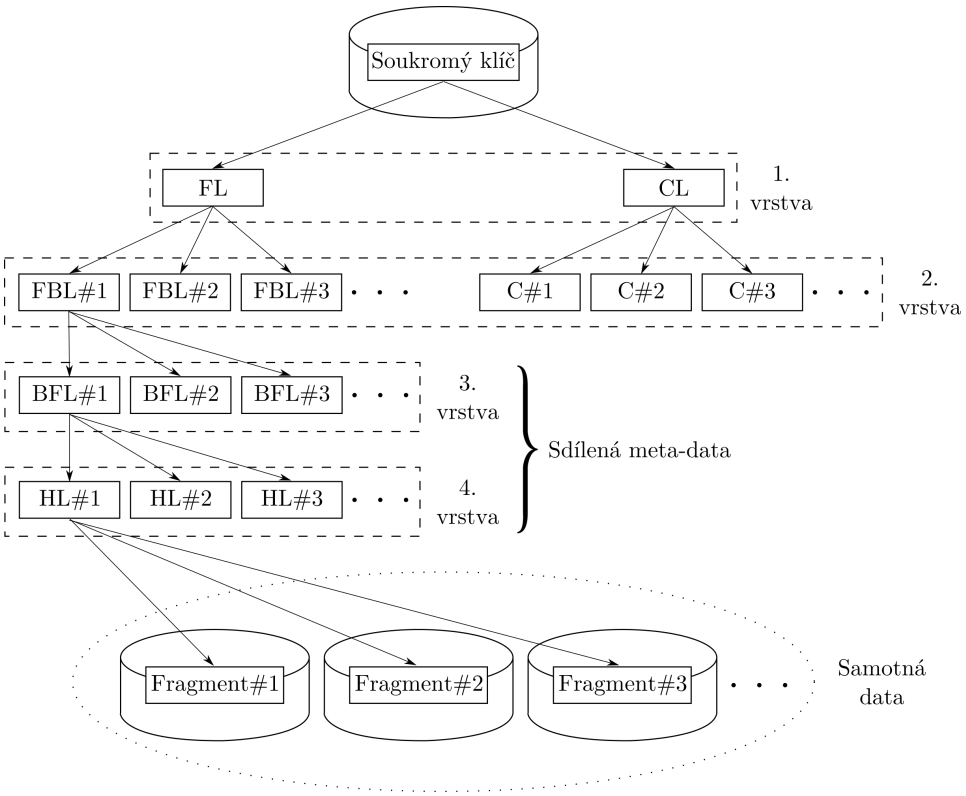
3.3.3 Organizace záloh

V BTVaultu se uživatelský soubor typicky neuchovává v síti jako celek, místo toho se rozdistribuovává po celé síti v podobě malých ($<$ 1MB) fragmentů. Hlavní přínos tohoto přístupu je v tom, že umožňuje použití efektivních technik pro řešení problému s dostupnosti dat. Kupříkladu je zřejmé, že je mnohem efektivnější replikovat malé fragmenty do několika uzlů v síti, než je tomu v případě celých souborů, jejichž velikost není nijak omezená.

Další výhodou rozdělení souborů do fragmentů je to, že nám dovoluje zrychlit proces obnovení záloh, a to díky malé velikosti samotných fragmentů. Umožňuje nám totiž paralelizovat celý proces stahování dat ze sítě, tedy jsme schopni současně stahovat několik fragmentů od několika uzlů.

Dále, při nasekání dat do menších fragmentů existuje určitá možnost, že někdy dojde k vytvoření identických fragmentů. Její pravděpodobnost je tím větší, čím menší je velikost vytvářených fragmentů. Identické fragmenty pak mohou být uložené v síti pouze jednou, a jejich vlastníci je tak mohou efektivně sdílet mezi sebou.

V BTVaultu se snažíme maximálně využít všech výše uvedených výhod, proto se celý proces fragmentace uživatelských dat provádí ve dvou fázích:



Obrázek 3.1: Organizace dat v systému BTVault.

1. Rozdělíme každý soubor do několika bloků různé velikosti.

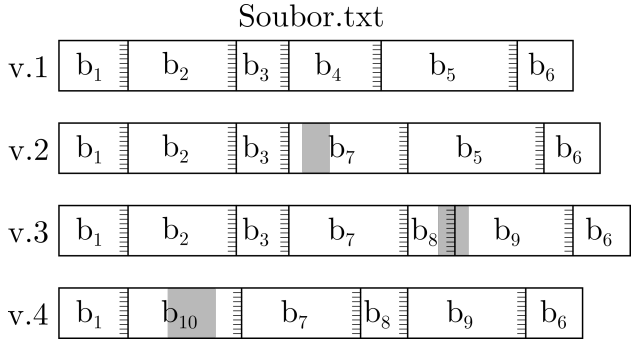
2. Rozdělíme každý z bloků vytvořených v 1. fázi do čtyř fragmentů fixní velikosti.

Dále následuje detailnější rozbor každé fáze.

Rozdělení souborů do bloků

V první fázi se nasekání souborů provádí na základě jejich obsahu. Používáme k tomu tzv. metodu posuvného okna [8] (*sliding window algorithm*). Tato metoda je užitečná tím, že vytváří bloky, které jsou odolné vůči posunům (*shift-resistant*) uvnitř souborů, což výrazně usnadňuje identifikaci duplicitních bloků během inkrementálního zálohování. Příklad takového rozdělení je názorně ilustrován na obrázku 3.2. Samotný proces rozdělení do bloků pak probíhá podle následujícího algoritmu:

1. Definujeme kladné celé číslo d, které bude reprezentovat střední velikost bloků (např. pro bloky velikosti $\approx$ 2MB použijeme $d = 2^{11}$).



Obrázek 3.2: Vývoj bloků souboru `Soubor.txt` mezi jeho různými verzemi. Horizontální proužky demonstrují koncové posloupnosti 48 bajtů. Šedou barvou jsou označeny změněné části souboru.

2. Definujeme pohyblivé okno fixní délky m bajtů (typicky $m = 48$) a nastavíme jej na začátek souboru.

3. Pro danou posloupnost bajtů uvnitř pohyblivého okna $P = [\,p_0, p_1, \ldots, p_{m-1}\,]$:

   (a) Vypočteme hodnotu Rabinova otisku [9] (*Rabin Fingerprint*) posloupnosti P podle vzorce:

   $$\texttt{Rabin}(P) = \texttt{Rabin}([\,p_0, p_1, \ldots, p_{m-1}\,]) = (\sum_{i=0}^{m-1} q^{m-(i+1)} p_i) \bmod d,$$

   kde q je náhodně vybrané prvočíslo.

   (b) Pokud se hodnota $\texttt{Rabin}(P)$ rovná $d - 1$, označíme posloupnost P jako hranici bloku a oříznememe soubor na pozici p_{m-1}.

4. Pokud jsme na konci souboru, ukončíme běh algoritmu.

5. Jinak posuneme okno o jeden bajt dále a skočíme na krok č. 3.

Je zřejmé, že výše uvedený algoritmus má časovou složitost $O(mn)$, kde n je velikost zpracovávaného souboru v bajtech. Jelikož ale používáme Rabinovy otisky, v praxi lze tento algoritmus implementovat tak, aby ve většině případů běžel mnohem rychleji. Například pokud si na začátku algoritmu předpočteme hodnoty q^k pro všechny možné $k \in \{0, \ldots, 255\}$, zpracovávaní souboru velikosti 1GB na moderních procesorech zabere jen pár desítek milisekund.

Rozdělení bloků do fragmentů

Ve druhé fázi se každý blok souboru velikosti s bajtů rozdělí do čtyř fragmentů fixní velikosti $\lceil \frac{s}{4} \rceil$ bajtů, přičemž případné zbylé místo v posledním fragmentu vyplníme nulovými bajty. Fixní velikost fragmentů nám pak

umožní použití sofistikovanějších algoritmů pro zvýšení spolehlivosti systému a zajištění dostupnosti záloh. Podrobný popis těchto technik najdete v Sekci 3.5.

3.4 Bezpečnost

Jelikož se veškerá data v P2P sítích uchovávají na pevných discích nedůvěryhodných uzlů, potřebujeme tyto data nějakým způsobem ochránit. V BTVaultu se všechna uživatelská data před odesláním šifrují, a to takovým způsobem, že dešifrování těchto dat je schopen provést jedině jejich majitel.

Každý uživatel systému BTVault vlastní jeden šifrovací klíč, který se skládá ze dvou části: veřejné a soukromé. Veřejná část klíče se odesílá do centrálního serveru při registraci a je veřejně přístupná. Veřejné části se především používají k ověření digitálních podpisů vygenerovaných pomocí odpovídajících soukromých částí.

Jak již bylo naznačeno v Sekci 3.3, privátní uživatelská metadata se šifrují algoritmem AES-GCM [10], který zajišťuje jejich důvěrnost, integritu a autentizaci. Šifrovací klíč se v tomto případě odvozuje z privátní části uživatelského klíče pomocí funkce HDKF [11]. Inicializační vektor se generuje náhodně a na konci se připojuje k zašifrovaným datům.

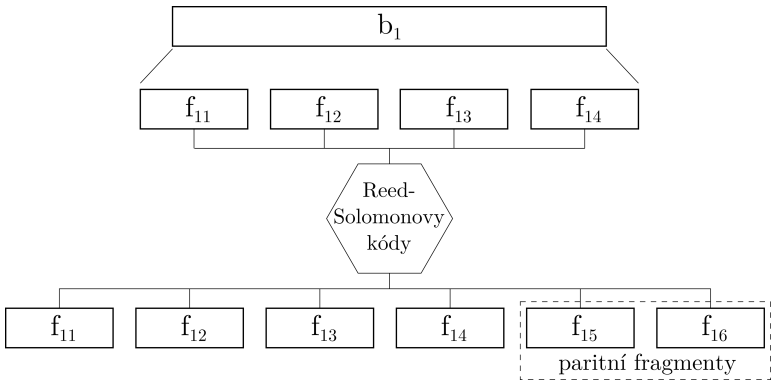
Samotná uživatelská data a sdílená metadata se šifrují algoritmem AES-CTR [12]. Šifrovací klíč se odvozuje přímo ze šifrovaných dat pomocí kryptografické hašovací funkce (např. SHA-2). Takovému přístupu se v odborné literatuře ještě někdy říká konvergentní šifrování [3] (*convergent encryption*). Fakt, že se šifrovací klíč generuje přímo z obsahu dat, přináší ihned několik výhod:

- Schopnost správně dešifrovat data implikuje jejich integritu.
- Umožňuje sdílení stejných dat mezi uzly (resp. deduplikace dat), jelikož identické soubory jsou zašifrovány stejně.

3.5 Dostupnost dat

Jak již bylo zmíněno v Sekci 3.3, veškerá metadata se vkládají do DHT. Tím pádem všechny záležitosti spojené s udržováním metadat a zaručením jejich nepřetržité dostupnosti přenecháváme DHT. DHT zajišťuje dostupnost metadat udržením ihned několika replik těchto metadat a jejich případným přemístěním z méně spolehlivých uzlů do více spolehlivých.

Jelikož se ovšem samotná uživatelská data neukládají do DHT, o jejich dostupnost se má postarat systém BTVault. Obecně v P2P sítích není možné nikdy 100% garantovat neustálou dostupnost všech uchovávaných dat, a to kvůli dynamické podstatě podobných sítí. Je totiž poměrně těžké předpovědět chování a stabilitu jednotlivých uzlů, obzvlášť pokud se jedná o rela-



Obrázek 3.3: Příklad rozdělení bloku b_1 do fragmentů.

tivně „nové“ členy sítě. Proto v BTVaultu pro zvýšení spolehlivosti systému a snížení pravděpodobnosti ztráty dat používáme kombinaci dvou nejpopulárnějších technik zabezpečení dat: samoopravných kódů (*erasure codes*) a klasické replikace.

Ze Sekce 3.3 víme, že uživatelská data se nejdříve dělí do bloků, a každý blok se následně rozděluje do čtyř fragmentů fixní velikosti. Z těchto čtyř fragmentů se pomocí Reed-Solomonových [13] (RS) kódů generují další dva paritní fragmenty, jak je ukázáno na Obrázku 3.3. Velká výhoda RS kódů spočívá v tom, že umožňují obnovení původního obsahu každého bloku z libovolné 4-členné kombinace jeho fragmentů. Jinými slovy pro každý konkrétní blok jsme schopni tolerovat ztrátu libovolných dvou fragmentů patřících danému bloku.

Tento přístup ovšem má i jednu očividnou nevýhodu: pokud dojde ke ztrátě nějakého fragmentů v síti, jeho obnovení bude vyžadovat přístup ke všem zbylým fragmentům bloku. A jelikož v P2P sítích ztráta dat je nevyhnutelná a docela běžná situace, potřebujeme mnohem efektivnější způsob, jak se vypořádat se ztrácenými fragmenty. Jedním z možných řešení tohoto problému je replikace fragmentů do několika uzlů sítě. V systému BTVault udržujeme k (momentálně $k = 5$) replik všech fragmentů, přičemž tolerujeme možnou ztrátu $i < k$ (momentálně $i = 2$) replik jednotlivých fragmentu. V drtivé většině případu se ztráta fragmentů bude řešit jednoduchým přeposláním kopií již existujících replik. Jen ve velice řídkých případech, kdy dojde k současné ztrátě všech k replik, se budeme muset uchýlit k „dražšímu“ způsobu obnovení obsahu fragmentu vyžadujícího stáhnutí všech ostatních fragmentů bloku.

3.5.1 Kontrolování partnerů

Abychom mohli zaručit dostupnost uživatelských dat v síti, potřebujeme neustále kontrolovat jejich existenci a dosažitelnost. V BTVaultu pro tyto

účely používáme zkouškový mechanismus (*challenge mechanism*), pomocí kterému všechny uzly jsou schopni kontrolovat své partnery. Tento mechanismus slouží především k ověřování toho, zda všichni partneři opravdu splňují svoje smluvní povinnosti a uchovávají jím svěřená data.

Zkoušky se vytvářejí během procesu zálohování dat, před samotnou distribuci dat do partnerských uzlů. Nechť uzel A chce uložit množinu fragmentů F na disku uzlu B na základe smlouvy c. Pak je seznam zkoušek Z vygenerován podle následujícího algoritmu:

1. Vybereme náhodnou neprázdnou variaci fragmentů V z množiny F.
2. Vygenerujeme náhodné číslo r.
3. Spočteme hash $H(V, r)$ podle následujícího rekurentního vzorce [2]:

$$H(V,r) = \begin{cases} r, \text{ pokud } \mathbf{len}(V) = 0, \\ \texttt{sha-2}(\mathbf{hd}(V) \parallel H(\mathbf{tl}(V), r)), \text{ jinak.} \end{cases} \tag{3.10}$$

4. Přidáme trojici $\langle V, r, H(V,r))\rangle$ do seznamu Z.
5. Pokud je počet elementů v Z roven n, ukončíme běh algoritmu.
6. Jinak skočíme na krok č. 1.

Ve výše uvedeném algoritmu číslo n musí být dostatečně velké, aby nedocházelo k opakovanému použití zkoušek (momentálně $n = 100$).

Samotné kontrolování partneru pak probíhá následujícím způsobem: uzel A periodický vybírá nějakou dvojici $\langle V, r\rangle$ ze seznamu Z a žádá svého partnera B o výpočet $H(V, r)$. Žádost se odesílá v podobě trojice $\langle V, r, c\rangle$. Tímto způsobem bude uzel B schopen ověřit pravost požadavku. Uzel A pak porovnává odpověď svého partnera s hodnotou $H(V, r)$ uloženou v Z. Jak již bylo zmíněno v Sekci 3.2, záznamy o úspěšnosti těchto pravidelných kontrol se zapisují do lokální databáze uzlu. Pokud není uzel B delší dobu schopen prokázat plnění svých smluvních povinnosti (ať již z důvodu nedostupnosti nebo nezvadnutím zkoušek), tak mu hrozí trest ze strany uzlu A. Uzel A se totiž může rozhodnout vypovědět uzavřenou mezi uzly smlouvu c a smazat případnou zálohu uzlu B z disku. Neslušné chování uzlu B se také odrazí na jeho celkové reputaci v systému. Nízká reputace nedovolí uzlu B efektivně obchodovat s jinými uzly sítě, což mu zabrání v navázání dalších partnerství.

[2] Použitá notace:

$\mathbf{hd}(A)$ – funkce vracející první prvek posloupnosti (např. $\mathbf{hd}([a, b, c]) = a$)

$\mathbf{tl}(A)$ – funkce vracející zbytek posloupnosti bez prvního prvku (např. $\mathbf{tl}([a, b, c]) = [b, c]$)

$\mathbf{len}(A)$ – funkce vracející délku posloupnosti (např. $\mathbf{len}([a, b, c]) = 3$)

3.6 Proces zálohování

Uživatel zahájí proces zálohovaní spuštěním klientské aplikace. Zálohovací proces se skládá z několika etap:

1. Výběr souborů k zálohování.
2. Rozdělení souborů do bloků.
3. Deduplikace bloků.
4. Rozdělení bloků do fragmentů.
5. Připravení zálohovacích sad.
6. Distribuce zálohovacích sad do partnerských uzlů.

Tyto etapy se provádějí sekvenčně. Dále následuje podrobný popis každé etapy.

Výběr souborů k zálohování

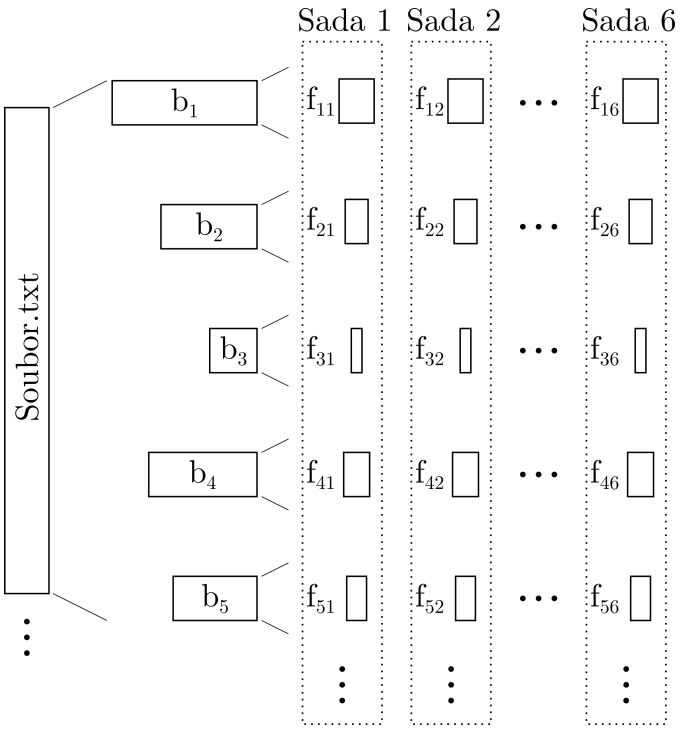
Než začneme samotný proces zálohování, potřebujeme nejdříve identifikovat soubory ve sledovaném adresáři, které se změnily od předešlé inkrementální zálohy. To provedeme porovnáním času poslední změny souborů s časem vytvoření jejich minulých záloh. Informaci o všech dostupných verzích pro konkretní soubor lze dohledat v příslušných metadatech. Pokud je zjištěno, že poslední verze byla vytvořená ještě před tím, než došlo k úpravě souboru, označíme soubor jako změněný a obnovíme jeho zálohu. V opačném případě budeme považovat soubor za nezměněný a vyřadíme jej z dalšího zpracovávaní.

Rozdělení souborů do bloků

Nejdříve potřebujeme nasekat velké ($>$ 1 MB) soubory na několik bloků menší velikosti. Rozdělení do bloků provedeme pro každý soubor zvlášť. Každý soubor se rozdělí do bloků na základě jeho obsahu, k čemuž použijeme Rabinovy otisky. Informaci o všech blocích souboru (hash, velikost, umístění uvnitř souboru) zapíšeme do speciálního meta-souboru *File Block List* (FBL).

Deduplikace bloků

Dále následuje proces deduplikace bloků. Nejprve se pro každý blok b zkontroluje existence jemu odpovídajícího meta-souboru *block fragment list* (BFL) v DHT. Ze Sekce 3.3 víme, že se BFL vkládají do DHT s klíčem $k =$ `sha-2`(`sha-2`(b)). Bloky s již existujícími BFL vyřadíme z dalšího zpracovávání.



Obrázek 3.4: Příklad vytvoření zálohovacích sad

Rozdělení bloků do fragmentů

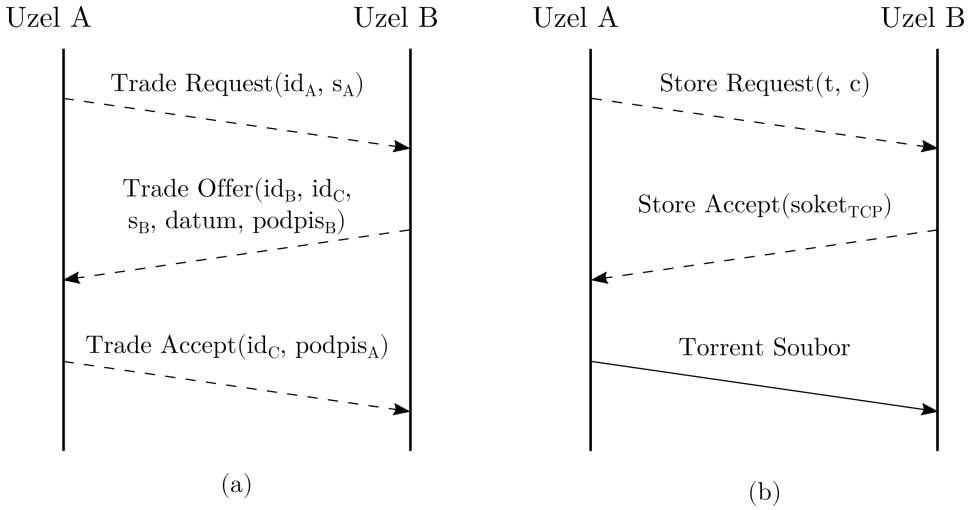
Každý ze zbylých bloků pak rozdělíme do čtyř fragmentů fixní velikosti. K těmto čtyřem fragmentům pak pomocí samoopravných Reed-Solomonových kódů vygenerujeme dva paritní fragmenty. Informaci o původních čtyřech fragmentech bloku (hash, velikost, umístění uvnitř bloku) a o dvou paritních fragmentech (hash, velikost) zapíšeme do příslušného BFL. Všechny fragmenty pak zašifrujeme pomocí konvergentního šifrování.

Připravení zálohovacích sad

Vytvořené fragmenty jsou pak rozděleny do šesti zálohovacích sad, jak je ukázáno na Obrázku 3.4. Tyto zálohovací sady jsou následně rozdistribuovány do partnerských uzlů.

Distribuce zálohovacích sad do partnerských uzlů

Jak bylo zmíněno v Sekci 3.2, uzly v systému BTVault používají optimální politiku obchodování. Celý průběh procesu obchodování je zobrazen na Obrázku 3.5. Uzel, který chce zazálohovat svá data, nejdříve rozešle po celé síti žádost o obchodování (*trading request*). Tato žádost bude obsahovat následující informaci: identifikační číslo žádajícího uzlu a velikost potřebného dis-



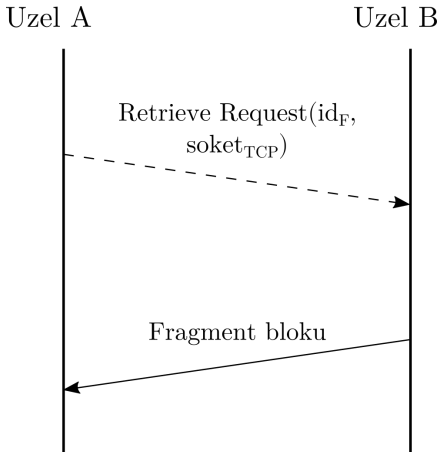
Obrázek 3.5: Příklad komunikace mezi uzly během procesu distribuce dat. Přerušovanými šipkami je vyznačena komunikace prostřednictvím DHT. Obyčejné šipky označují přímé soketové spojení mezi uzly.

kového prostoru. Pokud podobnou žádost obdrží uzel, který má prostředky a chuť obchodovat, odešle žádajícímu uzlu svůj návrh partnerské smlouvy. Tento návrh bude obsahovat následující informaci: identifikační číslo navrhujícího uzlu, identifikační číslo smlouvy, velikost diskového prostoru požadovanou navrhujícím uzlem (určuje se podle vzorce 3.8), dobu platnosti smlouvy a digitální podpis navrhovatele. Jakmile tento návrh obdrží původní žadatel, může rozhodnout, zda tuto nabídku přijme nebo zamítne. Pokud žádající uzel návrh schválí, na základě tohoto návrhu vytvoří novou smlouvu c. Již podepsaná kopie c se odešle zpět navrhovateli. Tento proces se bude opakovat do té doby, dokud zálohující uzel nesežene dostatečné množství partnerů pro uchování všech připravených zálohovacích sad.

Ihned po skončení procesu obchodování následuje proces přenosu uživatelských dat do partnerských uzlu. Pro každou zálohovací sadu $S = [s_1, \ldots, s_6]$ najdeme k (momentálně $k = 5$) smluv $C = [c_1, \ldots, c_k]$ odpovídající velikosti. Samotný přenos sad se uskuteční pomocí BitTorrent protokolu, takže se pro každou sadu vytvoří jeden torrent soubor. Tím pádem je i-tá zálohovací sada odeslána j-tému partnerovi v podobě $\langle \texttt{torrent}(s_i), c_j \rangle$, čímž jsou propojeny všechny fragmenty sady s_i se smlouvou c_j. Každý partner bude zapsán do *holder list* (HL) jako držitel všech jemu svěřených fragmentů.

3.7 Proces obnovení dat

Stejně jako u zálohování se proces obnovení dat spouští pomocí klientské aplikace. K obnovení obsahu konkretního souboru nejdříve potřebujeme do-



Obrázek 3.6: Příklad komunikace mezi uzly během procesu obnovení dat. Přerušovanými šipkami je vyznačena komunikace prostřednictvím DHT. Obyčejné šipky označují přímé soketové spojení mezi uzly.

hledat v DHT meta-soubor příslušný danému souboru. FBL tohoto souboru obsahuje informaci o všech blocích patřících danému souboru. Pomocí FBL jsme schopni zrekonstruovat původní obsah souboru z obsahu jeho bloků. Informace obsažená v FBL nám mimo jiné dovoluje najít BFL potřebných bloků. Z BFL zjistíme veškerou informací o fragmentech bloku. Tato informace nám umožní dešifrovat obsah fragmentů a najít jejich HL, které nám poskytnou informaci o umístění fragmentů v síti.

Pro stažení fragmentu f odešleme uzlům za něj zodpovědným žádost o jeho stažení. Tato žádost bude obsahovat pouze identifikační číslo požadovaného fragmentu. Po obdržení této žádosti uzel vyhledá ve svém souborovém systému požadovaný fragment a pošle jej žádajícímu uzlu. Tento proces je názorně ilustrován na Obrázku 3.6.

Kapitola 4

Realizace BTVaultu

Tato kapitola popisuje implementaci prototypu systému BTVault. V prototypu nejsou implementovány veškeré součásti předchozího návrhu. Nicméně, prototyp je schopen v současném stavu provádět základní operace zálohování a obnovení dat. Prototyp je složen ze tří částí: centrálního serveru (`btv-srv` viz. Příloha B.1), lokálních uzlů (`btv-node` viz. Příloha B.2) a klientské aplikace (`btv-cli` viz. Příloha B.3). Pro nedostatek času a náročnější implementaci do prototypu nejsou zahrnuty tyto funkce:

- Uzly sítě netvoří strukturovanou DHT síť, místo toho veškerou funkcionalitu DHT (předání zpráv a uchovávání metadat) plní centrální server.
- Neuskutečňují se pravidelné partnerské kontroly.
- Neřeší se neočekávaná odpojení uzlů během procesu přenášení dat.

4.1 Jazyk implementace a platforma

Veškerý zdrojový kód prototypu je napsán v jazyce Go[14]. Jazyk Go byl vybrán primárně z následujících důvodů:

- Dobrá podpora souběžnosti procesů na úrovni jazyka, která výrazně zjednodušuje napsání paralelních programů.
- Užitečná standardní knihovna poskytující kryptografické, síťové a systémové nástroje.
- Automatická správa paměti.
- Velké množství hotových open-source projektů/knihoven.

I přestože je jazyk Go multiplatformní, momentálně jsou podporovány pouze UN*Xové operační systémy. Toto omezení je dáno obecnou nekompatibilitou mezi souborovými systémy používanými v různých operačních systémech.

4.2 Server

Implementace serverové části prototypu se nachází uvnitř balíčku `gitlab.fel.cvut.cz/mansumax/btv-srv`[1].

Centrální server plní následující funkce:

- Registraci uživatelů.
- Uložení/stažení metadat na základě klíčů.
- Poskytování informace o reputaci jednotlivých uzlů.
- Odeslání a šíření zpráv uzlů.

Velká část funkcionality serveru je implementována v podobě webových služeb. Vzájemná komunikace mezi uzly a serverem probíhá pomocí protokolu HTTP. Ve výchozím nastavení na příchozí HTTP zprávy server poslouchá na portu 7771. Tabulka 4.1 poskytuje základní přehled všech služeb poskytovaných serverem. Jak je vidět z této tabulky, server vyžaduje digitální podepsání všech příchozích požadavků. Ověření podpisu se provádí pomocí standardního ECDSA[15] protokolu.

Ukládání a stahování metadat se provádí s použitím standardního TCP/IP protokolu. Server čeká na příchozí spojení na portu 7773. Server přijímá zprávy v následujících formátech:

- `"PUT` *`key size`*`\r\n` *`data`*`"` → požadavek na uložení metadat *data* velikosti *size* a klíčem *key*.
- `"GET` *`key`*`\r\n"` → žádost o stažení metadat asociovaných s klíčem *key*.

V případě neúspěchu server odpovídá řetězcem `"KO` *`error`*`\r\n"`, do kterého zahrnuje kód chyby. Bezchybové ukončení interakce se indikuje krátkým řetězcem `"OK\r\n"`.

Informace o uživatelích a metadatech se ukládá do SQLite databáze (přesněji do souboru `btv-srv-db.sqlite`).

4.3 Uzel

Implementaci uzlu lze najít v balíčku `gitlab.fel.cvut.cz/mansumax/btv-node`.

Uzel plní dvě hlavní funkce:

- Příjem a zpracování požadavků uživatele.
- Příjem a zpracování požadavků jiných uzlů sítě.

URI	Metoda	Atribut	Popis
`*/signup`	`POST`	`name`	emailová adresa nového uživatele
		`key`	veřejná část klíče nového uživatele
		`sigR/sigL`	digitální podpis uživatele
`*/confirm`	`POST`	`name`	emailová adresa nového uživatele
		`key`	veřejná část klíče nového uživatele
		`code`	ověřovací kód
		`sigR/sigL`	digitální podpis uživatele
`*/broadcast`	`POST`	`sender`	identifikační číslo odesilatele
		`key`	veřejná část klíče odesílatele
		`msg`	text zprávy
		`sigR/sigL`	digitální podpis odesilatele
`*/send`	`POST`	`sender`	identifikační číslo odesilatele
		`recip`	identifikační číslo příjemce
		`key`	veřejná část klíče odesílatele
		`msg`	text zprávy
		`sigR/sigL`	digitální podpis odesilatele
`*/heartbeat`	`POST`	`id`	identifikační číslo uzlu
		`key`	veřejná část klíče uzlu
		`ip`	IP adresa uzlu
		`port`	číslo portu pro příchozí zprávy
		`sigR/sigL`	digitální podpis uzlu
`*/reputation`	`GET`	`sender`	identifikační číslo odesílatele
		`key`	veřejná část klíče odesílatele
		`user`	identifikační číslo uzlu
		`sigR/sigL`	digitální podpis odesílatele

Tabulka 4.1: Přehled služeb poskytovaných serverem.

Příjem uživatelských požadavků je realizován pomocí RPC protokolu, s využitím prostředků nabízených standardní knihovnou jazyka Go. Ve výchozím nastavení na příchozí RPC volání uzel poslouchá na portu 7774. Obrázek 4.1 poskytuje výpis všech typů a metod vystavených v podobě RPC služeb.

Na požadavky ostatních uzlů sítě uzel poslouchá na náhodně vybraném portu. Výběr portu se uskutečňuje při nastartování uzlu. Jednou vybraný port se používá po celou dobu běhu uzlu. Číslo vybraného portu se posílá na server spolu s IP adresou uzlu v podobě HTTP žádosti (viz. `*/heartbeat` v Tabulce 4.1). Uzel přijímá zprávy v následujících formátech:

- `"TRADE REQUEST` *`uid size`*`\r\n"` → obchodní požadavek od uzlu *uid* o diskový prostor velikosti *size*.
- `"TRADE OFFER` *`uid cid size from to`*`\r\n"` → návrh obchodní smlouvy

[1] v jazyce Go se pro identifikaci balíčků používají URI adresy

```
// Proxy typ.
type NodeRPC struct {
    node *Node // Pointer na samotny uzel.
}

// Backup slouzi k zahajeni zalohovaciho procesu.
func (n *NodeRPC) Backup(req BackupRequest, res *BackupResponse) error

// List poskytuje seznam s informaci o zazalohovanych souborech.
func (n *NodeRPC) List(req ListRequest, res *ListResponse) error

// List slouzi k zahajeni procesu obnoveni zaloh.
func (n *NodeRPC) Restore(req RestoreRequest, res *RestoreResponse) error
```

Obrázek 4.1: Hlavičky metod uzlu vystavených v podobě RPC služeb.

cid od uzlu *uid* s požadavkem diskového prostoru velikosti *size* s dobou platnosti *from-to*.

- `"TRADE ACCEPT` *`cid`*`\r\n"` → notifikace o přijetí návrhu a založení smlouvy *cid*.
- `"STORE REQUEST` *`cid sid size`*`\r\n"` → požadavek na úschovu zálohovací sady *sid* velikosti *size* na základě smlouvy *cid*.
- `"STORE ACCEPT` *`cid sid ip port`*`\r\n"` → přijetí požadavku na úschovu zálohovací sady *sid* na základě smlouvy *cid*. Očekává se zaslání odpovídajícího torrent souboru na adresu *ip:port*.
- `"RETRIEVE REQUEST` *`fid ip port`*`\r\n"` → požadavek na zaslání fragmentu *fid* na adresu *ip:port*.

Neplatné požadavky a zprávy ve špatném formátu jsou ignorovány.

Informace o stavu zálohovacích sad, lokálně uložených datech a partnerech se ukládá do SQLite databáze (přesněji do souboru `\$HOME/.btvault/repo/btv-node-db.sqlite`).

4.3.1 Konfigurační soubor

Pro svojí práci uzel vyžaduje existenci konfiguračního souboru. Konfigurace uzlu je udržována uvnitř YAML souboru `config.yaml` a tabulka 4.2 poskytuje informaci o všech dostupných parametrech. Obsah konfiguračního souboru je načten pouze při spuštění uzlu. Většinu hodnot parametrů z konfiguračního souboru lze překrýt z příkazové řádky pomocí přepínačů.

Parametr	Výchozí hodnota	Popis
`username`	-	uživatelské jméno (momentálně se používá emailová adresa). Používá se pouze pro účely registrace.
`userid`	-	uživatelské unikátní identifikační číslo. Přiřazuje se serverem v procesu registrace. Požívá se zejména při komunikaci s ostatními uzly sítě.
`usercert`	-	uživatelský certifikát. Vydává se serverem v procesu registrace. V prototypu se nepoužívá.
`serverkey`	-	veřejná část klíče serveru. Vydává se serverem v procesu registrace. V prototypu se nepoužívá.
`keydir`	`$HOME/.ssh`	složka s uživatelským klíčem.
`repository`	`$HOME/.btvault/repo`	složka pro uložení dat uzlu.
`server`	`localhost:7771`	kontaktní adresa serveru.
`tracker`	`localhost:7772`	kontaktní adresa BitTorrent trackeru. V prototypu se nepoužívá.
`dht`	`localhost:7773`	kontaktní adresa *bootstrap* uzlu DHT.
`node`	`localhost:7774`	kontaktní lokálního uzlu. Používá se pro komunikaci s lokálně běžící instanci uzlu.

Tabulka 4.2: Obsah konfiguračního souboru `config.yaml`

4.3.2 Klíč

Uzel také potřebuje mít přístup k uživatelskému klíči. Uživatelský klíč se skládá ze dvou části: veřejné `public.btvkey` a soukromé `secret.btvkey`. V prototypu používáme šifry založené na kryptografii eliptických křivek (*elliptic curve cryptography*). Pomocí klientské aplikace si uživatelé mohou vygenerovat klíče z křivek P-256, P-384 a P-512.

4.3.3 BitTorrent klient

Distribuce zálohovacích sad se uskutečňuje pomocí BitTorrent protokolu. Jelikož implementace tohoto protokolu není vůbec triviální záležitost, bylo rozhodnuto pro tyto účely použít externí torrent klient - *transmission-daemon*.

Transmission-daemon je napsán v jazyce C a je součásti open-source projektu Transmission [16]. Hlavní výhodou tohoto klientu je jeho flexibilita a snadná integrovatelnost. Každý uzel při nastartování spouští vlastní instanci tohoto klienta. Číslo portu pro komunikaci s klientem se vybírá zcela

náhodně. Přidání, smazaní a spouštění jednotlivých torrentů pak probíhá prostřednictvím REST API nabízeného transmission-daemonem.

4.4 Klient

Implementace klienta je umístěna v baličku `gitlab.fel.cvut.cz/mansumax/btv-cli`.

Klientská aplikace poskytuje rozhraní pro ovládání lokálně běžících uzlů z příkazové řádky. Spouštění příkazů z příkazové řádky umožňuje snadné plánování akcí třeba pomocí cronu.

Základem je spuštění příkazu `btv-cli`, po kterém vždycky následuje příkaz, který chceme provést. Inspirací pro rozhraní bylo zčásti rozhraní verzovacího systému Git [17].

Pomocí tohoto rozhraní lze provádět všechny potřebné akce:

- `signup` - provést registraci uživatele (viz. Příloha B.3.1).
- `keygen` - vygenerovat/obnovit klíče (viz. Příloha B.3.2).
- `list` - zobrazit seznam zazálohovaných souborů a také sledovat historii jednoho konkretního souboru (viz. Příloha B.3.3).
- `backup` - spustit proces zálohování souborů/adresářů (viz. Příloha B.3.4).
- `restore` - obnovit jeden nebo více souborů do specifikované cesty (viz. Příloha B.3.5).
- `version` - zobrazit základní informaci o klientu (viz. Příloha B.3.6).

4.5 Testování

4.5.1 Testovací prostředí

Testování prototypu probíhalo převážně manuálním způsobem v simulovaném P2P prostředí.

Testování probíhalo zpočátku pouze na localhostu se dvěma až pěti uzly, následně se přešlo na testování na dvou počítačích zapojených do lokální NAT sítě. Výkonnější počítač byl použit pro simulaci funkčního P2P prostředí. Na tomto počítači byla spuštěna jedna instance centrálního serveru a stovka instancí uzlů sítě (shellový skript `demo/setup.sh` napsaný pro tento účel je součásti přiloženého CD). Na druhém počítači běžel pouze jeden uzel a klientská aplikace.

Konfigurace obou počítačů byla následující:

	Počítač č.1	Počítač č.2
CPU	Intel Core i5-3320M @2.6GHz	AMD Athlon II P320 @2.1GHz
RAM	8GB DDR3	2GB DDR3
HDD	500GB	250GB
OS	Debian 8.8 (Jessie) 64bit	Ubuntu 16.04LTS 64bit

Cílem testu bylo zazálohovat celý domovský adresář uživatele a následně jej obnovit. Celá uživatelská složka zabírala na disku ≈ 120MB a její struktura vypadala následovně:

```
/home
└── user
    ├── doc
    │   ├── dokument.odt
    │   ├── dokument.pdf
    │   └── dokument.txt
    ├── pic
    │   ├── cat.gif
    │   └── dog.png
    ├── mus
    │   └── song.mp3
    └── linux-4.11.2.tar.xz
```

4.5.2 Výsledky

Celkově testování proběhlo zdárně pro všechny typy souborů. Obsah všech zazálohovaných souborů byl úspěšně obnoven. Podařilo se zazálohovat uživatelská data u 30 různých partnerů.

Během testování se ovšem objevil největší nedostatek prototypu, který spočíval v rychlosti procesu zálohování. Měření ukázalo, že zálohování celé uživatelské složky zabralo necelých 18 minut, kdežto obnovení všech souborů trvalo pouze 6 minut. Během zálohování nejvíce času zabral právě proces přenosu zálohovacích sad (skoro 16 minut) pomocí BitTorrent protokolu. To bylo způsobeno hlavně tím, že uzly dlouhou dobu nemohly navázat spojení s ostatními peery. Tento problém by se asi dal řešit zavedením vlastního privátního trackeru pro uživatele systému BTVault.

Podařilo se nám tedy otestovat celý systém na reálných datech, bohužel se ukázaly některé implementační nedostatky. Naměřené časy jsou také velice dlouhé a je nutné věnovat určitý čas na optimalizaci všech součástí systému.

Kapitola 5

Závěr

V této práci jsme se pokusili nabídnout další řešení problému zálohování dat. V ideálním světě by toto nebylo nezbytné, nicméně náš svět ideální není a občas v něm dochází ke ztrátám či poškozením dat, což zdůvodňuje potřebnost zálohovacích systémů.

Původním cílem této práce bylo seznámení čtenáře se světem P2P distribuovaného zálohování. Na začátku práce jsme vypracovali rešerši již existujících systémů pro distribuované zálohování dat. Následně byly poznatky z této rešerše použity při analýze a návrhu vlastního systému pro P2P zálohování. Následovala realizace prototypu tohoto systému, který byl poté otestován.

Výsledkem práce je prototyp, který je schopen provádět inkrementální zálohování a obnovení uživatelských dat. Častým problémem zálohování je neochota uživatelů strávit čas s jeho nastavováním, nebo jeho složité použití v případě ztráty dat. Proto jsme se pokusili při vývoji BTVaultu vycházet z principu držet pro uživatele vše co možná nejpřímočařejší. Bohužel z důvodu nedostatku času se nepovedlo naimplementovat veškeré funkce zmíněné v návrhu.

5.1 Budoucí práce

Navržený zálohovací systém není zdaleka ideální a je potřeba věnovat hodně času na vylepšení samotného návrhu i na důkladnější implementaci. Další vývoj systému by se mohl ubírat následujícími směry:

1. Zapojení uzlu do skutečné DHT sítě. Bohužel v době vývoje nebyla k dispozici žádná vhodná DHT knihovna. Proto bychom pro budoucí rozvoj doporučovali vytvoření vlastní DHT knihovny, což je však samo o sobě velmi komplikovaná úloha.

2. Kompletní decentralizace sítě. Přidání nových uživatelů do sítě lze řešit decentralizovaně například pomocí systému pozvánek (*invitation systém*). Existují dokonce způsoby pro udržení reputačního systému v

plně decentralizovaných P2P sítích[18].

3. Nativní podpora BitTorrent protokolu.
4. Použití privátního BitTorrent trackeru pro uživatele BTVaultu.
5. Vylepšení bezpečnostního modelu. V aktuálním modelu totiž chybí podpora snadné výměny uživatelských klíčů.

5.2 Osobní názor autora

Celkově mi tato práce hodně dala a vzala. Prohloubil jsem si svoje znalosti programovacího jazyka Go. Předtím jsem jazyk Go používal výhradně pro psání malých personálních programů a skriptů. Čtením odborných článků a technických zpráv jsem si jednoznačně vylepšil svoji znalost angličtiny. Práce na tomto projektu mi poskytla skvělou příležitost aplikovat mnoho znalostí z operačních systému, distribuovaných systémů a kryptografie. Nakonec jsem si vlastní zkušeností ověřil, že návrh komplexních systémů vyžaduje velké mentální úsilí a hodně trpělivosti.

Bibliografie

[1] C. Batten et al. *pStore: A Secure Peer-to-Peer Backup System*. Technical Memo MIT-LCS-TM-632. Massachusetts Institute of Technology Laboratory for Computer Science, 2002.

[2] I. Stoica et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *SIGCOMM Comput. Commun. Rev.* 31.4 (srp. 2001), s. 149–160. ISSN: 0146-4833. DOI: 10.1145/964723.383071. URL: http://doi.acm.org/10.1145/964723.383071.

[3] J. R. Douceur et al. "Reclaiming space from duplicate files in a serverless distributed file system". In: *Proceedings 22nd International Conference on Distributed Computing Systems*. 2002, s. 617–624. DOI: 10.1109/ICDCS.2002.1022312.

[4] M. Lillibridge et al. "A Cooperative Internet Backup Scheme". In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '03. San Antonio, Texas: USENIX Association, 2003, s. 3–3. URL: http://dl.acm.org/citation.cfm?id=1247340.1247343.

[5] L. P. Cox, C. D. Murray a B. D. Noble. "Pastiche: Making Backup Cheap and Easy". In: *Proceedings of the 5th Symposium on Operating Systems Design and implementation*. OSDI '02. Boston, Massachusetts: USENIX Association, 2002, s. 285–298. ISBN: 978-1-4503-0111-4. URL: http://dl.acm.org/citation.cfm?id=1060289.1060316.

[6] Resilio Inc. *Reslio Web Page*. 2017. URL: https://www.resilio.com/ (cit. 01. 03. 2017).

[7] C. Blake a R. Rodrigues. "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two". In: *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*. HOTOS'03. Lihue, Hawaii: USENIX Association, 2003, s. 1–1. URL: http://dl.acm.org/citation.cfm?id=1251054.1251055.

[8] A. Muthitacharoen, B. Chen a D. Mazières. "A Low-bandwidth Network File System". In: *SIGOPS Oper. Syst. Rev.* 35.5 (říj. 2001), s. 174–187. ISSN: 0163-5980. DOI: 10.1145/502059.502052. URL: http://doi.acm.org/10.1145/502059.502052.

[9] M. O. Rabin. *Fingerprinting by Random Polynomials*. Tech. zpr. TR-15-81. Center for Research in Computing Technology, Harvard University, 1981.

[10] M. J. Dworkin. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. zpr. Gaithersburg, MD, United States, 2007.

[11] H. Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *Proceedings of the 30th Annual Conference on Advances in Cryptology*. CRYPTO'10. Santa Barbara, CA, USA: Springer-Verlag, 2010, s. 631–648. ISBN: 3-642-14622-8, 978-3-642-14622-0. URL: http://dl.acm.org/citation.cfm?id=1881412.1881456.

[12] M. J. Dworkin. *SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. Tech. zpr. Gaithersburg, MD, United States, 2001.

[13] I. S. Reed a G. Solomon. "Polynomial codes over certain finite fields". In: *Journal of the Society of Industrial and Applied Mathematics* 8.2 (1960), s. 300–304. URL: http://www.jstor.org/pss/2098968.

[14] Google Inc. *The Go Programming Language Specification*. 2017. URL: https://www.golang.org/ref/spec (cit. 01. 04. 2017).

[15] T. Pornin. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. RFC 6979. 2013, s. 1–79. URL: http://www.rfc-editor.org/rfc/rfc6979.txt.

[16] E. Petit, J. Elsasser a B. Varner. *Transmission Project Web Page*. 2017. URL: https://transmissionbt.com/ (cit. 01. 04. 2017).

[17] S. Chacon. *Pro Git*. 1st. Berkely, CA, USA: Apress, 2009. ISBN: 1430218339, 9781430218333.

[18] C. Aperjis a R. Johari. "A Peer-to-peer System As an Exchange Economy". In: *Proceeding from the 2006 Workshop on Game Theory for Communications and Networks*. GameNets '06. Pisa, Italy: ACM, 2006. ISBN: 1-59593-507-X. DOI: 10.1145/1190195.1190206. URL: http://doi.acm.org/10.1145/1190195.1190206.

Příloha A

Instalační a uživatelská příručka

A.1 Prostředí pro běh

Systém byl vyvíjen a testován na operačním systému Ubuntu 16.04LTS 64bit a testován na systému Debian 8.8 (Jessie) 64bit.

Před použitím systému je potřeba doinstalovat následující balíčky:

- Programovací jazyk Go verze alespoň 1.8 (`golang-1.8`).
- Knihovnu pro práci s databázi SQLite (`libsqlite3`, `libsqlite3-dev`).
- BitTorrent klient Transmission verze alespoň 2.5 (`transmission-common`, `transmission-daemon`).

A.2 Instalace a kompilace

Instalovat celý programu lze pomocí příkazu `go get`, a to dvěma způsoby:

1. Instalovat nejaktuálnější verzi přímo z gitového repositáře:

   ```
   $ go get -u gitlab.fel.cvut.cz/mansumax/btvault/...
   ```

2. Instalovat přímo ze zdrojových kódů na CD:

   ```
   $ cp -r /media/cdrom/gitlab.fel.cvut.cz $GOPATH/src
   $ go get gitlab.fel.cvut.cz/mansumax/btvault/...
   ```

Příkaz `go get` automaticky stáhne a nainstaluje všechný potřebné závislosti, pak celý projekt zkompiluje a umístí výsledné spustitelné binární soubory (`btv-srv`, `btv-node`, `btv-cli`) do složky `$GOPATH/bin`.

A.3 Spuštění

Spuštění celého systému je potřeba provádět ve správném pořadí. Nejdříve je potřeba nastartovat server a pak přidat do sítě několik uzlu. Pro tyto účely bude nejjednodušší použit speciální shellový skript `demo/setup.sh`. Pak lze vytvořit P2P síť se serverem a deseti uzly příkazy:

```
$ cd $GOPATH/src/gitlab.fel.cvut.cz/mansumax/btvault/demo
$ ./setup.sh 10
```

Po úspěšném nastartování sítě by se měly zobrazit následující řádky:

```
installing necessary programs ... done
creating temporary directory ... done [/tmp/btvault-20170521010656]
creating log directory ...done [/tmp/btvault-20170521010656/logs]
starting server ... done [8596]
added user: user000
starting node ... done [8611]
added user: user001
starting node ... done [8620]
added user: user002
starting node ... done [8630]
added user: user003
starting node ... done [8639]
added user: user004
starting node ... done [8660]
added user: user005
starting node ... done [8672]
added user: user006
starting node ... done [8684]
added user: user007
starting node ... done [8698]
added user: user008
starting node ... done [8722]
added user: user009
starting node ... done [8735]

press ENTER to terminate the session
```

Zastavit běh celé sítě lze stisknutím klávesy `ENTER`.

Příloha B

Návody k použití

B.1 btv-srv

```
NAME
        btv-srv - A central server of the BTVault system.

SYNOPSIS
        btv-srv [options]

DESCRIPTION
        This application represents a central server of the BTVault system.

OPTIONS
        --config="$HOME/.btvault/config.yaml"
                configuration file to use

        --dhtport="7773"
                port number for DHT to listen for incoming requests

        -h, --help
                display the help message

        --keydir="$HOME/.ssh"
                directory where the keys are stored

        --port="7771"
                port number for server to listen for incoming requests

        --srvdir="$HOME/.btvault/srv"
                main server directory

        -v, --verbose[=false]
                verbose mode
```

B.2 btv-node

```
NAME
        btv-node - A node of the BTVault system.

SYNOPSIS
        btv-node [options]

DESCRIPTION
        This application represents a local node that participates in the
    P2P network (e.g. provides backup services). This node is usually
    launched as a daemon process, to manage your local node use btv-cli.

OPTIONS
        --config="$HOME/.btvault/config.yaml"
                configuration file to use

        -h, --help
                display the help message

        -v, --verbose[=false]
                verbose mode
```

B.3 btv-cli

```
NAME
        btv-cli - A remote control command-line utility for btv-node.

SYNOPSIS
        btv-cli [command] [options]

DESCRIPTION
        This application allows user to join the P2P network (e.g. signup)
    and to manage the user's locally running nodes.

COMMANDS
        backup
                create a new backup of files and/or directories

        help
                help about any command

        keygen
                create a new key pair for the user

        list
                show the backed up files and their history.

        restore
                restore backups

        signup
                register a new user in the BTVault network
```

```
        version
                show version

OPTIONS
        --config="$HOME/.btvault/config.yaml"
                configuration file to use

        -h, --help
                display the help message
```

B.3.1 signup

```
NAME
        btv-cli-signup - Register a new user in the BTVault network

SYNOPSIS
        btv-cli signup [options] [<username>]

DESCRIPTION
        The "signup" command generates an BTVault configuration file and
    private/public key pair, stores them locally, and sends a signup
    request to the public BTVault server. Username should be a valid email
    address in format <user name>@<domain name>.

OPTIONS
        --curve="p256"
                cryptographic curve name: p256, p384 or p521

        --dht="127.0.0.1:7773"
                dht's endpoint in form <host>:<port>

        -f, --force[=false]
                create a new user even if keys or config file already exist

        --keydir="$HOME/.ssh"
                directory to store keys

        --node="127.0.0.1:7774"
                node's endpoint in form <host>:<port>

        --repo="$HOME/.btvault/repo"
                repository directory

        --server="127.0.0.1:7771"
                server's endpoint in form <host>:<port>

        --tracker="127.0.0.1:7772"
                tracker's endpoint in form <host>:<port>
```

B.3.2 keygen

```
NAME
        btv-cli-keygen - Create a new key pair for the user

SYNOPSIS
        btv-cli keygen [options]

DESCRIPTION
        The "keygen" command creates a new BTVault key pair and by default
    stores the pair in local files secret.btvkey and public.btvkey in $HOME
    /.ssh.  New users should instead use the signup command to create their
     first key. Keygen is usually used to restore keys from secret seeds.
    Secret seeds must be in proquint format.

OPTIONS
        --curve="p256"
                cryptographic curve name: p256, p384 or p521

        -f, --force[=false]
                generate new keys even if older keys still exist

        --keydir="$HOME/.ssh"
                directory to store keys

        --secretseed=""
                the seed containing a 128 bit secret in proquint format or a
     file that contains it
```

B.3.3 list

```
NAME
        btv-cli-list - Show the backed up files and their history

SYNOPSIS
        btv-cli list [<id>]

DESCRIPTION
        The "list" command without any arguments shows a list of the backed
    up files. If an ID of a specific file is provided, then the history
    that file will be displayed. This command is designed to be used in
    conjunction with "restore" command.

EXAMPLES:
        Print all available versions of the file with the given ID:

        btv-cli list 256BiTsLoNgFiLeHaShInBaSe64FoRmAtBlAbLaBlA

        Restore all files from the directory '/home/user/doc':

        btv-cli list | awk '/home\/user\/doc/{print $1}' | btv-cli restore
    --stdin
```

```
       Write identifiers of all PDF files into a plain text file:

       btv-cli list | awk '/\.pdf$/{print $1}' > restore.txt
```

B.3.4 backup

```
NAME
       btv-cli-backup - Create a new backup of files and/or directories

SYNOPSIS
       btv-cli backup [options] [<path>...]

DESCRIPTION
       The "backup" command starts the backup process and saves the files
    and directories given as the arguments.

OPTIONS
       -e, --exclude=[]
              exclude a pattern (can be specified multiple times)

       --exclude-file=""
              read exclude patterns from a file

       --files-from=""
              read the files to backup from file (can be combined with
    file args)
```

B.3.5 restore

```
NAME
       btv-cli-restore - Restore backups

SYNOPSIS
       btv-cli restore [options] [<id>...]

DESCRIPTION
       The "restore" command restores backups of the files. IDs should be
    in the format <file id>[#<version timestamp>].

OPTIONS
       --ids-from=""
              read IDs of the files to restore from file (cannot be
    combined with --stdin)

       -o, --output="$PWD"
              output directory

       --stdin[=false]
              read IDs of the files to restore from standard input
```

B.3.6 version

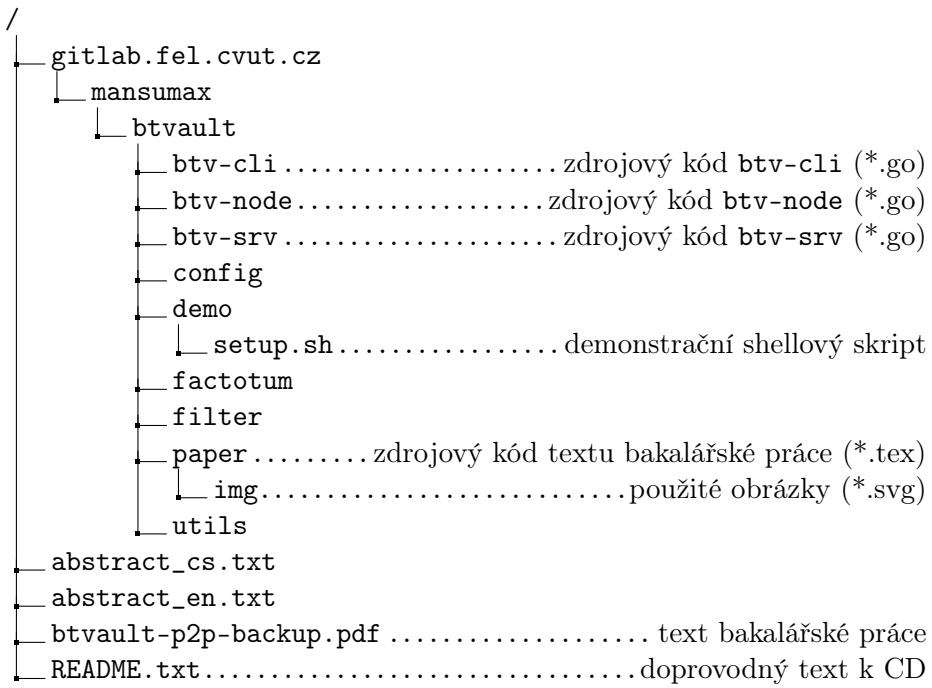
```
NAME
        btv-cli-version - Show version

SYNOPSIS
        btv-cli version

DESCRIPTION
        The "version" command shows a BTVault's logo and the version of the
    client.
```

Příloha C

Obsah přiloženého CD



České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Maxat M a n s u r o v

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Peer-to-peer zálohovací systém založený na BitTorrent protokolu

Pokyny pro vypracování:

Seznamte se s problematikou zálohování pomocí peer-to-peer (P2P) systémů pro sdílení dat. Proveďte rešerši existujících komerčních a open-source P2P zálohovacích systémů. Typický scénář zálohování je takovýto: každý uživatel zazálohuje svá data, zašifruje je a případně je digitálně podepíše a publikuje je do P2P sítě. Současně nabídne určitý diskový prostor ostatním účastníkům pro uložení jejich dat. Následně si uživatelé automaticky přepošlou data, aby jejich data byla rovnoměrně rozložena v síti a mohla být stažena jejich vlastníkem v případě ztráty dat.
Zaměřte se na strategie používané na distribuci dat do jednotlivých uzlů P2P sítě. Vyberte několik strategií distribuce dat a porovnejte je z různých hledisek (nároky na místo na úložišti/redundance/čas potřebný ke znovuvybavení zazálohovaných dat...). Navrhněte architekturu aplikace pro P2P distribuované zálohování s využitím protokolu BitTorrent. Naimplementujte uživatelsky přívětivého klienta a server postavený nad BitTorrent trackerem, který bude zařizovat distribuci dat mezi klienty, určovat kredit klientů (tj. poměr mezi velikostí úložiště určeného pro data ostatních klientů a místem určeným pro vlastní data) atd.

Seznam odborné literatury:

[1] Landon P. Cox, Christopher D. Murray, and Brian D. Noble: Pastiche: Making Backup Cheap and Easy. Proceedings of the 5th Symposium on Operating Systems Design and Implementation , Boston, Massachusetts, USA , December 9-11, 2002
[2] C. Batten, K. Barr, Arvind Saraf and Stanley Trepetin: pStore: A Secure Peer-to-Peer Backup System. Technical Report, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002

Vedoucí bakalářské práce: Ing. Miroslav Uller

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 17. 12. 2015