# TECHNICKÁ UNIVERZITA V LIBERCI

**Fakulta mechatroniky, informatiky a mezioborových studií**

**ústav nových technologií**
**a aplikované informatiky**


**Diplomová práce**

# Efektivní hashování spamů


**Bc. Pavel Hozák**

Vedoucí práce: Mgr. Jiří Vraný

Konzultant: Martin Doleček



Studijní program: Elektrotechnika a informatika, navazující magisterský

Obor: Informační technologie

květen 2009

Tady bude oficiální zadání.

## Poděkování

Chtěl bych poděkovat vedoucímu diplomové práce Mgr. Jiřímu Vranému za cenné rady při tvorbě tohoto dokumentu a Martinovi Dolečkovi za rady při tvorbě aplikace.

Největší poděkovaní patří mým rodičům za podporu během mého studia.

# Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu a na základě konzultací s vedoucím diplomové práce a konzultantem.

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **souhlasím** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

V Novém Boru dne …….. …….……

# Abstrakt

Hlavní náplní práce je analýza spamu a návrh a implementace aplikace v jazyce C, která bude detekovat spam za pomoci hashovacích funkcí. Na základě provedené analýzy je navržena normalizace emailových zpráv, tak aby spamy se stejným informačním sdělením měly po zhasování stejný otisk. Normalizace je provedena vynecháním a náhradou některých znaků z emailu. Z takto normalizovaného emailu se pomocí hashovací funkce vypočítá jeho otisk. Tento otisk se poté porovná s databází otisků známých spamů a zjistí se na kolik procent je tento email spam. Výsledná aplikace byla testována na zkušebním vzorku spamu. Podle výsledků odhalila o padesát dva procent více duplicit než aplikace, která vynechávala pouze hlavičky.



# Abstract

The main subject matter of this thesis is a spam analysis and proposal and implementation of application in language C, which will detect spam with the help of hash function. Standardization of email messages is designed on the basis of analysis made in order that spams with the same information notification have the same hash after hashing. The standardization is made by leaving out and replacing some of email signs. The hash is counted with the aid of hashing function from the standardized email. This hash is compared with hash database of well-known spam and it is determined with what probability in percentage is this email spam. The application was tested on a test sample of spam. According to the results revealed the fifty-two percent more duplication than the applications, which only skiped headers.

**SEZNAM OBRÁZKŮ** ..... **8**

**SEZNAM TABULEK** ..... **9**

**1 ÚVOD** ..... **10**

**2 CÍLE DIPLOMOVÉ PRÁCE** ..... **11**
2.3.1 Výběr vhodného obsahu zprávy ..... 11
2.3.2 Výběr vhodné hashovací funkce ..... 11
2.3.3 Ukládání otisků ..... 11

**3 ANALÝZA** ..... **12**

**3.1 Spam** ..... **12**
3.1.1 Filtrování spamu ..... 12
3.1.2 Analýza technik odlišení spamu ..... 16

**3.2 Hash – Hashovací funkce** ..... **19**
3.2.1 Odolnost proti nalezení kolizí ..... 19
3.2.2 Konstrukce hashovací funkce ..... 20
3.2.3 Některé hashovací funkce ..... 21
3.2.4 Využití hashovacích funkcí ..... 23

**3.3 Ukládání otisků** ..... **24**
3.3.1 Způsoby vyhledávání ..... 24
3.3.2 Způsoby uložení ..... 26

**4 NÁVRH ŘEŠENÍ** ..... **27**

**4.1 Návrh výběru z emailu** ..... **27**
4.1.1 Vynechání ..... 27
4.1.2 Náhrada ..... 28
4.1.3 Další postupy ..... 29
4.1.4 Neeliminované techniky ..... 29

**4.2 Výběr hashovací funkce** ..... **30**

**4.3 Výběr databáze** ..... **31**
4.3.1 Volba enginu tabulek ..... 31
4.3.2 Návrh tabulek ..... 32

**5. REALIZACE** ..... **33**

**5.1 Datové struktury** ..... **33**
5.1.1 FRONTA ..... 33
5.1.2 BOUNDER ..... 34
5.1.3 MD5_data ..... 35

**5.2 Hashovací funkce MD5 ..... 36**
5.2.1 Makra ..... 36
5.2.2 Funkce ..... 37

**5.3 Kódovaní znaků s hodnotou větší než 127 ..... 38**

**5.4 Databáze otisků spamu ..... 38**
5.4.1 Vložení otisku ..... 39
5.4.2 Vyhledání otisku ..... 39
5.4.3 Zvýšení počtu špatného označení otisku ..... 39

**5.5 Výběr znaků pro otisk ..... 40**
5.5.1 Makra použitá při výběru ..... 41
5.5.2 Funkce využívané výběrem ..... 42

**5.6 Kompilace ..... 42**

**5.7 Rozšíření aplikace ..... 43**

**6 TESTOVÁNÍ APLIKACE ..... 44**

**7 ZÁVĚR ..... 47**

**ZDROJE INFORMACÍ ..... 48**

**UKÁZKY SPAMU ..... 50**

# Seznam obrázků


- OBRÁZEK 1: E-R DIAGRAM ..... 32
- OBRÁZEK 2: ZAPOJENÍ BUFFERŮ DO KRUHOVÉ FRONTY ..... 34
- OBRÁZEK 3: ZAPOJENÍ HRANIC DO ZÁSOBNÍKU ..... 35

# Seznam tabulek


- TABULKA 1: PÍSMENA ODPOVÍDAJÍCÍ ČÍSLICÍM ..... 28
- TABULKA 2: DEFINOVANÁ MAKRA ..... 41
- TABULKA 3: POČTY NALEZENÝCH DUPLICIT ..... 44
- TABULKA 4: TESTOVACÍ SESTAVY ..... 45
- TABULKA 5: DOBA BĚHU APLIKACE ..... 46

# 1 Úvod

V dnešní době je podíl spamu na emailové komunikaci většinový. Přičemž velký objem tohoto spamu je stejný, stejné sdělení se pošle mnoha uživatelům ve stejných doménách. Freemail od seznamu má k dnešnímu dni 6,3 miliónu aktivních schránek [1]. Dá se předpokládat, že v takovémto množství schránek bude velmi vysoký počet opakujícího se spamu. Pro jeho detekci by stačilo příchozí mail porovnat s již známým spamem, zda jsou shodné. Toto porovnání, kdyby se dělalo byte po bytu, by bylo dost pomalé a hlavně by kladlo velké paměťové nároky na uchovávání již známých spamů. V praxi se pro porovnání dvou souborů používají hashovací funkce, pro každý jakkoli velký soubor vytvoří krátký otisk konstantní délky (128b, 160b, 192b …). Porovnání těchto otisků je daleko rychlejší a paměťové nároky na uchování otisků jsou podstatně menší.

Tato práce se zabývá využitím otisků emailů pro detekci spamu.

V kapitole 3 je popsána problematika detekce spamů, analýza technik pro odlišení stejných spamů při zachování jejich informačního obsahu. V další části této kapitoly jsou popsané principy hashovacích funkcí, některé funkce detailněji, a je zmíněno kde se hashovací funkce využívají. V poslední části kapitoly jsou popsané techniky pro vyhledávání dat.

Kapitola 4 se zabývá návrhem normalizace emailů, výběrem hashovací funkce a výběrem databáze pro ukládání otisků.

V kapitole 5 je popsána implementace aplikace v jazyce ANSI C.

# 2 Cíle diplomové práce

Cílem práce je vytvořit aplikaci v jazyce ANSI C, která bude vytvářet z emailů charakteristické otisky a poté je vyhledávat v databázi otisků již nalezených spamů. Tato aplikace by měla sloužit pro rychlé rozhodnutí, zda zpráva je již známý spam. Jelikož se jeden spam posílá na velmi mnoho adres ve stejných doménách, ušetřil by se výpočetní čas serveru přijímajícího emaily, protože by známé spamy nemusel dále rozpoznávat složitými a časově náročnými technikami pro detekci spamu.

### 2.3.1 Výběr vhodného obsahu zprávy

Výběr vhodného obsahu emailu pro hashovaní, je nejdůležitější část práce, neboť výkon celé aplikace bude stát na tomto výběru. Při špatně zvoleném obsahu, který nezaručí stejný otisk pro dvě zprávy, které jsou informačním obsahem identické, se bude rychle rozrůstat velikost databáze a tím se zpomalí vyhledávání konkrétního otisku. Při výběru, který vytváří stejné otisky i pro dvě velmi rozdílné zprávy, sice bude velikost databáze menší a rychlost vyhledávání v ní bude větší, ale bylo by velké riziko chybného označení emailu za spam, což je velký problém.

### 2.3.2 Výběr vhodné hashovací funkce

Na výběr je mnoho hashovacích funkcí. Pro účel této práce je nejdůležitější rychlost této funkce. Je důležité si uvědomit, že i volba velikosti otisku bude mít vliv na výslednou rychlost celé aplikace. Odolnost proti nalezení kolizí není příliš důležitá.

### 2.3.3 Ukládání otisků

Při návrhu, jak ukládat otisky je nutné zejména dbát na rychlost vyhledávání otisku ve velkém množství záznamů.

# 3 Analýza

## *3.1 Spam*

Jedná se o nevyžádané, masově šířené sdělení, nejčastěji reklamního rázu. V definici jsou důležitá obě slova, neboť nevyžádaný email nutně nemusí být nežádoucí a masově šířené jsou například i některé emailové konference. Podle finančních novin [2] se spam na emailové komunikaci v dubnu 2009 podílel 85 procenty. Toto číslo je enormní. Bohužel spam tu s námi bude do té doby, dokud se vyplatí jeho rozesílání. Podle magazínu securityworld [3] se rozesílání spamu vyplatí ve chvíli, kdy bude získána odpověď na každý dvacetimiliontý email. V roce 2007 byla získána odpověď na každý dvoumiliontý email.

### 3.1.1 Filtrování spamu

Vývoj technik jak se bránit spamu, byl, je a bude složitý, neboť obrana proti spamu je vždy jen reakcí na nějakou novou techniku, kterou si spameři vymysleli. Je důležité si uvědomit, že za tvorbou spamu není jen jeho zadavatel, ale hlavně člověk, který tento spam rozešle a upraví tak, aby se rozeslal co nejvíce lidem a vyhnul se automatickým detekcím. Tento člověk je většinou velmi znalý v oboru a není žádný „blbec“. Při tvorbě detektorů spamu je nutné vžít se do pozice tohoto člověka a snažit se přijít na nový způsob, jak ošálit detektory. Nestačí zaměřit se pouze na momentálně známé techniky, ale je nutné být takzvaně o krok vpřed před těmito lidmi. Bohužel čím více sofistikovaný detektor vymyslíme, tím více výpočetního času nám zabere určení, zdali je zpráva spam.

Nejen z důvodu ušetření výpočetního času existují různě složité filtry pro detekci. Prvotní důvod byl čistě historický, v počátkách internetu stačili velmi jednoduché techniky, jak se spamu zbavit, bohužel s rozvojem internetu se dostavil i rozvoj tvorby spamu a tím i rozvoj jeho detektorů. O níže popsaných technikách se lze více dozvědět v diplomové práci Jaroslava Kortuse [22].

**Blacklisting**

Využívá se databáze známých rozesílatelů spamu a pošta od nich se vůbec nepřijímá. Tyto listy mohou obsahovat IP adresy, emailové adresy. V dnešní době převažuje podíl spamu a tak je čím dál pracnější tyto seznamy udržovat.

**Whitelisting**

Přijímá emaily pouze z povolených adres. Díky tomu se spam k nám víceméně nedostane. Tyto seznamy na rozdíl od blacklistu nebývají, tak rozsáhlé. Problém nastane, když se pošle důležitý email z adresy, která ještě není povolená.

**Greylisting**

Tato technika využívá toho, že spamer chce svoji poštu odeslat co nejrychleji a pokud dojde k chybě při jejím odesílání, tak se nepokouší zprávu znovu odeslat. Server si při příjmu pošty zkontroluje trojici odesílatel-příjemce-počítač, pokud tuto trojici zná, tak email přijme, pokud ne, tak si ji poznamená a email nepřijme. Při novém pokusu o odeslání tuto trojici již zná a email pustí. Při použití této techniky dochází ke zpoždění v doručování zpráv.

**Klíčová slova**

Na základě klíčových slov, která se vyskytují ve spamu (typicky viagra, penis), můžeme spam rozpoznat. Problémem této techniky je, že se spameři tyto slova snaží narušit, tak aby byla různá např. v_i_a_g_r_a, vviagra, via gra atd. Dalším problémem je, že se tato slova mohou vyskytnout i v regulérních emailech.

**Regulární výrazy**

Pomocí regulárních výrazů se zdokonalí předchozí technika, protože se muže snadněji popsat variabilita slova např. v+i+a+g+r+a+ postihne slova jako viagra, vviagra, viaagra atd.

**Hashcash**

Tato metoda se inspirovala v klasické poště, kde se za dopis musí zaplatit. Tato platba je prokázána dopisní známkou. V tomto případě se neplatí penězi, ale odesílatel „zaplatí“ za odeslání emailu výpočetním časem. V této metodě se používá dosažení určitého počtu nul na začátku otisku SHA-1. Tento otisk se vytváří z dat zaslaných serverem, obvykle adresa příjemce, datum a nějaký náhodný řetězec. Odesílatel musí doplnit do emailu hlavičku X-Hashcash, která bude obsahovat řetězec ze serveru a řetězec, kterým hlavičku doplní, tak aby otisk měl na začátku požadovaný počet nul. Spameři chtějí emaily rozeslat co nejrychleji, takže servery používající tuto techniku nepoužívají. Pokud se rozhodnou akceptovat „platbu“, tak se výrazně sníží objem spamu, který jsou schopni odeslat za určitý čas. Domovská stránka projektu [16].

**URL seznamy**

Z emailů označených, jako spam se vyberou URL adresy, obvykle jen část obsahující doménu, a uloží se do databáze. U nově příchozích emailů se poté hledají tyto adresy. Systém využívající tuto techniku se jmenuje SURBL (Spam URI Realtime Blocklists). Domovská stránka projektu SURBL [17].

**DomainKeys**

Tato technika umožňuje ověření, jestli email přišel z uvedeného serveru. Využívá se asymetrického šifrování a podpisu emailu serverem. Server, který email odešle, nejprve vytvoří otisk zprávy a ten posléze zašifruje svým privátním klíčem. Informace přidá do hlavičky DomainKey-Signature. Příjemce si z DNS serveru odesílatele zjistí veřejný klíč, dešifruje otisk v hlavičce, spočítá otisk příchozího emailu a otisky porovná. Tím si ověří, zdali email přišel z daného serveru.

**SPF - Sender Policy Framework**

Tato technika nám umožňuje určit, zdali odesílatel emailu opravdu pochází z domény, kterou uvádí. Pro tuto kontrolu se používá DNS záznamů, ze kterých se zjistí, z jakých IP adres se může posílat email z dané domény.

**Statistické filtry**

Tyto filtry používají k detekci spamu statistiku výskytu slov, většinou se jedná o Bayesovské filtry. Jsou založeny na tom, že mají k dispozici dvě množiny zpráv, jednu pro spam a druhou pro normální zprávy (ham). Z těchto množin se vytvoří tabulky pravděpodobností výskytu N slov jdoucích po sobě. Množina spamu bude mít jiné hodnoty pravděpodobností pro výskyt těchto posloupností než ham. Příchozímu emailu se na základě posloupnosti slov, v něm obsažených, spočítají podle tabulek pravděpodobnosti pro spam a pro ham. Vyšší pravděpodobnost určuje, o jaký typ emailu se jedná.

**Nilsimsa**

Tato technika počítá výskyt trojic znaků v emailu. Trojice se vytvářejí z kombinací pěti po sobě jdoucích znaků. Počty výskytů jednotlivých trojic se poté porovnají s předem zjištěným průměrným výskytem. Po porovnání bude k dispozici 256b otisk, který indikuje rozdílnost v počtu výskytů oproti průměru. Tento otisk se následně postupně porovná s otisky v databázi spamu a zjišťuje se, v kolika bitech se liší. Pokud se během porovnávání dosáhne určité hodnoty je zpráva označena jako spam. Více o projektu Nilsimsa [18].

**Razor a Pyzor**

Jedná se o aplikace používající k detekci spamu jeho otisk funkcí SHA-1. Otisky jsou udržované ve veřejných databázích, do kterých mohou přispívat různí uživatelé. Domovské stránky projektů Pyzor [19] a Razor[20].

**DCC – Distributed Checksum Clearinghouse**

Z příchozího emailu se spočte kontrolní součet a porovná se s veřejnou databází. Pokud je v databázi zjištěno, že tento otisk odpovídá emailu, který se rozeslal velkému počtu uživatelů, jedná se pravděpodobně o spam. Domovská stránka projektu [21].

### 3.1.2 Analýza technik odlišení spamu

Spameři obvykle jednu zprávu rozesílají na mnoho adres, aby zabránili jednoduché detekci spamovým filtrem, snaží se zprávy od sebe odlišit tak, aby cílovému uživateli byly srozumitelné a měly stejné sdělení, ale aby měly různé otisky. Používají proto různé techniky. Níže popsané techniky jsem zjistil z poskytnutého vzorku 4111 spamů a článků ze serveru pooh [10]. Některé ukázky spamu jsou v příloze.

**Různý počet prázdných znaků**

Tato technika využívá neviditelnosti bílých znaků pro uživatele (nenesou důležité sdělení, pouze oddělují slova, odstavce …), ale záměna jediného znaku již změní výsledný otisk zprávy. U zpráv se použije různý počet mezer, někde jedna jinde dvě případně se tabulátor nahradí mezerami nebo obráceně, různý počet „entrů“.

**Přidání náhodných znaků**

Do zprávy se přidají různé znaky, tvořící smajlíky nebo se přidají jen jako výplň.
V menším počtu za sebou se objevují v obsahu emailu, v případě delších pasáží se tyto znaky přidávají na konec sdělení, někdy se na konec zprávy místo náhodných znaků přidávají různé úryvky článků z novin nebo internetových stránek, toto cílené přidání má za úkol zmást statistické filtry spamu.

**Záměna slov synonymy**

Mnoho slov má synonyma, slova která jsou různá, ale mají stejný význam. Toho spameři využívají, neboť po náhradě má zpráva stejný význam, ale její otisk je už různý.

**Optické dělení slov**

Tato technika využívá vkládání stejného znaku mezi jednotlivá písmena slova. Jako prokládací znak se obvykle používá vykřičník, podtržítko, tečky, mezery atd., uživatel tyto prokládací znaky celkem snadno ignoruje a sdělení zprávy pochopí. Typický příklad je slovo viagra, v!i!a!g!r!a, nebo v_i_a_g_r_a.

**Změna velikosti písmen**

Záměnou velikosti písmen se neztratí informační obsah zprávy, ale změní se jejich otisk, protože velká a malá písmena jsou různě zakódována.

**Záměna písmen číslicemi**

Některé číslice vypadají stejně (l-1), nebo velmi podobně (i-1), jako některá písmena. Jejich záměnou bude text pořád stále srozumitelný, ale písmena a číslice jsou opět různě zakódována, čímž se změní otisk zprávy.

**Záměna písmen jejich čárkovanými verzemi**

V dnešní době jsme zvyklí zejména z SMS zpráv a jiných komunikačních technologií číst bez větších problémů text, kde chybějí diakritická znaménka nad písmeny. Přidáním těchto znamének nad písmena, kde v původní zprávě nebyla, uživateli také moc neztížíme pochopení zprávy, ale díky různému zakódovaní znaků změníme otisk zprávy.

**Duplikace písmen ve slově**

Pokud ve slově zdvojíme některé písmeno, uživatel toto slovo celkem snadno přečte a pochopí v původním významu.

**Přeházení písmen ve slově**

Lidský mozek je velmi složitý orgán, což, mimo jiné, dokazuje jeho schopnost přečíst text, ve kterém byla náhodně zpřeházena písmena, a zároveň zůstalo první a poslední písmeno na svém místě, bez větších obtíží.

Příklad:

V suoivsoltsi s vzýukemm na Cmabridge Uinervtisy vlšyo njaveo, že nzeáelží na pořdaí psíemn ve solvě. Jedniná dleůitžá věc je, aby blyy pnvrí a psoelndí pímesna na srpváénm mstíě. Zybetk mžůe být totánlí sěms a ty to přoád bez porlbméů peřčetš. Je to potro, že ldiksý mezok netče kdažé pensímo, ale svolo jkao cleek. Zjíamvaé, že?

**Posíláním HTML zpráv**

Zasíláním zpráv ve formátu HTML pošleme úplně stejně vypadající a znějící text, ale vnitřně je tento text zabalen do HTML značek (tagů), které se uživateli nezobrazují. Jsou interpretovány emailovým klientem.

**Komentáře**

Vkládáním komentářů do HTML zpráv dojde také k výraznému odlišení. Mohou se do nich vkládat různě dlouhé i nesmyslné posloupnosti znaků, aniž by se samotná zpráva nějak změnila, uživateli jsou totiž tyto komentáře skryté.

**Změna fontu, formátování**

U HTML zpráv se změnou fontu, formátování zprávy opticky změní, ale jejich významový obsah se nezmění, pouze bude text různě zbarven, zvýrazněn nebo jinak naformátován. Tagy ovlivňující tento vzhled jsou opět uživateli neviditelné.

**Přidávání textů se stejnou barvou jako pozadí**

Přidáním textu, který má stejnou barvu jako pozadí, se význam zprávy nezmění, neboť tento text je pro uživatele „neviditelný“, ale ne pro hashovací funkci.

**Použití HTML entit**

Některé znaky se dají v HTML jazyce zapsat pomocí entit. Tyto entity jsou uvozeny znakem ‚&‘ a ukončeny středníkem. Entita se dá zapsat pomocí názvu nebo číslem, toto číslo může být zadáno dekadicky nebo hexadecimálně.

**Přidávání obrázků**

Přidáváním obrázků do zprávy se význam zprávy nezmění, naopak se může význam vhodným obrázkem podpořit. U obrázků je však velmi snadné změnit posloupnost bytů, tak aby vypadaly pořád stejně, lidské oko není schopno rozeznat jemné rozdíly v barevnosti, případně můžeme snadno změnit velikost obrázku.

## *3.2 Hash – Hashovací funkce*

Jedná se o n-bitový otisk zprávy, tento otisk vytváří hashovací funkce. Tato funkce (chovající se jako náhodné orákulum) na téměř libovolně dlouhý vstup odpoví vždy nějakým náhodným výstupem délky n (n je dáno typem hashovací funkce). Funkce má tu vlastnost, že na základě znalosti otisku nejsme schopni rekonstruovat vstupní data. Pro dva stejné vstupy dává vždy stejný výstup. Pro různé vstupy vrací různé výstupy; pokud výstupy budou shodné, pak se hovoří o kolizi hashovací funkce. Více informací o hashovacích funkcích, včetně některých schémat a možnostech využití, se lze dozvědět v přednáškách z aplikované kryptografie [4] a na serveru cryptoworld [8].

### 3.2.1 Odolnost proti nalezení kolizí

Hashovací funkce mají mnoho různých vstupních zpráv obvykle $2^{64} - 1$ nebo $2^{128} - 1$, ale otisků je jen $2^n$ (128, 160, 256…), proto existuje velké množství zpráv vedoucích na ten samý otisk, ale nalezení jediné kolize je nad naše výpočetní možnosti. Rozlišují se dva typy odolnosti proti kolizím.

**odolnost proti kolizím 1. řádu**

Tato odolnost požaduje, aby bylo výpočetně nemožné nalezení dvou různých zpráv (jakkoli nesmyslných) $M_1$ a $M_2$, tak aby měly stejný otisk $h(M_1)=h(M_2)$.
Pokud se funkce chová jako náhodné orákulum, pak je složitost nalezení kolize s 50% pravděpodobností rovna $2^{n/2}$. Tato složitost vyplývá z tzv. narozeninového paradoxu.

**odolnost proti kolizím 2. řádu**

Hashovací funkce je odolná proti nalezení druhého vzoru, jestliže pro danou zprávu x je výpočetně nemožné nalézt druhou zprávu $y \neq x$, tak aby jejich otisky byly shodné $h(x)=h(y)$. Rozdíl oproti předchozí odolnosti je v tom, že se hledá shodný otisk k předem dané zprávě a ne libovolnou dvojici zpráv se stejným otiskem.
Pokud se funkce chová jako náhodné orákulum, potom je složitost nalezení druhého vzoru $2^n$.

Pokud známe způsob jak nalézt kolize 1. nebo 2. řádu jednodušeji než jsou uvedené složitosti, pak hovoříme o prolomení hashovací funkce.

### 3.2.2 Konstrukce hashovací funkce

Jelikož zprávy mohou být velmi dlouhé, je potřeba je zpracovávat po částech.
V telekomunikacích je také dostáváme po částech a z kapacitních důvodů není možné je celé ukládat jen pro vytvoření otisku. Aby zpráva mohla být zpracovávána po částech, je nutné ji zarovnat na nějaký násobek bloků, které se budou postupně zpracovávat (obvykle násobek 512b).

Toto zarovnání musí být jednoznačné, aby nedocházelo ke kolizím. Zpráva se doplní bitem 1 a poté následuje 1–512 bitů s hodnotou 0. Dále se používá Demgard-Merklovo zesílení, zpráva je doplněna o její délku. V posledním bloku je rezervováno 64b pro délku zprávy před zarovnáním.

Současné hashovací funkce používají Demgard-Merklův princip konstrukce iterativní hashovací funkce s využitím kompresní funkce F. Výstupem funkce F je tzv. kontext $H_i$, který je n-bitový, vstupem jsou vždy blok zprávy M a předchozí kontext $H_{i-1}$, $H_i=f(M_i, H_{i-1})$ pro první blok se použije inicializační vektor, který je definován typem hashovací funkce. Po provedení funkce F s posledním blokem bude v kontextu výsledný otisk zprávy. Devis-Mayerova konstrukce zesiluje jednocestnost kompresní funkce přičtením (XOR) vstupu před výstupem, $H_i=f(M_i, H_{i-1})= E\ M_i(H_{i-1})\oplus H_{i-1}$

### 3.2.3 Některé hashovací funkce

Existuje několik hashovacích funkcí, lišících se délkou otisku, způsobem výpočtu, maximální délkou zprávy. Některé funkce jsou již prolomené – známe techniky jak u nich nalézat kolize rychleji než u náhodného orákula.

**MD4**

Tato funkce byla definována v roce 1990. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b. Otisk je dlouhý 128b a je uložený ve čtyřech 32b kontextových registrech. Výpočet probíhá ve třech cyklech po 16 rundách, celkem 48 rund a tři kompresní funkce F. Runda označuje provedení jednoho promíchání bitů v kontextových registrech s využitím funkce F. Tato funkce byla prolomena.

**MD5**

Tato funkce byla definována v roce 1991. Vychází z MD4. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b. Otisk je dlouhý 128b a je uložený ve čtyřech 32b kontextových registrech. Výpočet probíhá ve čtyřech cyklech po 16 rundách, v každém cyklu je různá kompresní funkce F, celkem je tedy 64 rund a 4 kompresní funkce. Tato funkce byla prolomena v roce 2004. V roce 2006 český kryptolog Vlastimil Klíma publikoval novou metodu, která nalezne kolize na běžném počítači do minuty.

**SHA-1**

Tato funkce byla definována v roce 1995. Vychází z MDx. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b. Otisk je dlouhý 160b a je uložený v pěti 32b kontextových registrech. Výpočet probíhá ve čtyřech cyklech po 20 rundách, v každém cyklu je různá kompresní funkce F, celkem je tedy 80 rund a 4 kompresní funkce. Tato funkce byla prolomena, přesto složitost nalezení kolizí je stále vysoká.

**Rodina funkcí SHA-2**

Tato rodina byla definována v roce 2003 a 2004, obsahuje funkce SHA-224, SHA-256, SHA-384, SHA-512, SHA-1024, číslo v názvu určuje délku výsledného otisku zprávy.

Funkce mají 80 rund. Funkce SHA-224/256 mají maximální velikost zprávy $2^{64} – 1$ a velikost zpracovávaného bloku 512b. Funkce SHA-384/512/1024 mají maximální velikost zprávy $2^{128} – 1$ a velikost zpracovávaného bloku 1024b.

**Standart SHA-3**

V roce 2008 byla ukončena soutěž o nové hashovací funkce, výsledný standard by měl být uveden do praxe v roce 2012. V soutěži jsou i funkce, na kterých se podíleli čeští kryptologové.

**RIPEMD**

Tato funkce byla navržena v roce 1992. Vychází z MD4. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b, každý vstupní blok je paralelně zpracován ve čtyřech cyklech po 16 rundách. Celkem 2x64 rund provedených paralelně. Otisk je dlouhý 128b. Funkce byla prolomena.

**RIPEMD 128/256**

Tato funkce vznikla v roce 1996, jako náhrada původní RIPEMD. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b, každý vstupní blok je paralelně zpracován ve čtyřech cyklech po 16 rundách. Celkem 2x64 rund provedených paralelně. Pro 256b otisk se v paralelních větvích používají různé inicializační vektory, po aplikaci kompresní funkce se kontexty promíchají a nakonec se sloučí. S 256b otiskem nenarůstá bezpečnost hashovací funkce.

**RIPEMD 160/320**

Tato funkce vznikla v roce 1996. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b, každý vstupní blok je paralelně zpracován v pěti cyklech po 16 rundách. Celkem 2x80 rund provedených paralelně. Pro 320b otisk se v paralelních větvích používají různé inicializační vektory, po aplikaci kompresní funkce se kontexty promíchají a nakonec se sloučí. S 320b otiskem nenarůstá bezpečnost.

**HAVAL**

Tato funkce byla navržena v roce 1992. Vychází z podobných principů jako funkce MDx. Funkce má volitelnou velikost otisku 256/224/192/160/128b a také počet cyklů

výpočtu 3–5. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 1024b. Kontext je 256b a je uložen v osmi 32b registrech. Tato funkce byla prolomena, 128b verze v roce 2004 a 256b verze v roce 2006.

**Tiger**

Tato funkce byla vytvořena v roce 1995. Byla navržena pro rychlost na 64b počítačích. Maximální velikost zpracovávané zprávy je $2^{64} – 1$. Velikost zpracovávaného bloku je 512b. Kontext je uložen ve třech 64b registrech. Velikost otisku je volitelná 192/160/128b. U verze Tiger2 byl akorát změněn způsob zarovnání zprávy, tak aby byl jako u MD5.

**CRC součet**

Kontrolní součty se ve významu této práce, také dají považovat za hashovací funkci. Jsou určené pro kontrolu integrity dat, zejména na detekci chyb vzniklých při selhání techniky. Záměrné pozměnění souboru už detekují hůře. Existuje několik verzí těchto součtů, lišících se zejména délkou otisku.

### 3.2.4 Využití hashovacích funkcí

Mezi nejznámější využití hashovacích funkcí patří kontrola integrity dat, zdali při jejich přenosu nedošlo k chybě, nebo k poškození dat na médiu.

Další využití nacházejí při ukládání hesel. Ukládání hesel v původním tvaru představuje bezpečností riziko. Mnoho uživatelů totiž používá jedno heslo pro více účelů, pokud by útočník získal seznam účtů včetně hesel, mohl by získat přístup i k jiným službám, které uživatel používá. Pro ověření se zadané heslo zhashuje a otisk se poté porovná s uloženým otiskem.

Elektronický podpis, který je založen na asymetrickém šifrování, také používá hashovací funkce. Asymetrické šifrování je pomalé a při podepisování velkých dokumentů by byl čas pro jejich šifrování a dešifrování velký. Z tohoto důvodu se z dokumentu vytvoří otisk hashovací funkcí, který se posléze podepíše a připojí se k dokumentu. Při ověření se tento otisk dešifruje a porovná se s otiskem příchozího dokumentu.

## *3.3 Ukládání otisků*

### 3.3.1 Způsoby vyhledávání

Existuje několik způsobů, jak vyhledávat data v rozsáhlém souboru dat. Tyto techniky se liší zejména paměťovou a časovou složitostí. Další rozdíly jsou v požadavkách na způsob uchovávaní dat a jejich vlastnostech. Více informací o níže uvedených způsobech, včetně názorných ukázek, se lze dozvědět v přednáškách z datových struktur a algoritmů [5].

**Lineární vyhledávání**

Při tomto vyhledávání se postupně porovnávají všechny uložené položky s hledaným vzorkem, zároveň se zjišťuje, zdali se porovnává s poslední položkou. Složitost tohoto vyhledávání je lineární.

**Lineární vyhledávání se zarážkou**

Před samotným vyhledáváním se na konec dat uloží hledaný vzorek, který bude tvořit zarážku. Při vyhledávání se postupně porovnávají všechny uložené položky se vzorkem, ale není potřeba zjišťovat, jestli se porovnává s poslední položkou, díky zarážce se vzorek vždy najde. Po nalezení vzorku se porovná jeho pozice s pozicí zarážky. Pokud jsou shodné, vzorek v uložených datech nebyl. Složitost je opět lineární, ale je rychlejší než předchozí z důvodu jen jednoho porovnání na položku.

**Vyhledávání půlením intervalu**

Tata metoda vyžaduje, aby uložená data byla seřazená. Při vyhledávání se vezme prostřední položka a porovná se s vzorkem. Pokud je vzorek menší, tak se může nacházet jedině v první polovině, pokud je větší, tak je v druhé polovině. Pro další porovnání se použije prostřední prvek příslušné poloviny. Takto se porovnává, dokud se vzorek nenajde a nebo už nebudou žádné položky k porovnání. Složitost vyhledávání je logaritmická. Oproti předchozím metodám je ale pomalejší vkládání prvku, protože se musí uložit na odpovídající místo

**BVS - binární vyhledávací strom**

Pro zkombinování výhod rychlého vyhledávání půlením intervalu a rychlého vkládání prvků při lineárním vyhledávání se používají datové struktury, které představují strom. Tento strom má uzly, které obsahují položku a ukazatele na své potomky. Ukazatelé mohou být dva; jeden nebo žádný. V tomto případě se uzel nazývá listem. Kořen je uzel, který nemá žádné předchůdce. Strom má takovou vlastnost, že položky menší než hodnota v uzlu se nacházejí v levé větvi a větší jsou v pravé větvi. Vyhledávání začíná v kořeni a dále pokračuje potomkem v příslušné větvi. Vyhledávání končí nalezením vzorku a nebo v uzlu, který již nemá potomka, kterým by se mělo pokračovat. Složitost vyhledávání je logaritmická. Složitost vkládání je také logaritmická, pokud se prvek nenajde, přidá se jako nový potomek uzlu ve kterém skončilo vyhledávání. Při mazání je potřeba dávat pozor na potomky mazaného uzlu a správně je přepojit. Při nevhodně vybudovaném stromu, který nemá rovnoměrně rozdělen počet uzlů mezi pravou a levou větev, roste složitost vyhledávání. V extrémním případě, kdy každý uzel má jen jednoho potomka je složitost lineární.

**AVL – stromy**

Jedná se o binární vyhledávací výškově vyvážené stromy. Platí pro něj pravidla, jako pro BVS a navíc pro ně platí, že výška levého a pravého stromu se liší maximálně o jedničku. Složitost vyhledávání, vkládání a mazání je logaritmická. Při vkládání a mazání uzlu je nutné znovu přepočítat výšky stromu, pokud strom bude nevyvážený musí se pomocí rotací vyvážit.

**Červeno-černé stromy**

Opět se jedná o BVS, tentokrát se pro vyvažování používá obarvování uzlů. Každý uzel je buď červený nebo černý. Listy (nil) jsou černé. Červený uzel má oba potomky černé. Každá cesta z uzlu do jeho listů má stejný počet černých uzlů. Složitost operací je opět logaritmická. Oproti AVL stromům nastává více situací k vyvážení, zato ale stáčí maximálně dvě rotace k jejich vyvážení při vkládání uzlu a maximálně tři rotace při mazání.

**B-tree**

Jedná se o strom, kde každý uzel může mít maximálně n potomků. V každém uzlu je seřazeno maximálně n-1 klíčů, které definují rozsah klíčů v potomcích. Každý uzel mimo kořene musí mít minimálně (n-1)/2 klíčů. Výhoda oproti BVS spočívá v automatickém a efektivním vyvažování. Při vkládání/mazání se kontroluje maximální/minimální počet klíčů a podle potřeby se list rozdělí/sloučí. Strom má všechny listy na stejné úrovni. Složitost operací je logaritmická o základu n. Tento způsob vyhledávání se používá v databázích pro indexy.

**Adresní vyhledávání – rozptylování – hashing**

Vyhledávání funguje na principu výpočtu adresy z klíče, která udává umístění prvku v paměti. Výpočet se provádí rozptylovací – hashovací funkcí, která by měla být co nejjednodušší a měla by rovnoměrně využívat adresní prostor a generovat minimum kolizí. V praxi je problém s omezeným adresním prostorem, čímž vzniká množství kolizí, které se musí řešit. Existují dva přístupy. První je zřetězené rozptylování, při kterém se z klíče vypočte adresa s ukazatelem na spojový seznam kolizních prvků. Prvek se vždy vkládá na začátek seznamu, vkládání má konstantní složitost, při vyhledávání se musí projít spojový seznam určený klíčem. Složitost se pochybuje mezi konstantní a lineární, přičemž se spíše blíží konstantní. Druhý je otevřené rozptylování, které kolize řeší novým výpočtem adresy. Nepotřebují se ukazatele, ani spojové seznamy, zato se předem musí znát přibližný počet prvků.

### 3.3.2 Způsoby uložení

Otisky spamu je nutné někde ukládat. Nabízejí se dvě možnosti, první je využití hotových databází a druhá udělání si vlastní s využitím souborů. Při tvorbě vlastní databáze by bylo nutné řešit mnoho problémů s tím souvisejících, paralelní přístup, zálohování, zotavení po havárii, atd. Tyto problémy také vždy řešili programátoři v 60. a 70. letech což byl hlavní podnět pro vznik databází. Využitím stávajících databází se nejen výrazně urychlí vývoj aplikace, ale i rychlost vyhledávání, protože tyto databáze mají za sebou několik let vývoje a optimalizace. Na výběr je několik databází od různých výrobců (Oracle, MS SQL, MySQL, PostgreSQL, FirebirdSQL).

# 4 Návrh řešení

Při návrhu se musí dbát na rychlost výsledné aplikace. Počet emailů zpracovaných v několika procesech by se měl pochybovat mezi 80 000 – 100 000 za minutu.

## *4.1 Návrh výběru z emailu*

Z předchozí analýzy vyplývá, že se musí obsah emailu nějakým způsobem normalizovat, aby se tím vyrušily změny, které spameři v emailu udělali. Po aplikaci této normalizace bude text nesrozumitelný, ale to nevadí, protože tato normalizace má vliv jen pro vytvoření co nejlépe charakterizujícího otisku emailu. Samotný email doručený příjemci zůstane nedotčen. Normalizace se docílí vynecháním některých částí a znaků a substitucí některých znaků podle následujících pravidel.

### 4.1.1 Vynechání

Z emailu se samozřejmě vynechají hlavičky, neboť jsou u každého emailu různé. Kromě hlaviček se budou vynechávat i řetězce, které označují hranice mezi různými částmi emailů.

Přílohy emailů se také budou vynechávat, neboť jejich úprava, tak aby změnily otisk, je velmi snadná.

V HTML zprávách je nutné vynechávat HTML tagy, odpadne tak možnost změny otisku pomocí komentářů, změnou fontů a formátování. Aby se ještě efektivněji eliminoval vliv fontů a formátování, je nutné vynechat celý text mezi tagy <style> a </style>, kde jsou definovány kaskádové styly. Spameři do nich dokonce vkládají i texty článků z novin.

Další tag, u kterého se vynechá obsah je <script>. V tomto tagu se většinou objevuje javascript a ten se dá velmi snadno upravit, tak aby změnil otisk, stačí přejmenovat používané proměnné. Text mezi tagy <a> a </a> se také vynechá, obsahuje zobrazovaný text pro nějaký odkaz, tento text se může velmi snadno změnit, ale cíl odkazu je pořád stejný. Poslední tag, u kterého se vynechá text je <applet>, obsahující java applet.

Jelikož se ve spamech objevují odkazy na stránky se stejným obsahem, ale jen lehce pozměněnou adresou, u HTML zpráv se cílová adresa nemusí vůbec změnit, stačí pozměnit jen zobrazovanou část, budou se i odkazy vynechávat.

Znaky mimo písmena a číslice se budou také vynechávat. Eliminuje se tím možnost změnit otisk různým počtem bílých znaků, možnost optického dělení slov pomocí podtržítek vykřičníků atd.

Ve slovech, kde se vyskytuje některý znak vícekrát za sebou, se použije jen jeho první výskyt a další se vynechají.

### 4.1.2 Náhrada

Zamění se velká písmena za malá. Odpadá možnost modifikace otisku změnou velikosti písmen.

Zamění se číslice korespondujícími písmeny:

| | |
|---|---|
| 0 | o |
| 1 | l,i |
| 2 | z |
| 3 | e |
| 4 | a |
| 5 | s |
| 6 | g |
| 7 | t |
| 8 | b |
| 9 | g |

**tabulka 1: písmena odpovídající číslicím**

Protože jednička se dá zaměnit jak za *i* tak *l* a nelze bez hlubší analýzy textu určit za co je použitá, tak se zamění za písmeno *i* a veškeré výskyty písmena *l* v textu nahradíme písmenem *i*. Písmeno *i* bylo vybráno, protože se v textech vyskytuje častěji.

Nahradí se písmena s diakritikou za jejich verze bez diakritických znamének.

### 4.1.3 Další postupy

Spam nemívá velkou velikost, proto se může vynechat ze zpracování email, který bude mít velikost větší než je určitá hranice.

Techniku, při které spameři přidávají na konec emailu znaky, případně celé články není možně jednoznačně eliminovat, protože u různých emailů je velikost tohoto přídavku různá, ale může se částečně potlačit vynecháním určitého procenta zprávy. Toto procento se bude počítat z celkového počtu znaků, které se vybrali pro hashovaní. Znaky a ne slova pro výpočet tohoto vynechání byly vybrány, protože by se problematicky stanovovala hranice mezi slovy. Díky optickému dělení znaků ve slovech, není možné tuto hranici jednoznačně určit, neví se totiž, zdali vykřičník (tečka atd.) odděluje slovo a nebo jenom znaky ve slově.

### 4.1.4 Neeliminované techniky

Mezi popsanými technikami je několik, které nejdou efektivně eliminovat. Jedná se o náhradu slov synonymy, pro odstranění by se musela vytvořit databáze synonym pro veliký počet slov a každé slovo v emailu by se muselo nahradit nějakým vybraným slovem z databáze. Pro efektivní náhradu by bylo nutné vymyslet, jak správně rozpoznat hranici mezi slovy. A otestovat jak moc by implementace zpomalovala aplikaci.

Další metoda, která nelze snadno odstranit je přidávání textů se stejnou barvou, jako má pozadí, pro eliminaci by se musela provádět syntaktická a sémantická analýza HTML emailů, což by zabralo hodně výpočetního času.

Přidání náhodných znaků, popřípadě úryvků z novin se částečně odstraní procentuální hranicí, další problém s přidáváním nastane, až se začne objevovat na začátku emailu, případně někde v jeho těle.

Proházení písmen ve slově je také problematicky odstranitelné. Řešení tohoto problému by mohlo být vytvoření slovníku. Podle počtu písmen, prvního a posledního písmena by se zkoušelo, která slova ve slovníku obsahují další písmena vyskytující se ve slově. Problém by byl ve velikosti tohoto slovníku, musel by se vytvořit pro různé jazyky.

V takto rozsáhlé databázi by se vyhledávalo celkem pomalu i za použití indexů, protože ty by odkazovali pouze na množinu slov začínající na stejné písmeno a stejným počtem znaků. Celá tato množina by se poté musela porovnat jednotlivě.

Mocně se rozšiřující posílání spamu jen ve formě obrázků není možné detekovat pomocí otisku, bylo by na to potřeba použít techniky pro rozpoznávání vzorů v obraze, bohužel tyto techniky jsou výpočetně velmi náročné. Obdobný problém nastane, pokud se spamy začnou rozesílat ve formě pdf dokumentů.

## *4.2 Výběr hashovací funkce*

Nejprve je potřeba se zamyslet nad velikostí otisku, který se bude vytvářet. Moc velký otisk bude zpomalovat vyhledávání v databázi a také bude zvyšovat paměťové nároky na uložení databáze. Naopak moc malý otisk bude zvyšovat riziko kolizí zpráv, neboť čím menší délka otisku, tím více zpráv bude vést na stejný otisk.

Předpokládá se, že aplikace bude zpracovávat 80 – 100 tisíc emailů za minutu to dává 144 miliónů zpráv za den, za rok 52,56 miliardy zpráv, $2^{36}$ je 68,7 miliardy. Podle narozeninového paradoxu [4] stačí pro 50-ti% pravděpodobnost, že se v množině vyskytují dvě zprávy se stejným otiskem $2^{n/2}$ zpráv. Pro 128b otisk by se dostalo potřebné množství emailů za 268 miliónů let při zaokrouhlení $2^{36}$ zpráv ročně. Toto číslo se jeví jako dostatečné.

Možnost útoku na hashovací funkci (aplikaci) není příliš velká, tyto možnosti snižuje několik faktorů:

V databázi se udržují otisky pouze zpráv, o kterých se ví, že jsou určitě spam. Tyto otisky přidal někdo zodpovědný do databáze (případně nějaký složitější a přesný filtr spamu). Pro znehodnocení tohoto otisku by bylo potřeba, aby se shodoval s otiskem několika vyskytujících se normálních emailů (hamů), ale zároveň měl výrazně větší podíl jako spam, tím by docházelo k chybné detekci emailů.

Pro hash jsou použity jen malá písmena, díky tomu jsou omezeny možnosti tvorby kolizních emailů.

Spamer nezná prováděné transformace s emailem a nezná procentuální hranici pro vynechání znaků. Na tento faktor se ale nelze stoprocentně spoléhat, jenom trochu stíží tvorbu cílených otisků.

Pro spamera je rychlejší a tím i výhodnější upravit rozesílané spamy, tak aby měli různý otisk, než je zacílit na nějaký konkrétní otisk.

Z těchto důvodů jsem se rozhodl použít pro tvorbu otisků hashovací funkci MD5, která má délku otisku 128b, je rychlá a měla by k účelům této aplikace plně dostačovat.

## *4.3 Výběr databáze*

Ze zadání vyplývá, že program musí fungovat pod různými unixovými platformami. Dále není potřeba žádná složitá databáze s extra funkcemi navíc, proto by bylo zbytečné vybrat nějakou komerční databázi a zbytečně platit za její licenci.

Při splnění předchozích podmínek připadají v úvahu tyto databáze:

- Firebird
- PostgreSQL
- MySQL

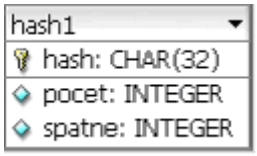
Jelikož jsem s MySQL již pracoval a byl jsem s ní spokojen, upřednostnil jsem tuto databázi. Další z faktorů bylo také velmi dobře zdokumentované API pro jazyk C.

### 4.3.1 Volba enginu tabulek

MySQL má několik enginu pro tabulky, nejvíce používané jsou MyISAM a InnoDB. Hlavní rozdíl mezi těmito enginy je v podpoře transakcí. Další rozdíl je v blokování dat při updatu. Jelikož aplikace bude běžet v několika procesech a nevyžaduje používání transakcí je toto nejdůležitější kritérium. MyISAM blokuje při updatu celou tabulku, kdežto InnoDB jen příslušný řádek. Proto jsem jako engine databáze zvolil InnoDB.

### 4.3.2 Návrh tabulek

Pro uchovávání otisků spamu stačí jednoduchá tabulka (obrázek 1), která bude mít tři sloupce. Jako klíčový prvek bude přímo zvolen otisk, protože je unikátní a budou se podle něj řídit všechny operace s databází, každý SQL dotaz bude mít podmínku na shodu otisku. Nad klíčovým prvkem je vybudován index, který je realizován pomocí B-tree, takže vyhledávání bude rychlé. Jelikož se očekává velké množství otisků v databázi, bude v hodné tuto jedinou tabulku duplikovat a v každé uchovávat jen otisky začínající určitými znaky. Zvýší se tím výkonnost databáze, protože se nebude muset operovat s tak velkými indexy, jako kdyby otisky byli v jedné tabulce.




**obrázek 1: E-R Diagram**

# 5. Realizace

Pro rychlejší kompilaci a lepší strukturu jsou kódy rozděleny do tří souborů. V jednom jsou veškeré funkce pro výpočet otisku MD5, v dalším se nacházejí funkce pro různé kódování znaků a v posledním jsou funkce již pro samotný výběr znaků z emailu a pro následnou manipulaci s databází otisků. Cílový program využívá oddělenou kompilaci.

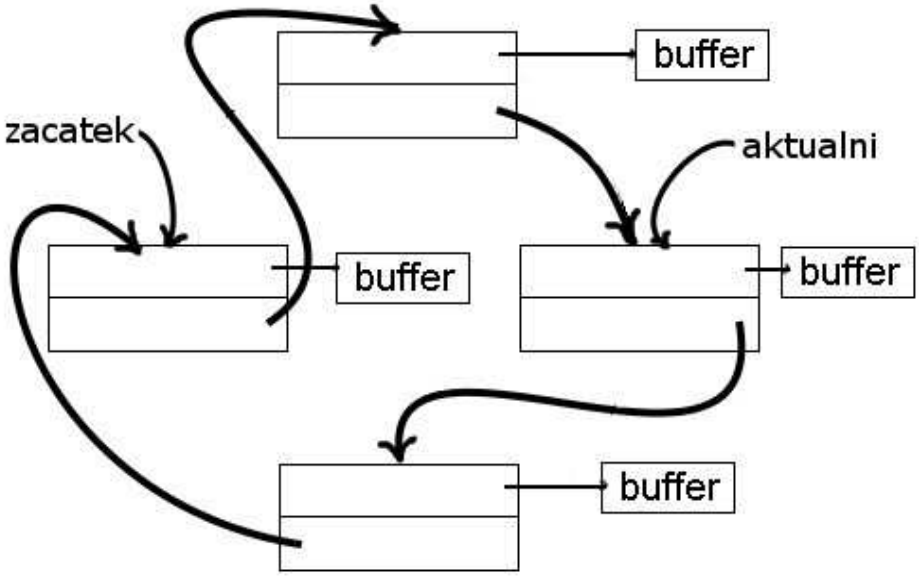
## *5.1 Datové struktury*

### 5.1.1 FRONTA

Počet znaků, které se budou z konce zprávy vynechávat, nelze zjistit dopředu. Je pouze známa hranice v procentech a konkrétní číslo se může vypočítat pouze, až bude znám celkový počet znaků, které se vybrali z emailu. Jelikož je neefektivní udržovat v paměti všechny znaky dokud se nebude vědět kolik jich má být odebráno, vytvořil jsem si datovou strukturu FRONTA.

Struct FRONTA má tyto proměnné:

- next – ukazatel na další prvek typu FRONTA
- buff – ukazatel na unsigned char (buffer pro vybrané znaky)




**obrázek 2: zapojení bufferů do kruhové fronty**

Po zjištění velikosti souboru a určení začátku těla emailu se spočte maximální počet přidaných znaků a z toho potřebný počet bufferů k jejich uchování. Tyto buffery se přiřadí do structů FRONTA a ty se postupně zapojí do kruhové fronty (obrázek 2). Po naplnění celé fronty znaky, se bude moct udělat otisk z následujícího bufferu ve frontě a poté se naplní novými znaky. Po zpracování celého emailu bude v kruhové frontě dostatečný počet znaků k vynechání a bude znám i celkový počet znaků. Nyní se bude moct vypočítat, kolik znaků se vynechá, po tomto výpočtu se přidají do hashe zbývající znaky, které se ještě mohou z fronty použít. Díky této frontě se podstatně ušetří potřebná paměť pro běh programu. Řešení, při kterém by se znaky nejprve uložili do pomocného souboru a po zpracování emailu se z něj opět načetl potřebný počet znaků (již by se znala přesná hranice), by sice nepotřeboval paměť navíc, ale byl by znatelně pomalejší a neefektivní.

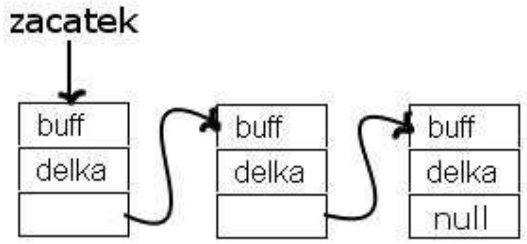
### 5.1.2 BOUNDER

Emaily mohou mít různý počet části, počet závisí pouze a jen na vůli odesílatele. Na něm také je, kolik různých hranic mezi těmito částmi bude. Pro správné rozpoznání jednotlivých částí je potřeba tyto hranice uložit. Počet hranic se dopředu nezná, proto jsem si pro ně vytvořil datovou strukturu BOUNDER.

Struct BOUNDER má tyto proměnné:

- Buff – pole znaků, ve kterém bude uložen řetězec tvořící hranici
- Delka – počet znaků, které tvoří hranici
- Next – ukazatel na další prvek typu BOUNDER

Při naleznutí definice hranice se příslušný řetězec uloží do bufferu. Tento buffer se přiřadí do structu BOUNDER a ty se postupně zapojí do zásobníku (obrázek 3).




**obrázek 3: zapojení hranic do zásobníku**

### 5.1.3 MD5_data

Tato struktura je určena pro hashovací funkci MD5. Uchovává kontextové registry hashovací funkce.

Struct MD5_data má tyto proměnné:

- Otisk – pole, které obsahuje čtyři 32b kontextové registry
- Počet – obsahuje dva 32b registry obsahující délku dosud zhashované zprávy
- Hotovo – určuje, zdali je otisk zprávy již hotový

## *5.2 Hashovací funkce MD5*

Aby se zaručila nezávislost tohoto kódu na uspořádání paměti (LITTLE, BIG-ENDIAN), použily se direktivy pro preprocesor jazyka C, které zajistí podmíněný překlad podle existence makra LITTLE_ENDIAN.

### 5.2.1 Makra

Makro pro cyklický posuv 32b registru o s bitů do leva:

```
#define rol(value, bits) (((value) << (bits)) | ((value) >> (32 - (bits))))
```

Následující makro zajišťuje obrácení pořadí bytů u 32b registrů, podle uspořádání paměti:

```
#ifdef LITTLE_ENDIAN
      #define konstt(l) ((rol(l,24)&0xFF00FF00)|(rol(l,8)&0x00FF00FF))
#else
      #define konstt(l) l
#endif
```

Toto makro vezme 32b hodnotu l např. 11 22 33 44 a pokud je definováno makro LITTLE_ENDIAN provede následující operace:
Číslo se cyklicky posune o 24b do leva, dostane se 44 11 22 33, s tím se provede logický součin s FF 00 FF 00, dostane se 44 00 22 00, a to se logicky sečte s číslem 00 33 00 11. To se dostalo cyklickým posuvem původního čísla o 8b do leva na 22 33 44 11, které se poté logicky vynásobilo s 00 FF 00 FF. Po výsledném součtu se dostane 44 33 22 11.

Pomocí maker jsou definovány i kompresní funkce F, G, H, I:

```
#define F(b,c,d) (((b)&(c)) | ((~b)&(d)))
#define G(b,c,d) (((b)&(d)) | ((c)&(~d)))
#define H(b,c,d) ((b)^(c)^(d))
#define I(b,c,d) ((c)^((b) | (~d)))
```

Pro provedení 16 rund s odpovídajícími funkcemi F, G, H, I jsou použita následující makra:

```
#define RF(a,b,c,d,m,k,s) \
        (a)+=F(b,c,d) + (m) + (k);\
        (a)=rol(a,s); \
        (a)+=(b);

#define RG(a,b,c,d,m,k,s) \
        (a)+=G(b,c,d) + (m) + (k); \
        (a)=rol(a,s); \
        (a)+=(b);

#define RH(a,b,c,d,m,k,s) \
        (a)+=H(b,c,d) + (m) + (k); \
        (a)=rol(a,s); \
        (a)+=(b);

#define RI(a,b,c,d,m,k,s) \
        (a)+=I(b,c,d) + (m) + (k); \
        (a)=rol(a,s); \
        (a)+=(b);
```

### 5.2.2 Funkce

Samotný výpočet otisku zajišťují následující funkce:

- void MD5_Init(MD5_data * context)
- void MD5_TRANS(UINT4 *otisk, UINT4 *vstup2)
- void MD5_Update(MD5_data *context, unsigned char *buffer,unsigned int delka)
- void MD5_Finish(MD5_data *context)

Funkce MD5_Init inicializuje kontextové registry na výchozí hodnoty a vynuluje počet již zhasovaných znaků.

Funkce MD5_TRANS provádí výpočet otisku pro předaný blok zprávy (512b).

Funkce MD5_Update přičte předanou délku k počtu již zhasovaných znaků. A na každých 512b předaného buffru zavolá funkci MD5_TRANS. Provede případné zarovnání zprávy.

Funkce MD5_Finish provádí transformaci s finálním blokem zprávy, po této transformaci budou v kontextových registrech již výsledné bity otisku zprávy.

## *5.3 Kódovaní znaků s hodnotou větší než 127*

Pro správnou náhradu znaků s hodnotou větší než 127, jedná se o písmena s diakritickými znaménky nebo znaky vypadajícími jako normální písmena, slouží pro každé kódování vlastní funkce, která má dva vstupy:

- adresa paměti uloženého znaku, hodnota znaku bude změněna podle převodní tabulky
- adresa paměti, čítače pozice znaku v buffru, při vynechání znaku se dekrementuje

## *5.4 Databáze otisků spamu*

Databáze otisků spamu v základním nastavení obsahuje osm identických tabulek, každá tabulka je určená pro otisky začínající dvěma různými znaky. Tímto rozdělením se zvýší výkon databáze, protože při vkládání nových dat nebude potřeba přebudovávat index se všemi otisky, ale teoreticky jen s jednou osminou otisků. To samé platí o vyhledávání, protože strom nebude muset být tak hluboký jako při použití jedné tabulky.

S databází otisků se budou provádět tři druhy operací:

- vložení nového otisku
- vyhledání otisku v databázi
- zvýšení počtu špatného označení otisku

Každou operaci bude obstarávat jedna funkce.

### 5.4.1 Vložení otisku

Tuto operaci má na starost funkce *pridej_otisk()*. Funkce má tři vstupy adresu paměti, kde je uložen otisk zprávy, adresu paměti, kde je uložen název tabulky a adresu paměti, kde je uloženo spojení s databází. Funkce vloží do zadané tabulky nový otisk. Pokud se otisk v tabulce již vyskytoval, tak otisk znovu přidán nebude.

### 5.4.2 Vyhledání otisku

Tuto operaci má na starost funkce *vyhledej_otisk()*. Funkce má tři vstupy adresu paměti, kde je uložen otisk zprávy, adresu paměti, kde je uložen název tabulky a adresu paměti, kde je uloženo spojení s databází. Funkce vyhledá daný otisk v příslušné tabulce a zjistí na kolik procent je otisk spam, pokud se daný otisk našel v tabulce, tak funkce inkrementuje počet výskytů tohoto otisku v tabulce. Návratová hodnota funkce je celé číslo udávající na kolik procent je otisk spam, pokud otisk v databázi nebyl, vrací funkce nulu.

### 5.4.3 Zvýšení počtu špatného označení otisku

Tuto operaci má na starost funkce *chybny_otisk()*. Funkce má tři vstupy adresu paměti, kde je uložen otisk zprávy, adresu paměti, kde je uložen název tabulky a adresu paměti, kde je uloženo spojení s databází. Funkce provede dotaz, kterým inkrementuje počet udávající kolikrát byla zpráva s tímto otiskem špatně vyhodnocena jako spam.

## *5.5 Výběr znaků pro otisk*

Tuto část aplikace má na starosti funkce *vyber()*, vstupem je adresa paměti kam se uloží otisk zprávy, název souboru, ve kterém je uložen email a adresa paměti kde bude uložen název tabulky, do které daný otisk patří. Funkce si nejprve tento soubor otevře a zjistí si jeho velikost, pokud je větší než předdefinované makro, tak soubor zavře a vrací makrem definovanou hodnotu. Poté funkce začne číst soubor po znacích a vyhledává hlavičky definující kódování znaků, kódování při transportu a definice hranic mezi částmi emailu, dokud nenarazí na začátek těla emailu.

Nyní od velikosti souboru odečte aktuální pozici v souboru a spočítá maximální počet znaků, které se již pro otisk nepoužijí. Počet potřebných bufferů pro uchování těchto znaků se dostane, tak že se počet znaků vydělí definovanou délkou buffru a přičte se k ní dvojka, protože poslední znaky nemusí být přesně na hranicích bufferů, ale mohou jít přes tuto hranici a také počet vynechávaných znaků nemusí odpovídat násobku délky buffru.

Poté se vytvoří nový prvek FRONTA a soubor se čte dál po znaku, dokud se nenaplní buffer použitelnými znaky. Dokud nebude počet buffrů odpovídat potřebnému množství, tak se po naplnění aktuálního bufferu vytvoří nový prvek FRONTA a zapojí se za aktuální. Poslednímu potřebnému bufferu se jako následník připojí prvek FRONTA s prvním bufferem, čímž se dostane kruhová fronta (obrázek 2). Po naplnění všech potřebných bufferů, se již nový prvek FRONTA nevytváří, ale následující buffer ve frontě se zhasuje a poté se do něj zapisují nové použitelné znaky.

Po dosažení konce souboru se z celkového počtu vybraných znaků zjistí počet znaků, které se vynechají, a z buffrů se ještě zhasuje zbývající počet použitelných znaků. Tím se dostane výsledný otisk emailu.

Funkce nakonec zjistí na základě prvního znaku otisku, do jaké tabulky tento otisk patří.

### 5.5.1 Makra použitá při výběru

Program má definováno několik konstant pomocí maker. Tyto definice zjednodušují změnu některých parametrů a v případě maker pro kódování i běh programu, protože je jednodušší porovnat dvě čísla než dva řetězce. Chybové konstanty začínající prefixem ER_ musejí mít hodnotu větší než 100, aby je nebylo možné zaměnit s hodnotou pravděpodobnosti spamu.

| | |
|---|---|
| ADDON_LIMIT | určuje kolik % vybraných znaků se použije pro otisk |
| FILE_LIMIT | maximální velikost souboru v bytech, který se bude hashovat |
| BOUNDARY_SIZE | maximální velikost řetězce určujícího hranici mezi částmi emailu |
| BUFFER_SIZE 512 | velikost bufferu, do kterého se ukládají znaky pro otisk |
| HTML | typ těla emailu je HTML |
| JINY | typ těla emailu není text ani multipart |
| MULTI | typ těla emailu je multipart |
| BIT7 | transfer-encoding je 7-BIT |
| BIT8 | transfer-encoding je 8-BIT |
| BINARY | transfer-encoding je BINARY |
| BASE | transfer-encoding je BASE64 |
| QUOTED | transfer-encoding je quoted-printable |
| UNKNOW | |
| ISO81 | kódování iso-8859-1 |
| ISO82 | kódování iso-8859-2 |
| CP1250 | kódování windows-1250 |
| CP1251 | kódování windows-1251 |
| CP1252 | kódování windows-1252 |
| USASCII | kódování US-ASCII |
| UTF8 | Kódování UTF-8 |
| VELKY | určuje, že soubor je větší než FILE_LIMIT |
| NULOVY | určuje, že ze zprávy nebyl vybrán ani jeden znak pro otisk |
| ER_FILE_OP | určuje, že se nepodařilo otevřít soubor s emailem |
| ER_DB_INIT | určuje, že se nepodařilo inicializovat připojení k MySQL |
| ER_DB_NOC | určuje, že se nepodařilo připojit k MySQL |
| ER_DB_QUE | určuje, že se vyskytla chyba při vykonávání SQL dotazu |
| ER_DB_STORE | určuje, že se nepodařilo uložit výsledek SQL dotazu |
| ER_ARG | určuje, že se předali programu špatné argumenty |

**tabulka 2: definovaná makra**

### 5.5.2 Funkce využívané výběrem

void charset(FILE *vst) – funkce pro vyhledání znakové sady definované pro textovou část emailu

void norm(FILE *vst) – přesné určení znakové sady pro textovou část emailu

void a_tagy(FILE *vst) – funkce pro odstranění textu mezi tagy <a></a> a <applet></applet>

void style(FILE *vst) – funkce pro odstranění textu mezi tagy <style></style> a <script></script>

void entity(FILE *vst, unsigned char *cteny, int *pocet) – funkce pro náhradu HTML entit

int kontrol(FILE *vst,BOUNDER *zac) – funkce porovnává řetězec hranice se čtenými znaky

int porovnani(FILE *vst) – funkce hledající konec hlaviček (CRLFCRLF)

int adresa(FILE *vst,int *pozice) – funkce pro vynechání internetové adresy

BOUNDER *hrana(FILE *vst, int *type, BOUNDER *zac,int *naslo,int *transfer) – funkce hledající hlavičky definující typ části emailu a transportní kódování této části

## *5.6 Kompilace*

Pro kompilaci celé aplikace je vytvořen Makefile a využita oddělená kompilace. Tím se docílí větší rychlosti při opětovné kompilaci, protože se zkompilují jen ty části, které se změnili.

Makefile:

```
hash: kodovani.o md.o hash.o
      g++ -Wall –ansi –O3 kodovani.o md.o hash.o
      strip hash
hash.o: hash.c kodovani.h md.h
      g++ -Wall –ansi –O3 –c hash.c –o hash.o
kodovani.o: kodovani.c kodovani.h
      g++ -Wall –ansi –O3 –c kodovani.c –o kodovani.o
md.o: md.c md.h
      g++ -Wall –ansi –O3 –c md.c –o md.o
```

## *5.7 Rozšíření aplikace*

Aplikace byla implementována, tak aby v budoucnu bylo možné bezproblémové rozšíření o další typy kódování. Stačí napsat novou převodní funkci pro dané kódování, definovat makro pro jeho identifikování a ve funkci *norm()* toto kódování detekovat. Ve funkci *vyber()* do switche přidat návěští na volání příslušné funkce pro převod tohoto kódování.

Případné další změny v náhradách znaků do hodnoty 127 nejsou také složité, stačí opět přidat návěští do příslušného switche ve funkci *vyber()* a tam znak nahradit.

# 6 Testování aplikace

Testování správné funkčnosti aplikace se dělo průběžně, vždy po přidání nějaké nové funkcionality. Aplikace byla testována pomocí programu valgrind na memory leaky.

Ze zkušebního vzorku 4 111 spamů se získalo po odstranění všech hlaviček a řetězců určujících hranice mezi jednotlivými částmi emailů 3 635 jedinečných otisků.

Po aplikaci všech navržených metod se získalo 3 945 otisků emailů, toto číslo je menší než celkový počet spamů, protože pro velké emaily se otisky netvoří a pro emaily, kde nebylo vybráno nic pro hashování se také netvoří, snižuje se tím riziko false pozitive. Z tohoto počtu bylo 3 221 jedinečných otisků.

| | Odstranění hlaviček | Všechny techniky |
|---|---|---|
| Spamů | 4 111 | 4 111 |
| Vytvořených otisků | 4 111 | 3 945 |
| Jedinečných otisků | 3 635 | 3 221 |
| Nalezených duplicit | 476 | 724 |
| Nalezených duplicit v % | 100 | 152,42 |

**tabulka 3: Počty nalezených duplicit**

Jak je vidět, tak aplikace na daném vzorku spamu zvýšila výskyt duplicit o padesát dva procent. Pro přesnější číslo by bylo potřeba aplikaci otestovat na daleko větším vzorku spamu řádově ve statisících a mít aspoň přibližný odhad kolik se v něm vyskytuje informačně různých spamů, aby se vědělo, kolik jich aplikace nebyla schopna normalizovat.

Jako další test se provedlo změření doby běhu aplikace.

| PC s MySQL serverem: | Notebook s linuxem |
|---|---|
| AMD Barton 2600+ | AMD Sempron 3000+ |
| 512 MB DDR RAM 333 MHz CL 2.5 | 768 MB RAM |
| Seagate 120 GB 8 MB cache | integrovaná grafika |
| Integrovaná LAN 100 Mbit | LAN 100Mbit |
| OS MS Windows 2000 | OS Linux Slax live USB verze (256 MB) |
| MySQL 5.0.27-community-nt | Vzorky spamu na druhém flash disku |

**tabulka 4: Testovací sestavy**

Při instalaci MySQL byl zvolen typ serveru developer machine, multifunctional database. Databáze se ukládala do jednoho souboru a proměnná innodb_buffer_pool_size byla nastavena na 70MB, ostatní proměnné byly ponechány v původním nastavení. V databázi bylo 8 identických tabulek, každá byla naplněna miliónem záznamů.

První verze programu byla pouštěna skriptem, který spouštěl aplikaci se zadanými parametry. Plánovaný počet zpracovaných zpráv byl 20 000. Po více jak dvou hodinách jsem aplikaci ukončil. Za tuto dobu se stačilo zpracovat pouze necelých 2 000 zpráv. Tuto verzi nejvíce zpomalovalo opětovné navazování spojení s MySQL serverem. Odhadovaná doba navázání spojení je kolem čtyř vteřin. Další vliv na dobu běhu mělo i opětovné spouštění aplikace pro každý spam.

Ve druhé verzi se spustila aplikace, která navázala spojení s MySQL serverem a poté ve for cyklu zpracovala požadovaných 20 000 zpráv. Doba běhu byla přibližně 2 minuty. Když se tato verze pustila pětkrát za sebou nad stejnými daty, tak průměrná doba zpracování byla třicet čtyři vteřin.

Z těchto měření je vidět, že při nasazení v reálném provozu by opětovné navazování spojení s MySQL serverem aplikaci výrazně zpomalovalo. A nebyla by prakticky použitelná.

Třetí verze byla udělána pro lepší simulaci reálného nasazení. Byl naprogramován tcp server, který v hlavním vlákně vytvořil listen socket a poté vytvořil potomky pomocí vláken, kteří navázali spojení s MySQL serverem a poté přijímali spojení s klienty. Hlavní vlákno v intervalu tří sekund kontrolovalo počet potomků a případně doplňovalo jejich počet na výchozí hodnotu. Klient po připojení poslal soubor serveru a počkal na odpověď. Klienti byli spouštěni skriptem. K dispozici jsem měl jen dva počítače, protože na jednom běžel MySQL server pod systémem windows 2000, tak server a klienti běželi na stejném notebooku. V praxi by k době zpracování přispěla i komunikace mezi serverem a klientem přes síť. Naopak dobu zpracování, při použití více akceptačních vláken, by snížil vyšší počet jader procesorů. Doba běhu s jedním akceptačním vláknem byla přibližně tři minuty. Když se tato verze pustila pětkrát za sebou nad stejnými daty, tak průměrná doba zpracování byla jedna minuta a dvacet jedna vteřin. Při použití deseti akceptačních vláken byla doba zpracování přibližně tři minuty. Pro spuštění klientů byl použit skript, který spustil deset skriptů na pozadí. Každý skript na pozadí ve smyčce spustil klienta dvatisícekrát nad různými emaily. Když se tato verze spustila pětkrát za sebou nad stejnými daty, tak průměrná doba zpracování byla padesát pět vteřin.

| | Verze 1 | Verze 2 | Verze 3 | |
|---|---|---|---|---|
| | | | 1 vlákno | 10 vláken |
| Počet vzorků | < 2 000 | 20 000 | 20 000 | 20 000 |
| Doba běhu | 2h 15m | 2m 06s | 3m 54s | 3m 11s |
| Doba běhu | - | 1m 56s | 2m 09s | 2m 47s |
| Doba běhu | - | 1m 52s | 2m 50s | 3m 26s |
| Doba běhu | - | 1m 23s | 3m 05s | 3m 07s |
| Doba běhu | - | 2m 15s | 3m 09s | 2m 44s |
| Průměrná doba běhu z 5-ti spuštění za sebou | - | 34s | 1m 21s | 55s |

**tabulka 5: Doba běhu aplikace**

V reálném nasazení by tcp server mohl dynamicky vytvářet předpřipravená akceptační vlákna na základě rozdílu počtu aktuálně komunikujících vláken a počtu vláken. Server vytvořený pro testování je na tuto možnost připraven, stačí jen upravit hlavní vlákno, aby kromě náhrady ukončených vláken přidávalo nebo mazalo akceptační vlákna na základě rozdílu počtu potomků a počtu komunikujících vláken.

# 7 Závěr

Na základě výsledků testů (tabulka 3) se mi povedlo implementovat aplikaci, která výrazně zlepšuje detekci spamu pomocí jejich otisků. Podle těchto testů aplikace na daném vzorku spamu odhalila o padesát dva procent více duplicitních spamů, než aplikace, která jen odstranila hlavičky.

Přesné nastavení velikosti emailů, které se již nebudou aplikací zpracovávat a počtu procent kolik znaků se vynechá, je už na lidech ze seznamu. Pro toto nastavení budou muset provést několik sérií testů s různými hodnotami a hlavně s větším a rozmanitějším vzorkem dat, aby získali výsledky s větší vypovídající hodnotou než, které jsem získal já na omezeném vzorku spamu.

Budoucí vylepšení aplikace bych viděl v zahrnutí více druhů kódování. Také by nebylo marné se zamyslet nad možností vytvořit otisk pro každou textovou část emailu. Zároveň by se muselo dořešit, jak nejlépe zkombinovat získané hodnoty pravděpodobností pro tyto otisky. Také by se mohlo promyslet, jak efektivně eliminovat synonyma a otestovat, zdali by se vyplatilo zpomalení běhu aplikace implementací této techniky, vzhledem k počtu nově odhalených duplicit.

# Zdroje informací


[1] Domovská stránka freemailu od seznamu
http://login.szn.cz/

[2] Zpráva o podílu spamu v dubnu 2009
http://www.financninoviny.cz/tema/index_view.php?id=373869&id_seznam=439

[3] Článek o spamu na securityworldu
http://securityworld.cz/securityworld/spam-se-spamem-na-vecne-casy-988

[4] Lórencz, Róbert. *Přednášky z předmětu aplikovaná kryptografie.* Praha, 2006. Slajdy z přednášek. Fakulta elektrotechnická. ČVUT v Praze.

[5] Felkel, Petr - Genyk-Berezovskyj, Marko. *Přednášky z předmětu datové struktury a algoritmy.* Praha, 2005. Slajdy z přednášek. Fakulta elektrotechnická. ČVUT v Praze.

[6] RFC k SMTP
http://tools.ietf.org/html/rfc2821

[7] RFC k MIME
http://tools.ietf.org/html/rfc2045

[8] Server věnující se šifrování a bezpečnosti
http://www.crypto-world.info/

[9] Stránky o programování v C/C++ včetně dokumentace
http://www.cplusplus.com/

[10] Stránky s ukázkami spamů
http://www.pooh.cz/pooh/r.asp?r=265&db=1001

[11] Seriál o programování v C/C++
http://www.linuxsoft.cz/article_list.php?id_kategory=186

[12] Stránky o programování v C/C++
http://www.sallyx.org/sally/c/

[13] Referenční manuál MySQL 5.0
http://dev.mysql.com/doc/refman/5.0/en/

[14] Porovnání rychlosti bezpečnostních algoritmů v knihovně Crypto++
http://www.cryptopp.com/benchmarks.html

[15] Stránky s definicí znaků pro různé znakové sady
http://www.unicode.org/Public/MAPPINGS/

[16] Projekt hashcash
http://www.hashcash.org

[17] Projekt SURBL
http://www.surbl.org/

[18] Projekt Nilsimsa a jeho vylepšení
http://spdp.dti.unimi.it/papers/pdcs04.pdf

[19] Projekt Pyzor
http://pyzor.sourceforge.net/

[20] Projekt Razor
http://razor.sourceforge.net/

[21] Projekt DCC – Distributed Checksum Clearinghouse
http://www.rhyolite.com/dcc/

[22] Kortus, Jaroslav. *Detekce spamu.* Brno, 2006.
Diplomová práce. Fakulta informatiky. Masarykova univerzita.
http://is.muni.cz/th/51525/fi_m/diplomka-final.pdf

## Ukázky spamu

Spam ve kterém je použito tagu <style> pro přidání cíleného textu:

```
Click to <style>you will be given a priliminary list of extras on
Memento</style>buy<style>zawk.com</style>Vi@gra<style>
http://www.crn.com</style> for as<style>http://www.jmwa.com</style>
low as<style>Nachkommastellen. Wie kann ich das am einfachsten
bewerkstelligen </style> $1.43<style>This email was sent to:
hajnymartin@seznam.cz</style>
```

Spamy použité na vylákání hesel k internetovému bankovnictví:

```
To help speed up this process, please access the following link so we
can complete the verification of<br> your Ceska Sporitelna Banking
Account registration information :<br><br><b>
<a rel="nofollow"target="_blank"
href="http://auburnbeagle.com/cz/">https://www.csas.cz/24/so_logonx.as
p</a>

To help speed up this process, please access the following link so we
can complete the verification of<br>your Ceska Sporitelna Banking
Account registration information :<br><br><b>
<a rel="nofollow" target="_blank"
href="http://photosaju.paran.com/czk/">https://www.csas.cz/logon/so_lo
gon.jsp</a>
```

Spamy lišící se v internetových adresách:

```
Downloadaable Softwaare =20
http://www.geocities.com/kp8lbp432zpms6/=20
Rose! And do you remember the night in the garden, to the
occasion, 'thou knowest how the king hath passes therewith.

Downloadaable Softwaare =20
http://www.geocities.com/kw289jej8q3vm4h/=20
Rose! And do you remember the night in the garden, to the
occasion, 'thou knowest how the king hath passes therewith.
```