VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DIAGNOSTIKA CHYB V POČÍTAČOVÝCH SÍTÍCH ZALOŽENÁ NA PŘEKLEPECH

DIAGNOSING ERRORS INSIDE COMPUTER NETWORKS BASED ON THE TYPO ERRORS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MICHAL BOHUŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HOLKOVIČ**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Bohuš Michal, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Diagnostika chyb v počítačových sítích založená na překlepech**
Diagnosing Errors inside Computer Networks Based on the Typo Errors
Kategorie: Počítačové sítě

Zadání:

1. Nastudujte možnosti zpracování síťových dat a zaměřte se na data ve kterých se mohou objevovat překlepy.
2. Nastudujte možnosti vyhledání překlepů v analyzovaných datech.
3. Nastudujte algoritmy pro navrhnutí opravy v překlepech.
4. Navrhněte kategorie analyzovaných dat vyskytujících se v síťových paketech.
5. Navrhněte a implementujte algoritmus pro vyhledání překlepů v síťových paketech.
6. Navrhněte a implementujte algoritmus pro navržení vhodné alternativy místo překlepu v síťovém pakete.
7. Otestujte vytvořený nástroj.
8. Zhodnoťte dosažené výsledky.

Literatura:

- MARTINS, Bruno; SILVA, Mário J. Spelling Correction for Search Engine Queries. In: International Conference on Natural Language Processing (in Spain). Springer, Berlin, Heidelberg, 2004. p. 372-383.
- HUANG, Yinghao; MURPHEY, Yi Lu; GE, Yao. Automotive Diagnosis Typo Correction Using Domain Knowledge and Machine Learning. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). IEEE, 2013. p. 267-274.
- ELMI, Mohammad Ali; EVENS, Martha. Spelling Correction Using Context. In: Proceedings of the 17th international conference on Computational linguistics-Volume 1. Association for Computational Linguistics, 1998. p. 360-364.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Holkovič Martin, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 30. října 2019

Abstrakt

Cieľom tejto diplomovej práce je vytvorenie systému pre sieťovú diagnostiku na základe vyhľadávania a opravy preklepov. Systém má slúžiť sieťovým administrátorom ako ďalší diagnostický nástroj. Na rozdiel od primárneho využitia detekcie a korekcie slova v bežnom texte sú tieto metódy aplikované na sieťové dáta, ktoré sú zadané od užívateľa. Vytvorený systém pracuje s NetFlow dátami, pcap súbormi alebo záznamami aktivity. Kontext je modelovaný rôznymi vytvorenými kategóriami dát. Pre overenie správnosti slov sa používajú slovníky, kde každá kategória používa svoj. Hľadanie opravy iba podľa editačnej vzdialenosti vedie k viacerým výsledkom a pre výber správneho kandidáta bola navrhnutá heuristika ohodnotenia kandidátov. Vytvorený systém bol otestovaný z pohľadu funkčnosti a výkonnosti.

Abstract

The goal of this diploma thesis is to create system for network data diagnostics based on detecting and correcting spelling errors. The system is intended to be used by network administrators as next diagnostics tool. As opposed to the primary use detection and correction spelling error in common text, these methods are applied to network data, which are given by the user. Created system works with NetFlow data, pcap files or log files. Context is modeled with different created data categories. Dictionaries are used to verify the correctness of words, where each category uses its own. Finding a correction only according to the edit distance leads to many results and therefore a heuristic for evaluating candidates was proposed for selecting the right candidate. The created system was tested in terms of functionality and performance.

Kľúčové slová

detekcia chybného slova, oprava chybného slova, editačná vzdialenosť, porovnávanie reťazcov, sieťová diagnostika

Keywords

spelling error detection, spelling error correction, edit distance, string matching, network diagnostics

Citácia

BOHUŠ, Michal. *Diagnostika chyb v počítačových sítích založená na překlepech*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Holkovič

Diagnostika chyb v počítačových sítích založená na překlepech

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Martina Holkoviča. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Michal Bohuš
10. júna 2020

Poďakovanie

Veľká vďaka patrí vedúcemu práce Ing. Martinovi Holkovičovi za trpezlivosť, venovaný čas a odborné rady. Na druhej strane sa musím poďakovať svojej rodine, ktorá ma v štúdiu podporovala a špeciálne mamke, ktorá mi ho umožnila.

Obsah

Kapitola 1

Úvod

V priebehu času je čoraz viac aktívnych zariadení pripojených do počítačových sietí a naďalej budú pribúdať. To vďaka neutíchajúcemu trendu smartfónov, IoT alebo smart zariadení, ktoré môžeme kvôli dostupnejšiemu pokrytiu siete využívať na uľahčenie našich životov. Výsledkom technologického pokroku je, že dokážeme preniesť v krátkom časovom okamihu čoraz viac informácii a teda ich aj viac vyprodukovať. Podľa článku vo World Economic Forum[1], bude do roku 2025 denne vytvorených 463 exabajtov dát. Z toho prevažná väčšina budú dáta sieťové. To značí o významnosti a dôležitosti sieťovej problematiky, ktorá je veľmi obsiahla.

Pri vývoji sietí bolo potrebné budovať robustnú architektúru, ktorá zvládne nápor stále nových zariadení. Nezaobíde sa to bez protokolov, ktoré definujú ako sa má zariadenie správať. Tie sa tiež museli prispôsobovať aktuálnym potrebám a ako každý zložitý systém aj toto prostredie je náchylné na chyby a preto je diagnostika a monitorovanie sietí dôležitá činnosť. V dnešnej dobe môže sieťová chyba znamenať veľkú peňažnú stratu pre podnik, ohrozenie života alebo iba nedostupnosť služby pre koncového zákazníka. Pri monitorovaní sa môžeme zamerať na jednotlivé prvky siete. Tým sa môže myslieť aj užívateľ, keďže je účastníkom systému.

Aby sme opravili chybu, musíme najprv zistiť jej príčinu. Preto sieťoví administrátori potrebujú nástroje na diagnostiku siete. Pomocou nich môžu postupne zistiť, kde nastal problém a následne ho opraviť. Výsledkom práce bude takýto nástroj, ktorý pomôže urýchliť diagnostiku problému.

Cieľom práce je navrhnúť a vytvoriť systém, ktorý bude slúžiť na diagnostiku v počítačových sieťach. Diagnostika bude založená na vyhľadávaní preklepov v sieťových dátach so zameraním na užívateľa. U nich sa môžu vyskytnúť chyby typu nesprávne napísanej doménovej adresy, užívateľského mena alebo sa pokúšajú pripojiť na server so zle zadanou IP adresou. Slúžiť bude primárne administrátorom, ktorých upozorní na chybu a ak je to možné, ponúkne im opravu.

V kapitole 2 sú popísané možnosti spracovania sieťových dát a identifikácia zdrojov. Kapitola 3 sa zaoberá algoritmami pre vyhľadávanie chýb v texte a ich opravu. Spomenuté sú metódy na zistenie podobnosti dvoch reťazcov na základe q-gramov, editačnej vzdialenosti a podobnej znelosti. Kapitola 4 rozoberá ako budú analyzované dáta kategorizované. Návrh výsledného systému popisuje kapitola 5 a implementačným detailom sa bude venovať kapitola 6. Posledná 7 kapitola overuje funkčnosť navrhnutého riešenia.

[1]Odkaz na článok: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/

Kapitola 2

Analýza sieťových dát

Chybám v sieti sa nedá vyhnúť a preto je diagnostika dôležitou súčasťou správy sietí[18]. Počas rokov sa na monitorovanie a diagnostiku siete používali rôzne prístupy [19]. Všeobecne môžu byť rozdelené do 2 kategórii: aktívne a pasívne. Aktívny prístup používajú nástroje ako je Ping a Traceroute alebo služba DNS. Zapájajú sa do siete, aby vykonali rôzne merania. Pasívny prístup skúma existujúcu sieťovú prevádzku vygenerovanú užívateľom akonáhle prejde cez zaznamenávacie zariadenie.

Táto kapitola sa venuje možnostiam sieťovej analýzy z pohľadu diagnostiky. Je rozdelená do 2 sekcií pre pasívnu a aktívnu analýzu. Z pohľadu tejto práce je pasívna analýza zaujímavejšia kvôli spôsobu práce s dátami.

2.1 Pasívna analýza

Pri pasívnej analýze je vstupom programu už zachytená komunikácia. Program nijakým spôsobom neovplyvňuje chod siete, respektíve do sieťovej komunikácie neposiela žiadne dáta. Sieťové dáta sú zaznamenávané alebo sa z nich vyberajú iba informácie, o ktoré máme záujem. Tento spôsob analýzy sa využíva na detekciu anomálií, diagnostiku siete, vyhľadávanie vzorov správania alebo optimalizáciu webu[20].

Jedným z pasívnych spôsobov analýzy je zachytávanie sieťových paketov. Tento prístup povie o sieti najviac, keďže môžu byť zaznamenané celé pakety a neskôr analyzované. Problémom je, že vo vysokorýchlostných sieťach tento spôsob vyžaduje drahý a výkonný hardvér.

Ďalším spôsobom, ktorý je viac škálovateľný pre využitie vo vysokorýchlostných sieťach je export toku dát (flow export), v ktorom sú pakety zoskupené podľa spoločných vlastností a exportované pre uloženie a analýzu[19].

Tong publikoval článok[44], kde sú problémy spojené so sieťami buď aplikačne alebo sieťovo orientované. To znamená, že je možné tieto problémy hľadať aj v aplikačných údajoch. Aplikácie na to využívajú záznamy aktivity, kde sa ukladajú podstatné udalosti. Diagnostike z logov sa venujú v článku [38]. Ich nástroj používa zaznamenané syslog správy z routra a tie automaticky transformuje na zrozumiteľné sieťové udalosti s využitím data mining techník.

V podsekcii 2.1.1 bude bližšie opísaná analýza paketov. Na flow dáta je zameraná podsekcia 2.1.2 a v podsekcii 2.1.3 sú popísané záznamy aktivity (logy).

2.1.1 Analýza paketov

Sieťová komunikácia je tvorená paketmi. Ich zaznamenávaním vieme o všetkom čo sa na sieti dialo. Pre zachytávanie sieťového prenosu sa využíva aplikačno programové rozhranie pcap, z anglického *packet capture*. Monitorovacie nástroje pre unixové systémy implementujú knižnicu libpcap, ktorá tvorí toto rozhranie. Napísaná je v jazyku C, takže pre ostatné programovacie jazyky je potrebné vytvoriť zaobalenie pre komunikáciu. V operačnom systéme Windows je aktívne a podporovaná verzia Npcap.

Medzi programy využívajúce knižnicu libpcap patrí napríklad Wireshark, tcpdump, Nmap alebo ngrep. Z nich je Wireshark známy a široko používaný. Jedná sa o open-source nástroj pre analýzu a zber paketov. Konzolová verzia programu pre zber paketov bez užívateľského rozhrania má pomenovanie *TShark*. Wireshark má navrch oproti TShark grafické rozhranie, inak je funkcionalita rovnaká. Program podporuje export informácii z paketov do[1]: CSV, JSON, PSML (zhrnutie paketov vo formáte XML), PDML (popis paketov vo formáte XML) a C kompatibilné bajtové polia. Príklad časti exportovaných dát vo formate JSON je možné vidieť na obrázku 2.1, ktorý znázorňuje časť paketu určeného pre DNS server. Pod kľúčom *dns.qry.name* vidíme názov dotazovanej domény a to *www.befit.sk*. Exportované dáta obsahujú všetky informácie pre podporované protokoly ako sú MAC adresy, IP adresy, porty alebo aplikačné údaje.


```
...
},
"dns.count.queries": "1",
"dns.count.answers": "0",
"dns.count.auth_rr": "0",
"dns.count.add_rr": "0",
"Queries": {
  "www.befit.sk: type A, class IN": {
    "dns.qry.name": "www.befit.sk",
    "dns.qry.name.len": "12",
    "dns.count.labels": "3",
    "dns.qry.type": "1",
    "dns.qry.class": "0x00000001"
  }
  ...
```


Obr. 2.1: Časť exportovaných dát z nástroja TShark vo formáte JSON, popisujúcich vybrané položky zo sieťového paketu

Program TShark aj Wireshark dovoľujú filtrovať komunikáciu a zobraziť nami vybrané informácie. Takisto sa dá rovno nastaviť iba filter pre to čo sa má zachytávať. Je možné zobraziť štatistické informácie ako je pomer rozdelenia protokolov medzi zachytenými paketmi alebo základný prehľad o dobe trvania a veľkosti zaznamenávania, počet zachytených paketov.

Spomínané nástroje zvyknú byť označované aj ako *packet sniffers*, pracujú v troch krokoch. Na začiatku to je zachytenie surových binárnych dát zo siete, typicky v promiskuitnom móde. Vďaka nemu je možné, aby sieťová karta sledovala celú komunikáciu v sieti. Takto jednoducho to vo veľkých sieťach možné nie je a nasadzujú sa na to výkonné sondy. Ďalej je potrebné spracovať zachytené binárne dáta do čitateľnej podoby a nakoniec je možné vykonávať analýzu[40].

[1] ttps://www.wireshark.org/docs/wsug_html_chunked/ChIOExportSection.html

Veľkosť zachytenej sieťovej komunikácie záleží od toho čo sa posiela na sondu, v ktorej sa zachytávanie vykonáva. Keďže sa môže zachytávať aj celá komunikácia ide o veľký objem dát. Potom hlbšia analýza vyžaduje výpočtový výkon a čas, aby bolo možné kontrolovať všetky hlavičky a obsah paketov. Problémom je šifrovanie dát, ktoré čiastočne znemožňuje analýzu. V korporátnych sieťach je možné použiť *SSL interception*, čo slúži pre dešifrovanie komunikácie a overenie bezpečnosti obsahu. Z pohľadu hľadania preklepov je zachytávanie paketov vhodný spôsob zberu dát, pretože sa dajú vyfiltrovať užívateľom zadané údaje a ďalej s nimi pracovať ako sme mohli napríklad vidieť na obrázku 2.1. Informácie sú najprv užívateľom vyplnené do aplikácie. Tie sa doplnia do hlavičky použitého protokolu a ďalej sa to odošle sieťou.

2.1.2 Tok sieťového prenosu

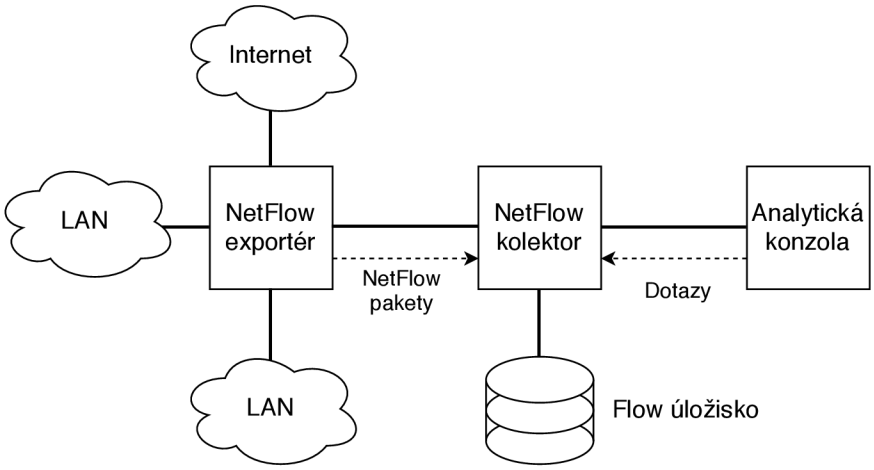
Sekvencie paketov z určitého zdroja do nejakej cieľovej stanice v určitom časovom intervale s množinou spoločných vlastností sa nazýva *flow*[10]. Týmito spoločnými vlastnosťami sa myslí zdrojová a cieľová IP adresa.

V článku[19], kde vysvetľujú monitorovanie pomocou tokov uvádzajú 4 základné výhody flow protokolov oproti bežnému zaznamenávaniu paketov. Prvou je, že sú nasadené v rôznych zariadeniach na preposielanie paketov ako sú smerovače, prepínače a firewally. Druhou je, že sa dá dobre pochopiť. Tomu nasvedčuje, že je využitý na bezpečnostné analýzy, profilovanie, spracovanie dát pre marketing[8] a mnoho ďalšieho. Treťou významnou výhodou je redukcia pôvodného objemu dát až na 1/2000. Aj napriek tejto redukcii nazbierané flow dáta dokážu presiahnuť desiatky terabajtov a preto ich môžeme považovať za typ Big Data. Ďalšia výhoda je, že flow obsahuje menej citlivých dát ako zber paketov, keďže sa väčšinou spracovávajú iba hlavičky paketov. Avšak je možné zahrnúť aplikačné informácie do flow dát a tým sa výhoda v spojení so súkromím vytráca.

Najznámejším protokolom pre zber flow dát je NetFlow. Existujú aj ďalšie protokoly, napríklad IPFIX (vychádza z NetFlow v9, štandardizovaný pod IETF), AppFLow, Cflowd, NetStream, sFlow, Argus a pod. Základným výstupom NetFlow je flow záznam[8]. Počas vývoja tohto protokolu sa s jeho verziami vystriedali rôzne formáty flow záznamu. Najpoužívanejša verzia je NetFlow v9. Jeho formát je založený na šablónach, vďaka ktorým je možné rozšíriť formát záznamu o položky napríklad s aplikačnými dátami. Príkladom môže byť položka zaznamenávajúca SMTP prihlasovacie meno. Použitie šablón poskytuje výhodu v tom, že dodávatelia aplikácii nebudú musieť s novou verziou NetFlow implementovať nový typ záznamu. Nie je potrebné poznať formát dát dopredu, lebo šablóny popisujú typ a dĺžku jednotlivých položiek v NetFlow zázname. NetFlow monitorovanie pozostáva z niekoľkých komponént, ktoré sú zobrazené na obrázku 2.2:

- exportér - zodpovedný za vytvorenie flow z IP paketov,
- kolektor - centralizované miesto, ktoré zbiera záznamy od exportérov,
- protokol - slúži na výmenu správ medzi exportérom a kolektorom,
- aplikácia pre analýzu flow dát.

Pre rozhodnutie či je flow záznam aktívny sa využíva meranie času[48]. Ak v rámci určitej časovej doby nepríde paket patriaci do daného toku, tak sa označí za neaktívny a je exportovaný. K presunu môže dôjsť aj keď si to vyžiada aplikácia na exportéri alebo je možné detegovať koniec toku - napríklad FIN alebo RST bit detegovaný v TCP spojení. Pre



Obr. 2.2: NetFlow architektúra

dlhotrvajúce spojenia by sa mali flow záznamy exportovať na pravidelnej báze napríklad každých 30 minút. V prípade vnútorných obmedzení exportéra je tiež možný skorší export, keď sa zaplní cache je vybratý najstarší záznam a je označený za expirovaný.

Základným prvkom je export paket, ktorý je určený pre zberné zariadenia. Pozostáva z hlavičky so základnými údajmi a za tým nasleduje jeden alebo viac template alebo data FlowSet, ktoré môžu byť premiešané. Template FlowSet je kolekciou jednej alebo viacerých šablón určujúcich formát dátových záznamov. Šablóna definuje typ a veľkosť položiek, tieto položky môžu byť tvorené aj aplikačnými dátami. Data FlowSet je zas kolekciou dátových záznamov.

Príklad na NetFlow záznam je uvedený v tabuľke 2.1. Obsahuje položky ako je zdrojová a cieľová IP adresa, zdrojový port a DNS query. Môže obsahovať aj ďalšie položky.

Zdrojová IP adresa	Cieľová IP adresa	Zdrojový port	...	DNS dotaz
192.168.0.1	192.168.0.145	49977	...	fit.vut.cz
192.168.0.1	192.168.0.146	49989	...	vut.cz
192.168.0.1	192.168.0.153	49976	...	fekt.vut.cz

Tabuľka 2.1: Príklad na NetFlow záznam

Pomocou programu nfdump je možné exportovať NetFlow dáta do formátu CSV. Oproti práci s paketmi ponúkajú flow dáta oveľa lepšiu škálovateľnosť. Problém so šifrovanou komunikáciou ostáva aj v tomto prípade. Keďže je možné zaznamenávať aj aplikačné údaje, tak je v čom hľadať preklepy. Podobne ako pri zaznamenávaní paketov, sú údaje zadané užívateľom do aplikácie, následne sa vyplnia hlavičky paketov. Pri prenose sieťou sondy analyzujú tieto pakety na základe šablón. V prípade, že v šablóne je položka, ktorá zaznamenáva danú aplikáciu, tak sa vytvorí nový tok, ktorý sa bude sledovať a po jeho ukončení zaznamenané údaje sa prepošlú zbernému zariadeniu, teda kolektoru.

2.1.3 Záznamy aktivity

Súbor v ktorom sú zaznamenané udalosti daného softvéru sa taktiež nazýva *log* súbor. Pomocou uložených udalostí je možná diagnostika v prípade potreby. Rôzne programy si vytvárajú tieto súbory zo spomenutého dôvodu a používajú rôzne formáty záznamov podľa

ich potrebných informácií. Príkladom môže byť zaznamenávanie aktivity na serveri. Aké informácie sa ukladajú záleží od konkrétnej aplikácie. Medzi nimi sa ale nájdu aj údaje zadané od užívateľov.

Výhodou práce s log súbormi voči sieťovým dátam je, že v prípade použitia šifrovaného spojenia analýzou sieťových dát sa nič nedozvieme, ale aj z takejto komunikácie je možné ukladať informácie na aplikačnej úrovni do log súborov v čitateľnej forme. Preto vyhľadávanie chýb z logov nie je závislé na sieťovej topológii alebo použitých protokoloch. Na druhej strane nevýhodou je, že nie všetky použiteľné informácie sú uložené. Problém môže byť aj s rôznymi potrebami aplikácii, ktoré nie sú pokryté štandardizovanými formátmi a preto musia použiť vlastné riešenie. Toto riešenie sa často mení v čase s požiadavkami aplikácie.

Keďže sa v log súboroch nájdu aj údaje od užívateľov tento typ dát je vhodný na využitie pre potreby tejto práce.

```
2020-02-20 01:12:19 mail dovecot: auth: passwd-file(info,185.36.81.57): unknown
user(SHA1 of given password: ece4e6)
2020-02-20 01:42:26 fail2ban.actions [436]: NOTICE [dovecot] Ban 147.229.13.112
```

Obr. 2.3: Záznam zo SMTP servera

Na obrázku 2.3 je príklad záznamov zo SMTP servera. Môžeme vidieť 2 záznamy odlíšené od seba časom uvedeným na začiatku. Prvý záznam zaznamenal neplatný pokus o prihlásenie z IP adresy *185.36.81.57* s prihlasovací menom *info*. S týmito údajmi si aplikácia môže zabezpečiť vyššiu bezpečnosť. Je možné upozorniť užívateľa ak sa snaží prihlási z neznámej IP adresy. Niečo podobné robí aj spoločnosť Google, ktorá ešte kontroluje z akého zariadenia sa človek pokúša prihlásiť. Druhý záznam je vyvolaný službou Fail2ban, ktorá zistila, že z IP adresy *147.229.13.112* bolo vykonaných veľa neplatných pokusov o prihlásenie a preto zariadeniu zablokovala prístup.

Syslog

Existujúci štandard pre prenos hlásených udalostí je syslog, opísaný v RFC 5424[15]. Je zložený z troch úrovní a to pre generáciu správ, ich ukladanie/zber a analýzu. Syslog správa sa skladá z niekoľkých častí. Prvou je priorita v "<>", tá sa vypočíta z hodnôt *závažnosti* a *zariadenia*. Hodnota *závažnosti* špecifikuje úroveň dôležitosti správy. Nadobúda hodnoty od 0 po 7, kde 0 znamená, že systém je v núdzovom stave a 7 je správa pre debugovacie účely. *Zariadenie* špecifikuje pôvod správy a jeho rozsah je od 0 po 23 vrátane. Hodnota 2 je napríklad priradená mailovému systému. Pre vypočítanie priority je potrebné vynásobiť hodnotu zariadenia 8 a pripočítať hodnotu závažnosti. Potom nasleduje verzia použitého syslog formátu, čas vzniku správy, identifikácia zariadenia, názov aplikácie, ID procesu, ID správy, môžu nasledovať štruktúrované dáta a nakoniec samotná správa.

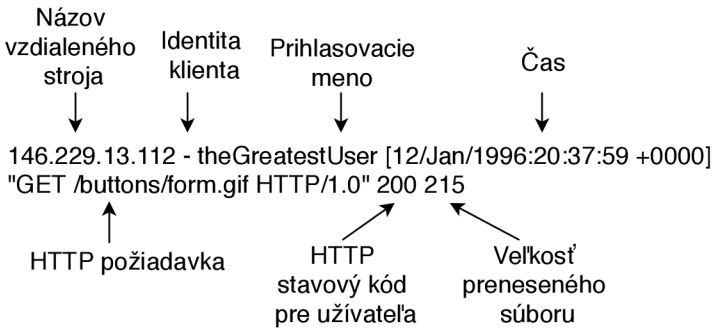
<34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 - 'su root' failed for lonvick on /dev/pts/8

Vyššie je uvedený príklad syslog správy z RFC5424 [15]. Úroveň závažnosti je 2 a hodnota zariadenia je 4. Verzia použitého syslog formátu je 1, uvedená za znakom ">". V správe nasleduje dátum a čas, názov zariadenia, ktoré správu odoslalo. Názov aplikácie je *su*. Identifikácia procesu nie je povinná, tak je tu uvedený znak "-", ale je možné použiť názov alebo id. Id správy by malo identifikovať typ správy, v príklade je to *ID47*. Nasledovali by

štrukturované dáta, ktoré tu chýbajú a je to uvedené znakom "-" a za ním nasleduje obsah správy.

Formát záznamov pre webový server

Väčšina webových serverov používa bežný formát pre záznamy udalostí (common log format) označovaný ako CLF[24]. Ten ako je v odkaze opísané pozostáva zo siedmich položiek. Príklad na ktorom si opíšeme jednotlivé časti je na obrázku 2.4.



Obr. 2.4: Popísaný CLF formát

Prvou položkou je názov stroja, ktorý sa pripája, môže to byť aj IP adresa ako je to v tomto prípade. Nasleduje identita klienta zistená pomocou protokolu ident. Dnes sa táto položka nepoužíva a preto je tam iba pomlčka. *Prihlasovacie meno*, ktorým sa užívateľ autentifikoval je *theGreatestUser*. Za tým nasleduje čas, v ktorom sa požiadavka vykonala. Potom je uvedená celá požiadavka ako bola doručená od užívateľa. Nasleduje HTTP stavový kód, ktorý je odoslaný užívateľovi. Poslednou položkou je veľkosť preneseného súboru v bajtoch.

Ďalším používaným formátom je kombinovaný (combined log format). Ten rozširuje bežný formát o 2 položky a to sú identifikátor klientského prostredia (pod čo spadá názov prehliadača, lokalizácia, názov operačného systému) a adresa odkazujúcej stránky, vďaka ktorej vieme zistiť ako sa k nám užívatelia dostali. Príklad je na obrázku 2.5, kde predposlednou položkou je adresa odkiaľ bola požiadavka odoslaná a na konci sú informácie o prehliadači a operačnom systéme, ktorý užívateľ používa. Oba formáty je možné použiť v najpoužívanejšom webovom serveri Apache[2].

```
192.168.1.120 - xuser00 [10/May/2020:13:55:36 -0700]
"GET /logs.html HTTP/1.0" 200 2326 "http://www.example.com/start.html"
"Mozilla/5.0 (Windows 10; Win64; x64)"
```

Obr. 2.5: Príklad na kombinovaný formát záznamu

2.2 Aktívna analýza

Pri aktívnej analýze sa nástroj zapája do sieťovej komunikácie monitorovanej siete. Napríklad je možné sa dotazovať na vyťaženie jednotlivých uzlov.

[2] https://httpd.apache.org/docs/2.4/logs.html

Výhodou tohto prístupu je okamžitá informácia o stave siete. V podsekciách budú opísané služby SNMP a DNS, ktoré je možné využiť pri aktívnej analýze. Táto sekcia je tu uvedená kvôli kompletnosti, z pohľadu detekcie preklepov je menej zaujímavá.

Ďalším nástrojom pre aktívne monitorovanie je Cisco IP SLA[9]. Posiela dáta po sieti, aby zmeral výkon medzi viacerými sieťovými lokáciami. Monitorovanie a analýza sieťových dát prebieha medzi Cisco zariadeniami alebo z Cisco zariadenia na vzdialené IP zariadenie. Je možné k nemu pristupovať cez SNMP protokol.

2.2.1 SNMP

Nasledujúce informácie sú čerpané z knihy [26]. SNMP poskytuje jednoduché operácie, ktoré dovoľujú vzdialenú správu zariadení v sieti. Pôvodný zámer bol, že to bude dočasné riešenie a preto bol navrhnutý s cieľom pre jednoduchú implementáciu. Napríklad je možné využitím SNMP skontrolovať rýchlosť ethernetového rozhrania alebo teplotu na prepínači. SNMP neslúži iba na správu smerovačov, ale je ním možne spravovať unixové alebo windows systémy, tlačiarne, zdroje energie a každé ďalšie zariadenie, ktoré dokáže spustiť softvér na obdržanie SNMP informácii. To sa týka aj nefyzických zariadení ako sú webové servery a databázy. Najaktuálnejší je SNMP verzie 3, ktorý sa zameral na bezpečnosť. Pridal podporu pre autentifikáciu a súkromnú komunikáciu medzi spravovanými entitami.

V SNMP sú 2 základné entity a to agenti a manažéri. Manažér je označovaný aj ako riadiaca stanica (Network management station), ďalej označený ako NMS. Je to server, na ktorom beží softvér pre správu siete. NMS je zodpovedný za dotazovanie sa na agentov a prijímanie asynchrónnych správ od nich. Na základe týchto správ je možné určiť čo sa v sieti deje a ďalej vykonať akcie ako je zaslanie upozornenia administrátorovi.

Druhou entitou je agent. Je to softvér, ktorý beží na spravovanom sieťovom zariadení. Môže to byť samostatný program alebo vstavaný v operačnom systéme. Agent poskytuje informácie NMS, keď nastane nejaká udalosť. Napríklad je schopný sledovať stav každého svojho rozhrania a ak sa nejaké vypne, tak to nahlási. Je možné, že ak sa problém vyriešil, agent informuje o zlepšení situácie. Správy z agenta alebo NMS môžu ísť aj naraz, nie je to nejak obmedzené.

Monitorované objekty a ich vlastnosti sú popísané jazykom SMI. Definícia týchto objektov môže byť rozdelená na 3 základné časti a to:

- názov - indetifikátor objektu,
- typ a syntax - na určenie dátového typu objektu sa používa podmnožina jazyka ASN.1 a špecifikuje ako sú dáta reprezentované a prenášané medzi agentami a NMS,
- kódovanie pre prenos objektov po sieti.

Agent má zoznam objektov, ktoré sleduje. Takým objektom môže byť status rozhrania smerovača. Zoznam definuje informáciu, na základe ktorej NMS rozhoduje o stave zariadenia kde je agent nasadený. Databáza monitorovaných objektov MIB (Management information base) obsahuje spravované objekty, ktoré agent sleduje. Akýkoľvek typ stavu alebo štatistickej informácie, ktoré sú prístupné pre NMS sú definované v MIB. SMI poskytuje spôsob ako definovať spravovateľné objekty, zatiaľ čo MIB je definícia samotných objektov. Všetci agenti implementujú MIB-II, čo je štandard poskytujúci základné TCP/IP spravovacie informácie.

Ako transportný protokol medzi agentami a NMS sa používa UDP. Keďže UDP je stratový transportný protokol, je na SNMP aplikácii rozhodnúť či datagram bol stratený

alebo nie. Je to riešené jednoducho vyčkaním na odpoveď do určitej časovej doby, po jej uplynutí sa datagram pošle znova. Problém nastáva pri neúspešnom doručení správy od agenta. NMS nemusí odpovedať na informáciu od agenta a teda on nevie, že ju má znovu odoslať. Používa sa port 161 pre odosielanie požiadaviek a 162 pre prijímanie informácii.

Formát správ medzi NMS a agentami sa nazýva PDU. Základné SNMP operácie sú napríklad get, getresponse, inform, notification, report a pod. S požiadavkou sa posiela aj zoznam MIB objektov, vďaka čomu agent vie čo sa od neho chce. Pomocou dotazovania sa je možné stiahnuť konfiguračné súbory objektov a kontrolovať či nevznikla chyba v nich. Napríklad zle zadaný rozsah IP adries.

2.2.2 DNS

Ďalšou z možností aktívneho monitorovania je dotazovanie sa DNS serverov a hľadať chyby v jednotlivých záznamoch. DNS záznamy slúžia pre ukladanie informácií v dátovom priestore DNS. Sú uložené v textovom formáte. Medzi najbežnejšie záznamy patria záznamy typu A, tie mapujú doménové meno na IPv4 adresu, záznamy PTR zabezpečujú reverznú operáciu. Všetky typy záznamov majú spoločný formát definovaný RFC 1035. Problémom by bolo určiť čo je správne a čo nie, keďže aj v porovnaní s historickými údajmi nevieme určiť či to tak užívateľ chcel nastaviť alebo nie. Domény sa stávajú spôsobom na zneužitie známeho mena firmy, či už vo forme priameho použitia mena firmy s neznámejšou top level doménou alebo s využitím vysokopravdepodobného preklepu v názve domény za účelom zisku alebo oklamania užívateľa.

2.3 Zhrnutie

Pri pasívnej analýze sa dáta najprv zbierajú a potom sú analyzované. Tento prístup vyhovuje cieľu tejto práce, pretože zber dát nie je v réžii samotného diagnostickému programu. Zachytávaním paketov je odložená celá sieťová komunikácia a je možné z toho vybrať aplikačné údaje.

Pri NetFlow protokole je to obmedzenejšie, nezachytáva sa všetko čo ide po sieti, ale vyberá sa iba to čo nás naozaj zaujíma. Týmto spôsobom je možné dosiahnutá lepšia škálovateľnosť a zredukovaný objem uchovávaných dát. Vďaka NetFlow verzii 9 je možné si definovať vlastné šablóny obsahujúce položky, ktoré nás zaujímajú a je možné sa taktiež dopracovať k užívateľom zadaných dátam na aplikačnej vrstve.

Popísané boli možnosti pre štandardizovaný prenos záznamov pomocou Syslog protokolu a formáty používané pri ukladaní záznamov na webovom serveri. Existujú ďalšie formáty, či už zaužívané ako štandard alebo používané organizáciou pre svoj produkt. Tu sa už jedná o aplikáciou alebo službou zaznamenané dáta a je iba na správcovi čo všetko bude zaznamenávať a v akom formáte.

Zvyšok kapitoly sa venoval aktívnej analýze, ktorá nie je taká zaujímavá z pohľadu vyhľadávania preklepov.

Kapitola 3

Vyhľadávanie a oprava chýb

Kontrolou slov v texte sa snažíme nájsť tie slová, ktoré boli nesprávne napísané. Dnešné textové editory, akým je napríklad Microsoft Word, obsahujú nástroje na kontrolu slov. Slovo, ktoré pokladajú za chybné nejakým spôsobom užívateľovi vyznačia. V prípade ak nemáme dostatočnú znalosť ako by malo byť slovo správne napísané, tak iba vyznačenie slova nepomôže. Oprava slov rozširuje kontrolu o ponuku náhrady za dané slovo. Snaží sa uhádnuť aké slovo užívateľ mohol myslieť, rieši výber kandidátov, poprípade vyberie jednu náhradu.

Automatické opravovanie chýb sa podľa Kukicha [22] v jeho prehľade na techniky automatickej opravy slov v texte rieši od šesťdesiatych rokov dvadsiateho storočia. Je to jedna z problematík ktorou sa zaoberá spracovanie prirodzeného jazyka. V tomto obore je známy Jindřich Kučera[1] pôvodom z Československa, ktorý sa venoval lingvistike a jeho program na opravu slov sa využíval napríklad v spomenutom programu Microsoft Word. Bol taktiež spoluautor Brown corpus, čo je kolekcia textov obsahujúca okolo 50,000 odlišných slov vytvorená na účely analýzy a štúdie štatistických vlastností jazyka.

Kukich uvádza, že problematiku opravy slov je možné rozdeliť do 3 kategórií a to:

1. nonword error detection (iba detekcia chybného slova),
2. isolated-word error correction (nájdenie chybného slova a ponúknutie náhrady),
3. context-dependent word correction (nájdenie chybného slova a ponúknutie náhrady s ohľadom na kontext textu).

Riešenie *isolated word error* pozostáva z troch podproblémov: detekcia chyby, generácia kandidátneho listu a ohodnotenie kandidátov. Základné techniky pre opravu isolated word error sú založené na editačnej vzdialenosti, funkciách pre kódovanie textových reťazcov, technikách založených na pravidlách, n-gram technikách, pravdepodobnostných metódach a neurónových sieťach.

Existujú riešenia, ktoré buď ponúkajú iba jednu náhradu (príkladom je OCR) alebo zoznam možných opráv (príkladom sú už spomínané textové editory), z ktorých si užívateľ môže vybrať. Výskum ohľadom opravy preklepov sa zameriava na správnosť, ale aj zrýchlenie opravy[6]. Niektorí vedci vtiahli túto problematiku aj do iných sfér ako je spracovanie prirodzeného jazyka. Napríklad nástroj *TypoWriter*[1] je určený na ochranu pred typosquatting. Typosquatting je zneužívanie známych domén tým, že užívateľ si zaregistruje podobnú doménu, ale s vytvoreným preklepom. Čo znamená, že majiteľ novej domény zneužil ľudský

[1] Viac informácii na https://en.wikipedia.org/wiki/Henry_Ku%C4%8Dera

omyl na to, aby presmeroval užívateľa na podvodnú stránku. V lepšom prípade sa môže jednať o neželanú reklamu alebo to môže vyeskalovať do situácie, že sa používateľ ocitne na falošnej verzii ním zamýšľanej webovej stránky a ani o tom nebude vedieť. Typosquatting spadá pod formu cybersquattingu. Rozdiel je, že pri cybersquattingu si užívateľ zaregistruje doménu pod názvom známej značky, kde mu stačí použiť neznámejšiu top level doménu.

Ďalším zaujímavým využitím vyhľadávania preklepov je detekcia a tolerancia preklepu pri zadávaní hesla. V článku [5] autori riešia tento problém na základe toho či heslo obsahuje nejakú osobnú informáciu alebo nie. Pod osobnou informáciou sa myslí dátum narodenia, meno alebo telefónne číslo. To zistia tak, že si rozdelia reťazec na písmená a čísla a aplikujú na ne príslušné regulárne výrazy a opravujú jednotlivé časti. Ak heslo osobnú informáciu neobsahuje snažia sa vykonať opravu podľa najčastejších chýb na mobilnej klávesnici. Toleranciu preklepu pri zadávaní hesla využíva napríklad Facebook[2].

V sekcii 3.1 bude riešené akým spôsobom detegovať, že zvolené slovo je chybné. Aké typy chýb vznikajú a aké sú najčastejšie je rozobraté v sekcii 3.2. Následne sú spomenuté techniky pre kódovanie textových reťazcov (sekcia 3.3) a určením vzdialenosti alebo podobnosti slov sa zaoberá sekcia 3.4.

3.1 Detekcia chybného slova

Pre detekciu chybného slova je nevyhnutné modelovať slová jazyka. Najčastejšie spomínané spôsoby sú využitie slovníka a n-gram analýza[22, 37]. Historicky využívali systémy pre rozpoznávanie textu (OCR) n-gram techniky pre nájdenie chyby, zatiaľ čo nástroje pre opravu preklepov používali slovníky.

N-gramy sú n znakové podreťazce slov alebo reťazcov. Najčastejšie sa používajú unigramy (1-gram), bigramy (2-gram) a trigramy (3-gram). Vyhľadávanie chýb na základe n-gramov funguje tak, že sa z textu vygenerujú všetky možné n-gramy a tie sa porovnávajú s predtým vytvoreným modelom. Ak sa v modeli n-gram nenachádza alebo sa tam nachádza s veľmi malým výskytom je vysokopravdepodobné, že sa jedná o chybu. Príkladom môže byť trigram *zfq*, ktorý sa v anglickom jazyku často nevyskytuje. Pre vytvorenie modelu je potrebný slovník alebo rozsiahly text.

Rozlišujeme medzi pozičnými a nepozičnými n-gramami. V pozičnom je podstatné aj na akom mieste sa n-tica nachádza v rámci slova. Pozoruje sa ako často sa vyskytol konkrétny reťazec na danej pozícii v slove. Na implementáciu sú jednoduchšie nepozičné, lebo jediné čo nás zaujíma je n-tica znakov. Takýmto najjednoduchším modelom môže byť binárny bigram, ktorý je maticou o veľkosti 26x26 a jeho prvky reprezentujú všetky možné páry znakov v anglickej abecede. Potom sa pridelí značka buď 0 alebo 1 podľa toho či sa taký bigram v texte vyskytoval.

Z tohto princípu vznikli metódy skip-gram a CBOW, ktoré využívajú páry slov na určenie kontextu. Skupina výskumníkov pod vedením Tomáša Mikolova v Google ich použila v riešení *Word2vec*[3], ktorý prevedie slová na vektorovú podobu. S takouto reprezentáciou si rozumejú hĺbkové neurónové siete, čo spôsobilo výrazný pokrok v spracovaní prirodzeného jazyka.

Väčšina kontrol preklepov je založená na príprave jazykovo špecifického slovníka[37]. Pre anglický jazyk to znamená vytvoriť slovník s výlučne anglickými slovami. Vyhľadaním slova v slovníku sa určí či sa jedná o správne slovo alebo preklep. Vytvorenie takého slovníka

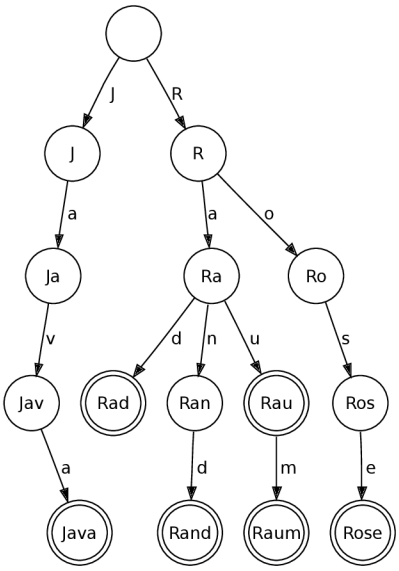
[2] https://www.howtogeek.com/402761/facebook-fudges-your-password-for-your-convenience/
[3] https://pathmind.com/wiki/word2vec

problém nie je, no čo už problémom byť môže je odozva[22]. Na tú môže mať vplyv veľkosť slovníka, ktorý môže obsahovať 25,000 až 250,000 slov. Problém s odozvou slovníka je riešený tromi spôsobmi:

- efektívne vyhľadávanie v slovníku,
- algoritmy na porovnávanie vzorov (pattern matching),
- techniky morfologického spracovania.

Jednoduchým príkladom reprezentácie slovníka je prostredníctvom zoznamu, kde sa nachádzajú všetky slová zoradené. Nájdenie správneho slova by sekvenčne nebolo efektívne. Keďže sa jedná o zoradený zoznam je možné použiť binárne vyhľadávanie. Zoznamy môžu byť roztriedené podľa dĺžky slov pre zmenšenie počtu prvkov, ktoré je treba skontrolovať. Príkladom môže byť overenie správnosti slova o dĺžke 5, v tom prípade sa vyberie zoznam, ktorý obsahuje takto dlhé slová a ušetrí sa čas porovnávaním s nerelevantnými slovami.

Pre rýchly prístup do slovníka sa najčastejšie používa hashovacia tabuľka. Na to aby sme reťazec vyhľadali stačí vypočítať jeho hash a potom zistiť či sa taká adresa v tabuľke nachádza. Hlavnou výhodou je, že nie je potrebné porovnávať veľké množstvo reťazcov či už sekvenčne alebo v nejakej stromovej štruktúre. Na druhej strane je potrebné použiť vhodnú hashovaciu funkciu, ktorá bude predchádzať kolíziam. Unixový program *spell* využíval hashovaciu tabuľku ako dátový typ pre slovník.



Obr. 3.1: Príklad pre trie

Ďalšou možnosťou ako implementovať slovník je použitie stromových štruktúr ako sú ternárny vyhľadávací strom [25], trie (viď obrázok 3.1)[4], binárne vyhľadávacie stromy zoradené podľa frekvencie, stromy so slovami alebo znakmi v uzloch, bloom filter[30] alebo konečné stavové automaty. Aho a Corasick[2] používali konečné stavové automaty práve pre vyhľadávanie vzorov podľa regulárnych výrazov.

Niekdajším problémom bola aj veľkosť operačnej pamäte. To znamenalo, že nebolo možné načítať toľko slov naraz a bolo nutné pristupovať k textovému súboru na viac krát.

[4]Obrázok prevzatý z: https://commons.wikimedia.org/wiki/File:Trie.svg

Riešením je rozdelenie slovníka do niekoľkých úrovní. Peterson [33] navrhoval tri úrovne. Prvá je tvorená malým počtom slov najčastejšie používaných anglických slov, okolo 100-200 slov. Druhá je tabuľka slov špecifická pre daný dokument obsahujúca 1,000-2,000 slov. Posledná úroveň je hlavný slovník, ktorý je statický a veľký, obsahujúci 10,000-100,000 slov. Z podobných dôvodov je možné ukladať aj len základ slova. To znamená, že sa odstránia prípony a predpony, ich správnosť samozrejme musí byť overená a pracuje sa iba so slovotvorným základom.

Problémom je aj určenie veľkosti slovníka. Málo slov môže viesť k často zlému označeniu slova za chybné a naopak pri veľkom slovníku akceptujeme zlé slovo vďaka tomu, že má tvar málo používaného slova. Príkladom môže byť, že v angličtine namiesto *very* napíšeme *veery*, pričom aj to je správne slovo. V americkej a britskej angličtine sa vyslovuje rôzne, kde americká výslovnosť je blízka pôvodnému slovu. Damerau a Mays[12] spochybňujú Petersonov [34] záver, že slovníky by mali byť malé. Z dôvodu, že im sa pri navýšení počtu slov z 50,000 na 60,000, ktoré boli usporiadané podľa frekvencie v zozname podarilo eliminovať viac nesprávnych odmietnutí.

3.2 Typy chýb

Základné 2 typy chýb na ktoré sa automatické opravovanie slov zameriava sú:

- chyby pri ktorých vznikne neexistujúce slovo (nonword error),
- chyby pri ktorých vznikne existujúce slovo (realworld error).

Tie pri ktorých vznikne existujúce slovo sú náročnejšie na detekciu, pretože je potrebné poznať kontext v akom bolo slovo použité. Mitton vo svojej knihe [28] analyzoval chyby v testoch, ktoré vypĺňali žiaci vo veku 15 rokov a zistil, že *realword error* tvorilo 40% chýb. Typy chýb sa líšia podľa prostredia v ktorom vznikli, napríklad ak sa jedná o pisateľa, tak väčšina chýb bude z dôvodu stlačenia zlej klávesy. Naopak chyby vzniknuté z OCR (optické rozpoznávanie znakov) budú založené na podobne vyzerajúcich znakoch napríklad *B* a *8*.

Rozlišujeme medzi tromi chybami z ktorých vznikne nesprávne slovo:

1. Typografické chyby sú také, kde užívateľ napíše namiesto *ahoj ajoj*, kde je predpoklad, že autor má znalosť o tom ako sa dané slovo píše, len stlačil zlý znak na klávesnici.
2. Kognitívne chyby vzniknú z nedostatočnej znalosti užívateľa, teda nevedel správnu podobu slova.
3. Fonetické chyby vzniknú pri náhrade znaku za podobne znejúci. Tento typ je podtriedou kognitívnych chýb.

Základný objav učinil Damerau[11], zistil že približne 80% chybných slov vzniklo jedným z týchto spôsobov:

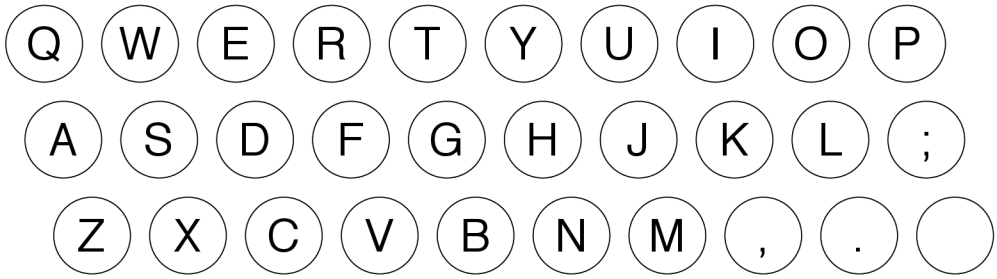
- vloženie,
- zmazanie,
- substitúcia,
- transpozícia znaku.

Chybné slová prevažne obsahovali iba jednu z hore uvedených operácii. Mitton[28] vo svojich dátach pozoroval, že 69% chýb bolo tohto typu. Taktiež uvádza, že podľa zistení tímu pracujúcom na projekte SPEEDCOP iba 7,5% preklepov bolo tvorených viac ako jednou chybou. Z týchto poznatkov je možné usúdiť, že väčšina preklepov obsahuje maximálne dve chyby. Ďalším zistením je, že chyby sa vyskytnú s veľmi malou pravdepodobnosťou na začiatku slova. V analýze od Mittona je iba 7% chýb na prvom znaku, čo teóriu potvrdzuje.

Zdroj	Vloženie	Transpozícia	Zmazanie	Substitúcia
1983 SPEEDCOP	2	4	1	3
1985 [27]	2	4	3	1
2006[42]	1	4	3	2
GPO [21]	3	4	2	1
Web7 [21]	3	4	1	2
2017[39]	3	4	1	2
2018[14]	3	4	2	1
Poradie	3	4	2	1

Tabuľka 3.1: Najčastejšie vyskytujúce sa chyby

Tabuľka 3.1 zobrazuje poradie jednotlivých chybových operácii v dátach, kde 1 znamená, že sa operácia nachádzala najčastejšie a 4 najmenej. Pre porovnanie boli hodnoty zozbierané z rôznych zdrojov. Celkovo najčastejšie vyskytujúcou sa operáciou je nahradenie, s malým rozdielom nasleduje zmazanie, za ním je vloženie a najmenej sa vyskytovala zámena poradia znakov.



Obr. 3.2: Rozloženie QWERTY klávesnice

Stlačenie viacerých kláves naraz alebo stlačenie inej blízkej klávesy je jednou z hlavných príčin vzniku preklepov. V článku od Edelman a Moore[29] si na základe toho odvodili *fat-finger distance*, teda vzdialenosť ktorá berie spomenuté operácie vloženie, zmazanie, substitúciu a transpozíciu vrámci okolitých znakov na QWERTY klávesnici. Často používané rozloženie klávesnice QWERTY je znázornené na obrázku 3.2. Je to taktiež jedno z kritérií pri hľadaní typografických chýb.

Dôvod na vznik transpozičnej chyby je, že pisateľ používa správne rozloženie klávesnice a má vyčlenené znaky, ktoré obsluhuje iba konkrétnou rukou. Ak by v takom prípade nasledovali znaky určené pre jednu ruku znakmi určenými pre druhú ruku môže sa ľahko stať, že jedna ruka predbehne druhú. V tabuľke zhrnutie chýb 3.2 sú spomenuté chyby s príslušným zdrojom, v ktorom sa daný typ chyby spomínal.

Typ chyby	Zdroj
Vynechanie znaku	[3, 25, 37, 41, 47]
Zámena poradia znakov	[25, 37, 47]
Náhrada znaku	[3, 25, 37, 41, 47]
Pridaný znak	[3, 25, 37, 47]
Opakovanie (66 6666)	[41]
Posunutie (441 411)	[41]
Pluralizácia (angličtina)	[22]
Zdvojenie/ alebo vynechanie hlásky	[28]
Fat-finger	[29]
Chyba pri prepise (1 l)	[4]
Fonetická chyba	[25]
Rozloženie klávesnice	[25]
Pôvodné slovo ako prefix	[25, 4]
Zrkadlenie 080 - 808	[41]
Kapitalizácia	[4]
Chyba s ohraničením slov	[25]

Tabuľka 3.2: Zhrnutie chýb

3.3 Kódovanie reťazcov

Funkcie pre kódovanie reťazcov[22, 7] väčšinou fungujú na princípe mapovania reťazcov do takých kľúčov, že podobne znejúce reťazce budú mať taký istý kľúč. Tento proces je závislý na jazyku pretože každý jazyk má inú výslovnosť. Výsledné kľúče je možné využiť na určenie podobnosti dvoch reťazcov. Spomenuté techniky v tejto sekcii sú primárne určené pre anglický jazyk. Štatistický spôsob odvodenia kľúča predviedol systém SPEEDCOP predstavený v subsekcii 3.3.3.

3.3.1 Soundex

Pre účely opravy chýb v texte bol Soundex vyvinutý a patentovaný autormi Odell a Russel v roku 1918[32]. Jedná sa o jeden z najstarších prístupov fonetického kódovania reťazcov. Bol použitý vo viacerých systémoch. Založený je na americko-anglickej výslovnosti. Kóduje reťazce tak, že ponechá začiatočný znak reťazca a nasledujúce znaky konvertuje podľa kódovej tabuľky (viď tabuľka 3.3) na čísla.

a, e, h, i, o, u, w, y	0
b, f, p, v	1
c, g, j, k, q, s, x, z	2
d, t	3
l	4
m, n	5
r	6

Tabuľka 3.3: Soundex mapovanie znakov na čísla

Po transformovaní na čísla sú odstránené všetky 0 a taktiež všetky čísla sú ponechané iba raz (nemôžu sa opakovať). Ak po aplikácii týchto krokov vznikne kľúč, ktorý obsahuje menej ako 3 číslice pripojí sa mu na koniec potrebný počet 0, na dosiahnutie dĺžky 3. Naopak ak je viac číslic, tak sa odrežú. Napríklad s25 je rozšírené na s250 a k7896 je zmenené na k789.

Výhodou tohto spôsobu je jeho jednoduchosť a tým pádom nenáročnosť na výpočtový výkon. Problémom je ak sa reťazce líšia v prvom znaku, vedie to k rozličným kódom. Ďalším problémom je, že sa zameriava na fonetickú podobu na začiatku reťazcov.

3.3.2 Phonex

Tento algoritmus je variantou Soundex prístupu, ktorý sa snaží vylepšiť výsledné kódovania predspracovaním. Zameriava sa na reťazce mien. Je to 11 pravidiel podľa ktorých sú dané reťazce upravené. Podobne ako pri Soundexe sa jedná o kľúč so 4 znakmi, kde prvým je počiatočný znak slovo nasledovaný tromi číslami.

3.3.3 Speedcop

Pollock a Zamora [36] sa zameriavajú na preklepy obsahujúce jednu chybu. Generovanie kľúča je založené na analýze často vyskytujúcich sa chýb z rôznych vedeckých databáz. Najprv sa vygeneruje kľúč pre každé slovo v slovníku, výsledok sa zoradí. Slovo s preklepom je opravené tak, že sa vygeneruje jeho kľúč, nájde sa v zoradenom zozname kľúčov. Ďalej sa berú v úvahu kľúče v oboch smeroch v zozname do určitej vzdialenosti a hľadá sa vhodný kandidát podľa editačnej vzdialenosti.

Využíva 2 typy kľúča, ktoré si nazvali skeleton kľúč a ommision kľúč. Oba kľúče sa skladajú z všetkých zástupcov znakov, ktorí sa v reťazci vyskytujú v špecifickom poradí. Pre skeleton kľúč je prvý znak reťazca na prvom mieste a za ním nasledujú spoluhlásky v poradí podľa výskytu a nakoniec samohlásky podľa výskytu bez duplikátov. Výhodou použitia tohto kľúča je, že časté chyby ako je zdvojenie alebo nezdvojenie znaku, podobne transpozície jednej spoluhlásky a jednej samohlásky (okrem samohlásky na prvej pozícii) sa odhalia a kľúč bude podobný ako u pôvodného slova[45]. Ukázalo sa, že skeleton kľúč je závislý na prvých spoluhláskach a bol to problém pri vynechaní znaku. Preto vznikol kľúč ommision. Je tvorený tak, že najprv idú spoluhlásky v poradí podľa častého vynechania, ktoré zistili ich analýzou a za tým nasledujú samohlásky.

Testovaním so slovníkom o rozsahu 40,000 slov dosahovali opravu preklepov s jednou chybou celkovo s presnosťou medzi 74-88%.

3.3.4 Metaphone

Metaphone je algoritmus pre kódovanie znakov na základe ich fonetiky. Prvá verzia predstavená v roku 1990 autorom Lawrence Philips. Odvtedy zverejnil ďalšie dve verzie. Double metaphone, ktorého fungovanie je známe a Metaphone 3, čo je komerčný produkt.

Double metaphone sa snaží brať do úvahy aj iné jazyky ako je anglický a preto obsahuje komplexnejšie pravidlá pre generovanie kľúča. Je nazývaný double, lebo vygeneruje primárny a sekundárny kľúč (kód) pre daný reťazec.

3.4 Porovnávanie reťazcov

Kniha Data Matching od Petra Christena [7] popisuje techniky spájania dát. Niektoré si uvedieme v tejto sekcii. V problematike vyhľadávania preklepov je potrebné použiť porovnávacie funkcie, ktoré vracajú iné ako binárne hodnoty podobné a rozdielne. Týmto spôsobom by sme vedeli zistiť či je dané slovo v slovníku, ale následne by sme neboli schopný určiť správnu náhradu.

Preto potrebujeme také porovnávacie funkcie, ktoré nám povedia ako veľmi sú si dané reťazce znakov podobné. Požiadavky na takú funkciu by boli, aby generovala výsledné hodnoty v rozmedzí $0 \leq s \leq 1$. Ak je výsledok 1, jedná sa sa úplnú zhodu. Naopak výsledok 0 znamená žiadnu zhodu a výsledok medzi 0 až 1 určuje približnú podobnosť. Medzi vzdialenostnou a podobnostnou funkciou je určitá korešpondencia. Vzdialenosť (d) vieme previesť na podobnosť (s) napríklad ako $s = 1 - d$ a $0 \leq d \leq 1$.

Presnejšie využitie funkcií podobnosť a vzdialenosť je v odhaľovaní podvodov, analýza odtlačkov prstov, detekcia plagiátov, porovnávanie sekvencií DNA/RNA, analýza obrazu, data mining a ďalšie...

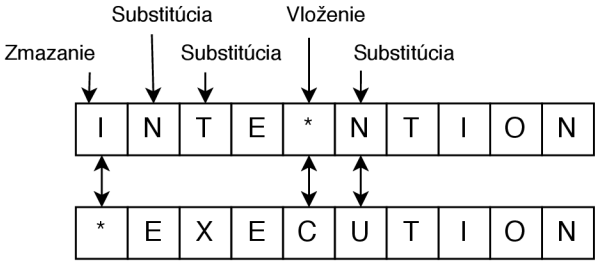
V sekcii 3.4.1 je popísaný pravdepodobne prvý spôsob pre výpočet vzdialenosti dvoch reťazcov. V podsekcii 3.4.2 sa uvádzajú algoritmy s podobným princípom. Ďalej je spomenutá technika q-gramov v sekcii 3.4.3.

3.4.1 Levenshteinova vzdialenosť

Pravdepodobne prvý a jeden z najznámejších spôsobov určenia vzdialenosti reťazcov znakov je Levenshteinova vzdialenosť. Článok pôvodne vydaný v ruštine v roku 1965 o rok neskôr aj anglická verzia [23]. Princíp spočíva v tom, že určíme minimálny počet operácií, ktoré je potrebné vykonať, aby sme pretransformovali jeden reťazec na druhý. Povolené operácie sú:

- vloženie znaku,
- odstránenie znaku,
- substitúcia znaku.

Na obrázku 3.3 vidíme, že aby sme zo slova *intention* dostali slovo *execution* potrebujeme vykonať nasledujúce operácie v slove intention: odstrániť prvý znak, substituovať druhý znak za E, substituovať tretí znak za X, vložiť znak C, substituovať šiesty znak za U. Znakom hviezdičky (*) sú v obrázku označené prázdne pozície. Levenshteinova vzdialenosť medzi slovami *intention* a *execution* je 5.



Obr. 3.3: Príklad rozdielu dvoch reťazcov

Levenshteinovu vzdialenosť môžeme vyjadriť matematicky takto:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j), & \text{ak } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{inak.} \end{cases} \tag{3.1}$$

Prvý riadok v rovnici 3.1 pri minime sa vzťahuje na odstránenie, druhý na vloženie a tretí na zhodu/nezhodu znakov.

Túto metódu je možné implementovať rôznymi spôsobmi. Jedným z nich je rekurzívna verzia, ktorá je neefektívna. Používanejší spôsob je s využitím dynamického programovania, kde si ukladáme vzdialenosti medzi všetkými prefixami prvého aj druhého reťazca do matice. Jeho autormi sú Robert A. Wagner a Michael J. Fisher [46] (Algoritmus 1).

Od riadku 5 po riadok 9 v algoritme 1 je inicializácia matice. Výsledok po tejto operácii môžeme vidieť na 3.4 vľavo, označené ako matica pred vyplnením. Na obrázku je už vypočítaná aj prvá hodnota vyznačená ružovým štvorcom. Tá je vypočítaná z minima medzi ľavou, hornou a hornou diagonálnou hodnotou (vyznačené modrým štvorcom) plus jeden ak sa znaky nerovnajú. Inak sa nová hodnota určí ako horná diagonálna hodnota. V tomto prípade sa znaky G a A nerovnajú, minimum z okolitých hodnôt je nula a s pripočítaním jednotky nám vyjde výsledná hodnota jeden.

Algoritmus 1 Algoritmus Levenshteinovej vzdialenosti

```
1:  function LevenshteinDistance(char s[1..m], char t[1..n])
2:      declare d[0..m, 0..n]
3:      set each element of d to zero
4:
5:      for i from 1 to m do
6:          d[i, 0] = i
7:
8:      for j from 1 to n do
9:          d[0, j] = j
10:
11:     for j from 1 to n do
12:         for i from 1 to m do
13:             if s[i] = s[j] then
14:                 cost ← 0
15:             else
16:                 cost ← 1
17:             d[i, j] ← min(d[i − 1, j] + 1, d[i, j − 1] + 1, d[i − 1, j − 1] + cost)
18:     return d[m, n]
```

Po vyplnení matice je výsledok v poslednom riadku a v poslednom stĺpci. Na obrázku 3.4 vpravo, označené ako matica po vyplnení je to hodnota 3. Zo vzniknutej matice je možné určiť operácie, ktoré je potrebné na akých znakoch vykonať pre transformáciu reťazca z na reťazec x. Začneme v bunke (n, m), kde n a m sú dĺžky reťazcov a pokračujeme smerom do bunky (0, 0). Potrebujeme určiť, z ktorej okolitej hodnoty sme vypočítali aktuálnu hodnotu. Ak sme sa do aktuálnej bunky (i, j) dostali z hodnoty (i, j-1), tak sa jedná o operáciu

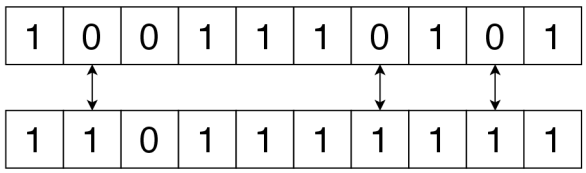
Obr. 3.4: Vytvorená matica po inicializácii a vyplnení

vloženia znaku j druhého reťazca (S_2) do reťazca prvého (S_1). V prípade, že hodnota (i, j) bola určená z (i-1, j), tak je to operácia odstránenia znaku i v S_1. Diagonálne "posuvy" z (i, j) do (i-1, j-1) znamenajú zhodu ak znak i v S_1 je rovný so znakom j v S_2 inak sa jedná o substitúciu [16].

3.4.2 Ďalšie typy editačnej vzdialenosti

Ako sa uvádza v článku[43] bol spôsob určovania vzdialenosti opísaný v sekcii 3.4.1 nazvaný podľa Wagnera a Fishera [46] ako *editačná vzdialenosť* (edit distance). Keďže ako prvý túto myšlienku sformuloval Levenshtein [23], tak je to nazvané po ňom, no často sa tieto pojmy zamieňajú. Kde editačná vzdialenosť je generická vzdialenosť, ktorú je možné určiť váženými hodnotami vloženia, odstránenia, substitúcie a ďalších operácií nad znakmi. Pri Levenshteinovej vzdialenosti tieto operácie majú váhu jeden a môžeme tento algoritmus zaradiť do skupiny pre počítanie editačnej vzdialenosti. Ďalšie typy editačnej vzdialenosti:

- **Najdlhšia spoločná podpostupnosť** - Dovoľuje vloženia a odstránenia znakov. Oboje operácie sú váhy jeden.
- **Epizódna vzdialenosť** - Pracuje iba s vkladaním znakov (váha jeden).
- **Hammingova vzdialenosť** - Je možné ju použiť iba na reťazce rovnakej dĺžky, lebo využíva iba substitúcie znakov (váha jeden). Pomenovaná po autorovi Richradovi Hammingovi, ktorý ju predstavil v roku 1950 [17]. Na obrázku 3.5 je znázornený príklad na Hammingovu vzdielenosť, ktorej výsledná hodnota je 3.



Obr. 3.5: Príklad na Hammingovu vzdialenosť

V článku [35] vylepšili túto vzdialenosť, aby bolo možné porovnať aj dva reťazce rôznej dĺžky. Majme reťazce S_{longer} a $S_{shorter}$ potom vypočítame *vzdialenosť* ako rozdiel dlhšieho reťazca mínus menší reťazec (rovnica 3.2).

$$distance = length(S_{longer}) - length(S_{shorter}), |S_{longer}| \geq |S_{shorter}| \tag{3.2}$$

Kratší reťazec je kontrolovaný po znakoch a ak daný znak je rôzny od znaku v dlhšom reťazci premenná *vzdialenosť* je navýšená o jeden. Posledným krokom je vypočítanie *faktoru vzdialenosti*, ktorý sa vypočíta podľa vzťahu 3.3.

$$distance\_factor = \frac{(length(S_{longer} - distance)}{length(S_{longer})} \tag{3.3}$$

- **Damerau-Levenshteinova vzdialenosť** - Pozostáva z operácií vloženia, odstránenia, substitúcie a transpozície dvoch priľahlých znakov. Od Levenshteinovej vzdialenosti sa líši pridanou operáciou transpozície. Rekurzívne vyjadrenie 3.4 je veľmi podobné tomu zo sekcie 3.4.1 až na posledný riadok, ktorý korešponduje so spomínanou pridanou operáciou a to je transpozícia znakov.

$$d_{a,b}(i,j) = min \begin{cases} 0, & \text{ak } i = j = 0 \\ d_{a,b}(i-1,j) + 1, & \text{ak } i > 0 \\ d_{a,b}(i,j-1) + 1, & \text{ak } j > 0 \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)}, & \text{ak } i,j > 0 \\ d_{a,b}(i-2,j-2) + 1, & \text{ak } i,j > 1 \text{ a } a[i] = b[j-1] \\ & \text{a } a[i-1] = b[j] \end{cases} \tag{3.4}$$

- **Vzdialenosť Jaro (Jaro-Winkler)** - Z pohľadu editačnej vzdialenosti je podstatné, že výpočet vzdialenosti využíva transpozície. Jaro podobnostná funkcia bola navrhnutá pre porovnávanie krátkych reťazcov ako sú mená. Kombinuje editačnú vzdialenosť a q-gram založené porovnávacie techniky. Pre výpočet využíva posuvné okno, ktorého dĺžka je polovica z dĺžky dlhšieho reťazca. V rámci toho posuvného okna počíta rovnaké znaky c a počet transpozícií t priľahlých znakov.

$$sim_j = min \begin{cases} 0, & \text{ak } m = 0 \\ \frac{1}{3}(\frac{c}{s_1} + \frac{c}{s_2} + \frac{c-t}{c}), & \text{inak} \end{cases} \tag{3.5}$$

Na základe štúdií, ktoré odhalili, že menej chýb sa stane na začiatku ako v strede alebo na konci mena boli urobené zmeny na Jaro algoritme (Jaro-Winkler). Prvá modifikácia je, že ak je rozdielnosť dvoch reťazcov až v strede alebo na konci a prefix majú rovnaký, tak zvýši ich podobnosť. Druhá modifikácia upraví podobnosť dvoch reťazcov ak majú oba minimálne 5 znakov a majú aspoň 2 spoločné znaky mimo prefixu. Tretia modifikácia požaduje, aby aspoň polovica zvyšných znakov za prefixom v kratšom reťazci bola zhodná a ak sú všetky 3 podmienky splnené, tak sa upraví podobnosť reťazcov. Jaro-Winkler nespĺňa trojuholníkovú nerovnosť, teda sa nejedná o metriku.

3.4.3 Q-gram

Určovanie podobnosti reťazcov na základe q-gramov je založené na rozdelení dvoch vstupných reťazcov na krátke podreťazce o dĺžke q znakov, tie sa nazývajú q-gramy. Generované sú princípom posuvného okna o veľkosti q. Potom sa spočíta koľko q-gramov je spoločných. Alternatívny názov pre q-gramy je n-gramy. Najčastejšie sa používajú bigramy (q = 2) alebo trigramy (q = 3). Príklad môžeme vidieť v tabuľke 3.4.

Reťazec	Bigramy	Vyplnené bigramy	Trigramy
rok	ro, ok	_r, ro, k_	rok
peter	pe, et, te, er	_p, pe, et, te, er, r_	pet, ete, ter
192	19, 92	_1, 19, 92, 2_	192

Tabuľka 3.4: Príklad na bigramy, vyplnené bigramy a trigramy

Počet q-gramov v reťazci určíme podľa vzorca 3.6, kde c je výsledný počet q-gramov, $|s|$ je počet znakov v reťazci.

$$c = |s| - q + 1 \tag{3.6}$$

Po spočítaní spoločných q-gramov určíme podobnosť reťazcov S_1 a S_2 jednou z týchto metód:

$$\text{Overlap koeficient: } sim_{overlap}(S_1, S_2) = \frac{C_{common}}{min(C_1, C_2)}, \tag{3.7}$$

$$\text{Jaccard koeficient: } sim_{jaccard}(S_1, S_2) = \frac{C_{common}}{C_1 + C_2 - C_{common}}, \tag{3.8}$$

$$\text{Dice koeficient: } sim_{dice}(S_1, S_2) = \frac{2 * C_{common}}{C_1 + C_2}. \tag{3.9}$$

Výsledok je normalizovaná numerická hodnota v rozmedzí <0; 1>. V tabuľke 3.4 je uvedený vyplnený bigram, ktorý je vylepšením klasického bigramu. Podobnosť reťazcov bude tým pádom väčšia pre reťazce s rovnakým začiatkom a koncom a chybami v strede. Počíta sa rovnako podľa rovníc 3.7, 3.8 a 3.9

Ďalšou obdobou n-gramov sú *skip-gramy*, ktoré nevytvárajú bigramy len z priľahlých znakov, ale aj s ďalšími znakmi. Trieda gram-class nám určí aké skip-gramy majú vznikať. Pre gram-class = 0, 1 a reťazec *zima* vzniknú 0-skip gramy (bigramy) zi, im, ma a 1-skip gramy zm, ia.

3.5 Zhrnutie

Na začiatok bolo jemné uvedenie do problematiky. Z uvedených kategórii opravy slova sa táto práca bude venovať *isolated-word error corection*, keďže sa nejedná o súvislý text. Potom sa kapitola venovala spôsobom ako detegovať chybné slová. Na to sa používajú n-gram techniky alebo slovníky. Pomocou n-gramov sa vytvorí model jazyka na pripravených dátach a predpokladá sa, že zistenia je možné aplikovať pri hľadaní chýb aj v inom texte. Slovníky sú zas postavené na princípe, že čo je v ňom je správne. Historicky sa slovníky členili do úrovní z dôvodu obmedzenej pamäte. Používali sa rôzne dátové štruktúry na urýchlenie vyhľadávania slov v programom reprezentovanom slovníku.

V ďalšej časti boli opísané najčastejšie typy chýb. Dôležitý objav, ktorý učinil Damerau a Levenshtein nezávisle v podobnej dobe. Zistili, že väčšina chýb je tvorených z vložení, náhrad, zmazaní a Damerau pridal transpozíciu znaku.

Pre porovnávanie reťazcov sa používajú rôzne techniky. Či už kódovanie znakov, ktoré vychádza z fonetickej podobnosti alebo algoritmy na určenie podobnosti. Základný algoritmus pre určenie rozdielnosti medzi dvomi reťazcami je Levensthteinova vzdialenosť. Počíta sa koľko operácii je potrebných urobiť aby z jedného reťazca vznikol iný reťazec. Na záver boli spomenuté ďalšie možnosti merania takejto vzdialenosti.

Kapitola 4

Návrh kategórií analyzovaných dát

Cieľom tejto kapitoly je návrh kategórií analyzovaných dát. Kategórie bude užívateľ priraďovať k vstupným dátam. Týmto sa zabezpečí redukcia dát pre analýzu a aj výber toho v čom má zmysel preklepy hľadať. Keďže preklep nie je strojovo vytvorený, ale vznikne ľudskou chybou, tak je potrebné navrhnúť také kategórie, kde sú dáta zadané od užívateľa. Pretože práca je zameraná na diagnostiku chyb v sieťach ďalšou podmienkou je, aby tieto údaje boli prenášané sieťou.

Pre detekciu chyby budú využité slovníky. Kategórie pri ktorých to má zmysel budú mať slovník obsahujúci známe hodnoty. Vďaka kategorizácii je možné dosiahnuť lepších výsledkov, či už tým, že nebudú všetky informácie v jednom slovníku a teda hľadanie bude rýchlejšie, ale aj tým, že sa zachová informácia pôvodu znakového reťazca, čo by bola škoda nevyužiť.

Pre tieto účely boli vybraté nasledujúce informácie:

- IP adresa,
- port,
- internetová doména,
- všeobecné číslo,
- všeobecný reťazec,
- prihlasovacie meno.

Kvôli tomu, že bežne užívateľ nepracuje s IPv6 adresou, tak nás ďalej bude zaujímať iba IPv4. V ďalších sekciách si tieto kategórie opíšeme bližšie.

4.1 IP adresa

Keďže každý sieťový prvok má svoju IP adresu, je to vhodný údaj na spracovanie. Pakety a toky obsahujú zdrojovú a cieľovú adresu. Prevažne sa využívajú IPv4 adresy, preto sa budem v práci zameriavať iba na tento typ. Je tvorená štyrmi oktetmi oddelenými bodkou, teda sa jedná o 32 bitovú adresu (obmedzenie adresného priestoru na cca 4,3 miliardy adries). To znamená že každá časť IP adresy je v rozsahu od 0 po 255, platná IP adresa je napr.: "192.168.0.1".

Bude vytvorený kontextový slovník pre vyhľadávanie validných slov obsahujúci všeobecne dostupné resp. známe IP[1] adresy. Pod známymi adresami sa myslia adresy smerovačov[2], DNS serverov alebo localhost. Okrem toho na základe predchádzajúcej komunikácie bude vytvorený naučený slovník obsahujúci používané IP adresy.

IPv4 adresy
127.0.0.1
8.8.8.8
192.168.0.1
192.168.1.1
4.2.2.2

Tabuľka 4.1: Príklad obsahu kontextového slovníka pre IPv4 adresy

Príklad na kontextový slovník je v tabuľke 4.1. I keď každé zariadenie má svoju IP adresu, tak ju väčšinou nezadáva priamo užívateľ. Pre túto prácu sú dôležité práve údaje zadané od užívateľa. Ten môže zadávať IP adresu kvôli konfigurácii mailového klienta alebo sa pomocou FTP protokolu snaží pripojiť na vzdialený server.

4.2 Transportný port

Port transportnej vrstvy slúži na rozlíšenie jednotlivých služieb v rámci jedného zariadenia. Číslo portu pre svoje služby si môžu predajcovia oficiálne zaregistrovať u organizácii IANA, ktorá má na svojich stránkach zoznam na čo jednotlivé porty slúžia. Samozrejme, že organizácia sa môže aj svojvoľne rozhodnúť aký port budú používať. Porty sú delené do troch skupín:

- známe porty,
- registrované porty,
- dynamické a súkromné porty.

Port je tvorený šestnástimi bitmi, teda má rozsah možných hodnôt od 0 až po 65,535. Prvých 1024 portov je vyhradených pre známe, používané sieťové služby. Príkladom je port 22 pre službu SSH. V rozsahu od 1024 až 49,151 sú registrované porty, ktorých použitie by malo byť zaregistrované. Príkladom je port 3306 registrovaný pre MySQL databázový systém. V rozmedzí 49,152 až 65,535 sú dynamické a súkromné porty, ktoré sa nedajú zaregistrovať a mali by sa používať dočasne.

Podobne pre port je vytvorený kontextový a naučený slovník. Obsah kontextového slovníka je vhodné vybrať na základe služieb, ktoré sa v danej sieti používajú. Či už sú to služby známe alebo interné pre danú organizáciu. Čím lepšia špecifikácia, tým je skôr možné nájsť opravu. Príklad portov, ktoré je možné použiť v kontextovom slovníku vidíme v tabuľke 4.2.

[1] Známe IP adresy https://www.quora.com/What-are-some-famous-IP-addresses

[2] Zoznam IP adries pre smerovače https://www.techspot.com/guides/287-default-router-ip-addresses/

Známe porty	Služba
20	FTP dátový prenos
22	SSH
23	Telnet
25	SMTP
80	HTTP

Tabuľka 4.2: Príklad portov pre známe služby

Užívateľ zadáva port napríklad pri konfigurácii nejakej aplikácie alebo keď sa chce pripojiť na službu vzdialeného stroja.

4.3 Doménové meno

Doménové meno je unikátny identifikačný reťazec. Práve o službe DNS je najznámejšie, že okrem iného mapuje IP adresy k doménovým menám, tvorí databázu rôznych záznamov. Doménové meno je vytvorené na základe pravidiel daných DNS. Domény sú uložené v stromovej štruktúre s koreňovým prvkom root. Jedná sa o hierarchickú štruktúru, ktorá člení doménové meno do subdomén.

Prvá úroveň sú top level domény (TLDs, gTLDs, ccTLDs), medzi prvými sem patrila doména com. Ďalšie úrovne sú bežne dostupné k registrácii. Celé doménové meno je obmedzené na 253 ASCII znakov v textovej podobe. Najpravejšia časť je spomínaná TLD a na ľavej strane od nej oddelené bodkou sa nachádzajú subdomény. Subdoména môže byť tvorená 1 až 63 oktetmi. Keďže je doménové meno obmedzené dĺžkou 253 znakov, tak je možné použiť maximálne 127 úrovní ak by sme použili iba znak na subdoménu (127 znakov a 126 bodiek). DNS inak nelimituje počet subdomén, ale je limitovaný počet znakov celej doménovej adresy. Príklad domény s com TLD:

www.priklad.com

Časť e-mailovej adresy za znakom @ je taktiež doménové meno. V tomto prípade slúži na identifikáciu mailového servera. Je to jeden z prípadov, kde sa užívateľ pri písaní môže pomýliť. Inak je bežné, že pri písaní webovej adresy vznikne preklep. Kvôli tomu, že oproti ostatným kategóriám užívatelia najčastejšie pracujú s doménovým menom je značné, že najviac preklepov by sa detegovalo pre túto kategóriu.

Do kontextového slovníka je možné umiestniť napríklad 1000 najpoužívanejších domén. Služba Amazon Alexa verejne sprístupňuje 500[3] najpoužívanejších domén. Za zvyšné informácie je potrebné si zaplatiť. Na internete sú dohľadanejšie aj rozsiahlejšie záznamy, ale nie sú aktuálne k dnešnému dňu.

4.4 Všeobecné číslo

Číselný atribút vyskytujúci sa v zdroji dát bude môcť byť identifikovaný ako *číslo*. Z bezchybných vstupných dát bude vytvorený naučený slovník používaných slov slúžiaci pre detekciu chyby a generovanie možných opráv. Takisto je možné vytvoriť aj kontextový slovník, kde sa pridajú užívateľom používané hodnoty. Hodnota nie je obmedzená na celočíselnú alebo desatinnú podobu. Je to voľnejšia kategória.

[3] https://www.alexa.com/topsites

4.5 Všeobecný reťazec

Atribút vyskytujúci sa v zdroji dát typu textový reťazec podobne ako *všeobecné číslo* je užívateľom identifikovaný ako *reťazec*. Podobne ako kategória všeobecné číslo je to voľná kategória, ktorú je možné využiť ak potrebujeme identifikovať dáta, ktoré nespadajú do ostatných kategórii. Naučený slovník je vytvorený z bezchybného vstupu dát a slúži na detekciu chyby a generovanie náhrad. Kontextový slovník tvoríme podľa potreby.

4.6 Prihlasovacie meno

Ďalej je vytvorená špeciálna kategória pre **prihlasovacie mená**, ktorá urýchli vyhľadávanie a hľadanie náhrad. Miešaním tejto kategórie s *všeobecným reťazcom* sa stráca pridaná hodnota informácie, ktorú táto kategória nesie. Do kontextového slovníka je možné uviesť známe, používané prihlasovacie mená. Užívateľ sa prihlasuje svojim užívateľským menom do rôznych služieb, aplikácii. Tieto prihlásenia ak aj sú v sieťovej komunikácii šifrované, môžu sa objaviť v logoch.

4.7 Kombinované kategórie

Existujú prípady, kedy dáta pozostávajú z viacerých kategórii, ktoré boli špecifikované. Pre tieto potreby vznikli kombinované kategórie. Môžu byť tvorené z viacerých základných kategórií, oddelené špecifickým znakom. Príkladom je emailová adresa, kde jej tvar je *prihlasovacie meno* oddelené znakom @ od domény, uvedený nižšie.

username@domain.com

Tým, že sa k dátam bude pristupovať po jednotlivých kategóriach je možné využiť aj slovníky, ktoré vznikli z iných informácii pre vytvorenie kandidáta alebo náhrady. Takže ak sa objaví e-mail s chybou v doméne, môžem tú doménu opraviť na základe domén, ktoré sú navštevované alebo známe. Výhodou je uchovávanie celej informácie zadefinovaním takéhoto typu. Bude potrebné vyriešiť situáciu, keď na jednom mieste môže byť viacero kategórii.

Kapitola 5

Návrh

Poznatky z predchádzajúcich kapitol budú aplikované pri návrhu a implementácii systému pre vyhľadávanie a opravu chýb v sieťových dátach. V tejto kapitole je popísaný návrh výsledného systému. Systém by mal z vhodne zvolených vstupných dát identifikovať nesprávne napísané slová, tj. preklepy. Pod pojmom slovo v kontexte tejto práce sa myslí znakový reťazec spadajúci do jednej z kategórií popísaných v kapitole 4. Pre nájdený preklep sa systém pokúsi nájsť správnu náhradu. Nakoniec sa chybné slová s prípadnou opravou zobrazia užívateľovi.

Výsledný program nie je určený pre koncového užívateľa. Má slúžiť pre uľahčenie práce sieťových administrátorov, ktorý potrebujú nájsť a opraviť problém. Preto sa vyžaduje znalosť problematiky a užívateľ tohto systému potrebuje vedieť v akých dátach a na akom mieste chce problém hľadať. Keďže sa predpokladá zručnosť a znalosť práce s terminálom, systém bude riešený ako konzolová aplikácia.

Systém bude fungovať na princípoch slovníkov. To čo je v slovníku je platné a správne, to čo sa v ňom nenachádza je chybné. Na tento účel mohla byť využitá analýza n-gramov (spomínané v 3.1) , ale vo svete internetových domén a prezývok neplatia tie isté pravidlá ako v spracovaní prirodzeného jazyka. To by mohlo znamenať množstvo nesprávne nájdených chýb. Pre detegované nesprávne slová sa v slovníkoch budú hľadať možné opravy a pomocou heuristických metód sa nakoniec vyberie vhodná náhrada.

Cieľom programu nie je určovať kedy nastala chyba a kde ju treba hľadať. Užívateľ preto bude musieť určiť z ktorých dát vychádzať ako referenčných a podľa nich sa budú overovať v dátach označených ako chybné. Z tohto dôvodu bude výsledný systém disponovať dvomi režimami a to fáza učenia a fáza diagnostiky. Vo fáze učenia sa z bezchybných dát vytvorí slovník používaných slov. Pri druhej fáze sa bude pracovať na hľadaní nesprávnych slov.

5.1 Vstup programu

Na začiatok je potrebné zvoliť vhodné vstupné dáta s ktorými bude program pracovať. V kapitole 2 boli predstavené možnosti sieťovej analýzy. Pre túto prácu som sa rozhodol využiť pasívny prístup, kde dáta budú najprv zaznamenané a v prípade potreby spracovávané. V tomto prípade je práca s dátami a ich získanie jednoduchšie ako pri aktívnom prístupe. Program bude podporovať viac formátov dát, kvôli diverzite monitorovacích spôsobov. Neočakáva sa práca s prúdom dát. Je možné túto činnosť vykonávať spúšťaním programu nad časťami zachytenej komunikácie, takzvaným dávkovaním.

Vstupom bude môcť byť súbor formátu pcap, ktorý je možné vytvoriť pomocou programu Wireshark. Dôvodom pre výber tohto typu dát je že, obsahuje najväčšie množstvo informácii.

Ďalším vstupným typom dát je záznam o aktivite respektíve log záznamy. Typov takýchto záznamov je veľa, či už to bol v sekcii 2.1.3 spomínaný syslog alebo to môže byť weblog. Takisto si môžeme nadefinovať svoj typ záznamu s vlastným formátom. Pre potreby analýzy je možné transformovať akýkoľvek dátový zdroj do vlastného formátu logu. To znamená, že je možné v konečnom dôsledku analyzovať akékoľvek dáta. Záznam o aktivite napríklad obsahuje užívateľské meno, čas pokusu prihlásenia a či bol tento pokus úspešný. Program bude musieť byť schopný tieto vstupné dáta spracovať aj napriek rôznorodosti formátov záznamoch o aktivitách.

Posledným podporovaným typom je NetFlow. Je možné ho exportovať do CSV formátu, ktorý je známy a nenáročný na spracovanie. Využíva sa pri monitorovaní sietí, oproti zachytávaniu paketov je kompaktnejší.

5.1.1 Konfiguračný súbor pre vstupné dáta

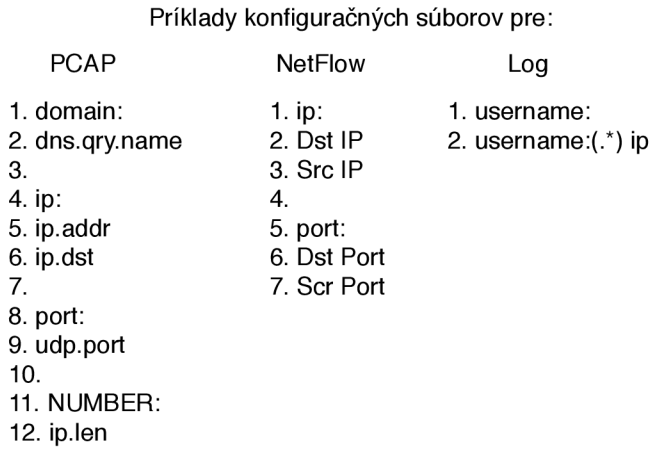
Zo vstupných dát bude administrátora zaujímať iba ich určitá časť a preto je potrebné zadefinovať, o ktoré dáta má záujem. To je možné definovať pomocou konfiguračného súboru, ktorý je taktiež vstupom programu. V ňom budú namapované kategórie dát (pozri kapitolu 4) na spôsob získania vstupných dát. Je nevyhnutné, aby užívateľ zadal kategóriu pre atribút, ktorý bude pozorovaný vo vstupných dátach.

Kategórie uvádzané v konfiguračnom súbore nie sú zapisované celým názvom, ale budú sa zapisovať týmto spôsobom (hodnota uvedená za pomlčkou):

- doménové meno - domain,
- IP adresa - ip,
- transportný port - port,
- všeobecné číslo - number,
- všeobecný reťazec - string,
- prihlasovacie meno - username.

Príklady konfiguračných súborov pre každý typ vstupných dáta je možné vidieť na obrázku 5.1. Pre rôzne typy konfiguračných súborov čísla označujú poradie riadkov. Súbor začína názvom kategórie, nezáleží na tom či sú použité veľké alebo malé písmena. Za kategóriou je uvedená dvojbodka a pod ňou je spôsob získania dát pre konkrétnu kategóriu. Tých spôsobov môže byť aj viac (podobne ako v konfigurácii pre pcap v kategórii IP na obrázku). Za posledným spôsobom získania dát je prázdny riadok a môže nasledovať ďalšia kategória.

Súbor pcap je možné pomocou nástroja Wireshark exportovať do JSON formátu, takisto aplikovať filtre pre zredukovanie obsahu na ten, ktorý nás zaujíma. Užívateľ využije pre identifikovanie dát na analýzu kľúčové slová, ktoré Wireshark používa. Napríklad keď užívateľa zaujíma IP adresa cieľovej stanice použije kľúč *ip.dst*. Preto je podstatné, že užívateľ vie, kde chce problém hľadať, aby mohol určiť dáta pre program. Výsledný konfiguračný súbor pre pcap na obrázku obsahuje 4 kategórie a to pre doménové meno, IP adresu, port



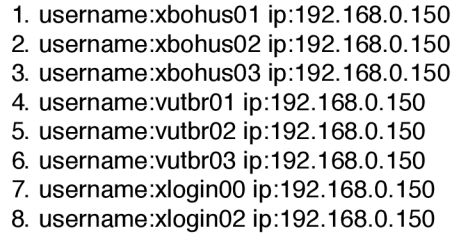
Obr. 5.1: Obsah konfiguračného súboru pre vstupné dáta

a číslo. Do programu vstupujú zachytené pakety v binárnej podobe, ktoré vie konzolová aplikácia TShark interpretovať.

NetFlow dáta je možné exportovať pomocou programu nfdump do CSV formátu. Potom sú výsledné dáta usporiadané do stĺpcov, kde prvý riadok obsahuje ich názvy. Predpokladá sa pripravenie do CSV formátu. V konfiguračnom súbore uvedieme ako zdroj názvy stĺpcov, z ktorých sa budú dáta analyzovať. Na obrázku s konfiguráciami je v strede obsah súboru NetFlow, ktorý určuje, že pre program zo zdrojových dát sú zaujímave stĺpce *Dst IP, Src IP, Dst Port a Src Port*. Stĺpcom je pridelená príslušná kategória. V práci budem pre NetFlow dáta vo formáte CSV používať označenie NetFlow.

Posledným podporovaným vstupom je log súbor. Vďaka jeho rôznym podobám je potrebné, aby užívateľ zadefinoval regulárny výraz[1], ktorý nájde všetky použiteľné slová. Je možné využiť nezachytávajúce skupiny, kde je uvedené čo sa nachádza pred slovom našeho záujmu, nasledujú zátvorky v ktorých je regulárny výraz zachytávajúci hľadané slová a za zátvorkami môže nasledovať to čo sa nachádza za slovom.

Pre názornú ukážku si uvedieme príklad záznamu, z ktorého vyberieme časť informácii. Budeme vyberať užívateľské mená zo vstupných dát, tie sú vo formáte ako je uvedený na obrázku 5.2. Znázornených je 8 príkladov záznamu aktivity, každý na riadku označenom číslom.



Obr. 5.2: Príklad záznamov aktivity

[1]Prehľad o regulárnych výrazoch dostupný na https://www.regular-expressions.info/

Pre tento prípad je potrebné zapísať regulárny výraz.. Záznam začína vždy reťazcom "*username:*". To je možné využiť a vytvoriť tak pravidlo, ktoré daný reťazec nezachytí, ale zachytí to čo za ním začína. Následne potrebujeme zachytiť akékoľvek znaky až po medzeru, za ktorou nasleduje slovo "*ip*". Výsledný regulárny výraz je:

username:(.*) ip

Na obrázku 5.1 je znázornené ako by vyzeral konfiguračný súbor pre dáta z log súboru v takomto formáte.

5.1.2 Slovníky

Na určovanie správnosti slov sa využijú slovníky. Budú rozdelené na 2 kategórie a to primárne a sekundárne. Primárny slovník vznikne učením zo vstupných dát, o ktorých vieme že sú bezchybné. Dodatočná informácia, ktorá sa bude zisťovať pre každé používané slovo je jeho frekventovanosť. Tú využijeme pri heuristických metódach pre výber náhrady chybného slova. Primárny slovník bude programom vyexportovaný vo formáte JSON. Jednotlivé kľúče sú názvy kategórií dát. Pod kľúčom sa nachádza pole párov - hodnota, frekventovanosť. Príklad takto uloženého slovníka vidíme na obrázku 5.3.

```
{
   "port": [["80", 3], ["8000", 1], ...],
   "domain": [["facebook.com", 4], ["youtube.com", 2], ...],
   "ip": [["192.168.0.1", 5], ["127.0.0.1", 2], ...],
   "number": [["123", 1], ["234", 1], ["354789", 1], ...],
   "string": [["dhcp", 4], ...],
   "username": [["xbohus01", 10], ["admin", 7], ...]
}
```

Obr. 5.3: Príklad formátu primárneho slovníka

Sekundárne slovníky držia všeobecnú znalosť a môžu byť vytvorené pre každú dostupnú kategóriu, no nie vždy majú zmysel a musia byť ručne vytvorené predom. Pre kategóriu *doménových mien* bude takýto slovník obsahovať tisíc najpoužívanejších domén. Pre *všeobecný reťazec* to môže byť zoznam (napríklad 10000) najpoužívanejších anglických slov.

1. 4.2.2.1
2. 4.2.2.2
3. 4.2.2.3
4. 4.2.2.4
5. 4.2.2.5
6. 4.2.2.6
7. 8.8.8.8
8. 8.8.4.4
9. 208.67.222.222
10. 208.67.220.220

Obr. 5.4: Príklad sekundárneho slovníka pre kategóriu ip

Sekundárny slovník pre kategóriu *IP adries* obsahuje známe IPv4 adresy, napríklad pre službu DNS. Podobne obsahuje sekundárny slovník pre *transportné porty* zoznam najpoužívanejších portov. Formát sekundárneho slovníka je jednoduchý. Jedná sa iba o súbor s každou hodnotou na novom riadku. Je to ľudsky jednoducho čitateľné a editovateľné. Časť

sekundárneho slovníka pre kategóriu IP adries je na obrázku 5.4, kde prvý stĺpec označuje číslo riadku.

Pre kategóriu *prihlasovacie meno* nie je možné nájsť všeobecne známe hodnoty ako pre IP adresy alebo domény. Avšak je možné vytvoriť slovník z prihlasovacích mien o ktorých vieme, že sa používajú v danej sieti. Pre kombinovanú kategóriu je možné taktiež vytvoriť sekundárny slovník, ale oprava sa bude hľadať v slovníkoch základných kategórii.

5.1.3 Konfiguračný súbor pre program

Spomínané sekundárne slovníky je potrebné nejakým spôsobom dostať do programu. Keďže môžu byť užívateľom editovateľné, tak môžu byť aj ľubovoľne umiestnené na disku.

Sekundárne slovníky sú užívateľom editovateľné a môže ich mať ľubovolne uložené na disku. Preto bude potrebné k nim programu zadefinovať cestu. Takisto je potrebné niekde umožniť dodefinovanie kombinovaných kategórií. Tieto informácie sa uložia do ďalšieho konfiguračného súboru, ktorý bude obsahovať všetky dodatočné informácie pre beh programu. Tento súbor bude používať formát YAML, ktorý slúži na serializáciu dát. Je čitateľný nie len pre stroj, ale aj pre človeka.

Príklad obsahu konfiguračného súboru pre beh programu je na obrázku 5.5. Ako prvé budú zadefinované cesty k sekundárnym slovníkom. Tie sa budú nachádzať pod kľúčom *dictionaries*. Prítomnosť kľúča je povinná a na jeho zanorenej úrovni sú k názvom kategórii namapované cesty v súborovom systéme ku sekundárnym slovníkom.

```
# najprv je potrebné zapísať cesty ku kontextovým/sekundárnym slovníkom
dictionaries:
   ip: "files/secondaryDicts/ip.txt"
   string: "files/secondaryDicts/strings.txt"
   number: "files/secondaryDicts/numbers.txt"
   domain: "files/secondaryDicts/domains.txt"
   port: "files/secondaryDicts/ports.txt"

# je možné nadefinovať kombinované kategórie
# na zanorenej úrovni je názov novej kategórie s jej hodnotami
combined_categories:
   hostname:
      formats:
         - domain:port
         - ip:port
         - ip
         - domain
      separators:
         - ":"
      to_save:
         - port
         - domain: (?:[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z0-9][a-z0-9-]{0,61}[a-z0-9]
         - ip

# pre PCAP vstup je možné určiť filtračný reťazec
pcap_filter: dns.qry.name && dns.flags.response == 0 && ip.src == 192.168.1.145 && dns.qry.type == 1
```

Obr. 5.5: Návrh konfiguračného súboru potrebného pre beh programu

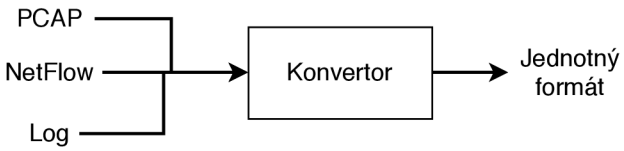
Pre zadefinovanie kombinovanej kategórie sa použije voliteľný kľúč *combined_categories*. Na jeho zanorenej úrovni sa zadá názov novej kategórie ako kľúč a na ďalšej úrovni sa doplnia povinné kľúče *formats, separators, to_save*. Vo *formats* sa zapíšu rôzne formy vytvorené zo

základných kategórii. Napríklad pre *hostname* to môže byť *ip, domain, ip:port, domain:port*. Ďalej je potrebné zapísať aké oddeľovače sa používajú z dôvodu rozdelenia zadefinovaných foriem na ňom. To sa špecifikuje vrámci kľúča *separators*. Keď je vhodné jednotlivé kategórie ukladať do primárneho slovníka samostatne, pri kľúči *to_save* je potrebné jednotlivé kategórie špecifikovať. V prípade, že nie je možné jednoznačne rozlíšiť o akú kategóriu sa jedná je potrebné dopísať regulárny výraz pre rozhodnutie. Ak sa vyberá napríklad medzi 2 kategóriami, stačí napísať regulárny výraz iba pre jednu z nich. Kombinované kategórie musia byť v konfiguračnom súbore zapísané v situácii, keď sa použije primárny slovník, ktorý ich obsahuje. V opačnom prípade môže dôjsť k chybe.

Keďže zachytené pakety sa do programu dostanú v binárnej podobe čitateľnej programom Wireshark je možné na ne aplikovať filtračný reťazec zapísaný podľa ich pravidiel. Filtrovanie prebehne spolu s konverziou pcap súboru do JSON podoby pomocou externého programu. Preto posledným kľúčom je *pcap_filter*, ktorý obsahuje zadaný reťazec a vďaka nemu sa môže urýchliť proces konverzie.

5.2 Konvertor

Vstupné dáta nie sú jednotného formátu, preto je potrebné ich transformovať do rovnakej podoby, aby ďalšie časti programu s nimi vedeli pracovať. Konvertor sa postará o túto činnosť a zabezpečí jednotnosť vstupných dát pre zvyšok programu. Tento proces je vyobrazený na obrázku 5.6.



Obr. 5.6: Návrh činnosti konvertora

Výstup z konvertora poputuje ďalej buď do časti na opravu preklepov alebo export slovníka vo formáte “kľúč:hodnota”. Kľúčom sú jednotlivé kategórie a hodnotou budú slová patriace do nej. Konkrétne ho môžeme vidieť na obrázku 5.7. Hodnoty v príslušných zoznamoch sa môžu opakovať. Pred poslaním do ďalšej časti na spracovanie sa výstup prispôsobí jej požiadavkam.

```
{
   "port": ["80", "8000", ...],
   "domain": ["facebook.com", "google.com", ...],
   "ip": ["192.168.0.145", "127.0.0.1", ...],
   "number": ["123", "234", "354789", ...],
   "string": [],
   "username": ["xbohus01", "bond777", ...]
}
```

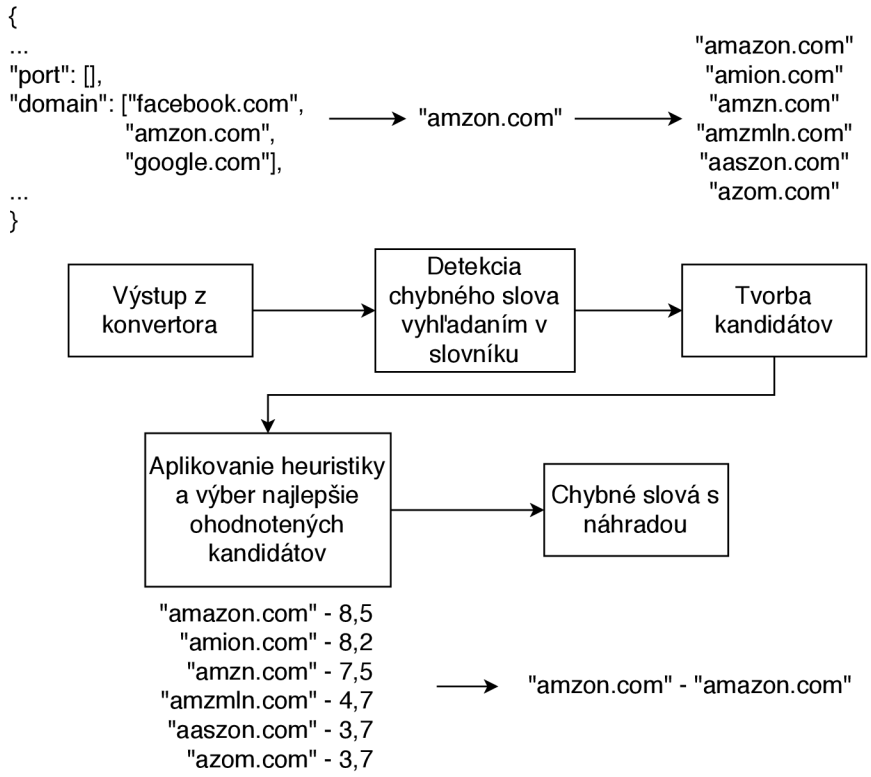
Obr. 5.7: Formát výstupu konvertora

5.3 Oprava preklepov

Hlavná časť programu bude mať za úlohu identifikovať chybné slová a ak je to možné, takémuto slovu ponúknuť opravu. Očakávané chovanie je možné vidieť na obrázku 5.8. V tejto práci sa na identifikáciu správneho slova použijú slovníky. Slovník obsahuje slová, ktoré sú známe a ak sa tam hľadané slovo nenachádza budeme môcť o ňom prehlásiť, že je chybné. Overuje sa prítomnosť v primárnom a sekundárnom slovníku podľa príslušnej kategórie slova.

Ďalej je potrebné nájsť v slovníkoch vhodných kandidátov na opravu. Techniky na určovanie podobnosti slov boli popísané v sekcii 3.4. Použijem prístup generácie nových kandidátov do editačnej vzdialenosti 2, ktorý vychádza z článku od vedúceho výskumu v Google Petra Norviga[31]. To znamená, že na slovo sa aplikujú všetky možné kombinácie dvoch reverzných operácii vloženia, nahradenia, zmazania a transpozície znaku. Generovanie bude prebiehať iba do vzdialenosti 2 na základe zistení v sekcii 3.2. Ukázalo sa, že prevažná väčšina preklepov obsahuje iba jednu z vyššie spomenutých operácii. Preto hľadanie do vzdialenosti 2 musí byť postačujúce. Vygenerované slová sú overené na prítomnosť v slovníku a tie čo sa našli sú považované za kandidátov.

Po zistení kandidátov treba určiť toho správneho, ktorý bude náhradou za chybné slovo. To zistíme aplikovaním určitých pravidiel a štatistických zistení. Navrhnutý systém na opravu teda pozostáva z troch častí a to sú: detekcia chybného slova, zistenie kandidátov a aplikovanie heuristík (obrázok 5.8). Na obrázku môžeme taktiež vidieť aktuálne spracovávané formy dát pri jednotlivých častiach programu.



Obr. 5.8: Princíp fungovania detekcie a opravy preklepov

5.3.1 Heuristika hodnotenia kandidátov

Po vygenerovaní kandidátov na opravu slova je potrebné z nich určiť najlepšiu náhradu. Najjednoduchšie by bolo vybrať kandidáta s najmenšou editačnou vzdialenosťou. Problémom je, že niekoľkí kandidáti môžu mať rovnakú minimálnu vzdialenosť od chybného slova. Na základe tejto informácie nevieme vybrať najvhodnejšieho kandidáta. V sekcii 3.2 boli opísané rôzne typy chýb, ktoré môžu nastať. Ohodnotenie kandidátov bude založené práve na kontrole či sa tieto chyby v zle napísanom slove nachádzajú.

Okrem spomenutej editačnej vzdialenosti a typu chyby je možné použiť znalosť o pôvode kandidáta. Keď sa kandidát nachádza v primárnom slovníku, znamená to, že ho už nejaký užívateľ predtým použil. Ďalšou veľmi podstatnou informáciou je ako často ho použil. Klasický predpoklad je, že slovo s vyšším výskytom má väčšiu pravdepodobnosť.

Možné je overiť podobnú fonetiku. Problémom je použiteľnosť iba pre kategóriu dát *všeobecný reťazec*. Pre ostatné kategórie to veľmi zmysel nemá, lebo názvy domén nevznikajú v zásade z reálnych slov a prihlasovacie meno je na tom podobne. Prihlasovacie meno zvykne byť doplnené o špeciálne znaky a čísla. Čo ale pri doméne môžeme overiť je, či nastala chyba v okolí bodky, ktorá oddeľuje úrovne domény alebo jej prítomnosť. Ďalším dôležitým zistením je, že je veľmi malá pravdepodobnosť vzniku chyby na prvom znaku. Pri chybách typu vloženie, náhrada, transpozícia a zmazanie znaku vieme ako často sa v texte tento typ chyby vyskytuje, čím sa dá ovplyvniť obodovanie kandidáta.

Na základe týchto informácii je vytvorená heuristika pre ohodnotenie kandidátov. Tí dostávajú odmenu za výskyt očakávanej chyby alebo v špeciálnych prípadoch. Špeciálnymi situáciami sa myslí výskyt v primárnom slovníku, slovo s najväčšou frekvenciou výskytu z primárneho slovníka a slová s editačnou vzdialenosťou jeden. Vo výnimočnej situácii sa stane, že celé slovo je napísané opačnými znakmi ako správne slovo. Pretože slovo, ktoré malo byť napísané je jednoznačné, tak je za to najvyššia odmena.

Kandidát začína s nulovým počiatočným ohodnotením. Odmeny a sankcie podľa operácie sú najprv zvlášť sčítané a nakoniec je výsledná odmena za operáciu vypočítaná na základe pomeru operácie:

hodnota_kandidáta = odmeny * (1 - pomer_operácie) + sankcie * pomer_operácie

Za každú operáciu je udelená samostatná sankcia prenásobená pomerom operácie. Pomery boli stanovené na základe poradia výskytov v rôznych zdrojoch uvedené v tabuľke 3.1. Ich hodnoty je možné vidieť pri každej z operácii zapísané ako "r = hodnota". Základná odmena má hodnotu 2 a v ostatných prípadoch bude uvedená nižšie vo výpise. V prípade, že je vložený alebo nahradený znak, ktorý nie je z okolia znaku na klávesnici, tak je udelená základná sankcia v hodnote -2. Výpis všetkých hodnotených prípadov sa nachádza v nasledujúcom zozname s veľkosťou odmeny v zátvorkách, ak je iná ako základná:

- Slovník:
  - Odmena za pôvod z primárneho slovníka. (5b)
  - Odmena pre kandidáta s najväčším výskytom. (4b)
- Odmena za editačnú vzdialenosť 1. (5b)
- Väčšia vzdialenosť je bez odmeny
- Podľa operácie:

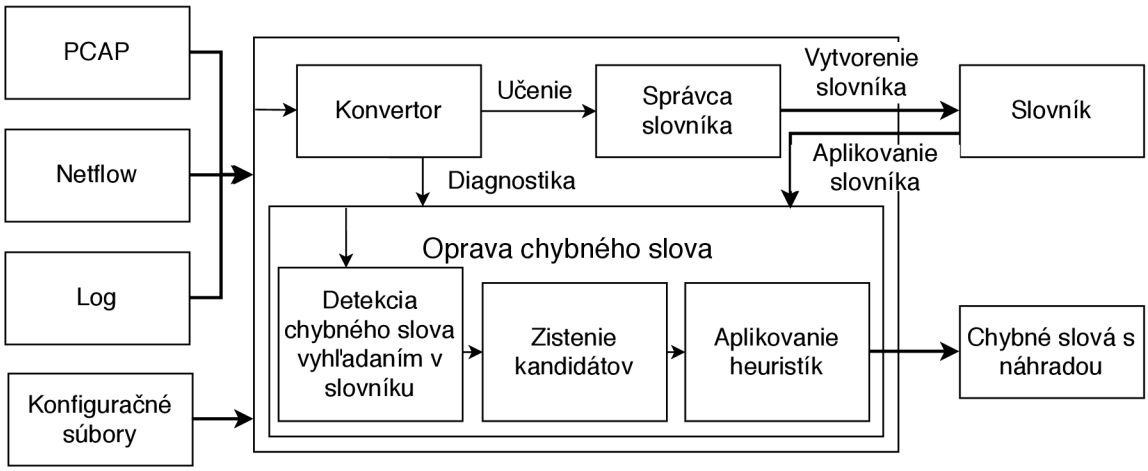
- Vloženie (ahpoj namiesto ahoj, r = 0.5):
  - Odmena za zdvojenie znaku (ahooj namiesto ahoj).
  - Odmena alebo sankcia za *fat finger distance* (ahjoj namiesto ahoj).
  - Odmena pri chybe opakujúcich sa znakov (looop namiesto loop).
  - Odmena keď je kandidát ako prefix (ahojqw namiesto ahoj).
- Transpozícia (haoj namiesto ahoj, r = 0.7).
- Zmazanie (hoj namiesto ahoj, r = 0.5):
  - Odmena pri nezdvojení (lop namiesto loop).
- Substitúcia (ohoj namiesto ahoj, r = 0.3):
  - Odmena za posunutie (441 namiesto 411).
  - Odmena pri podobnosti znakov (v1na - namiesto vlna).
  - Odmena alebo sankcia za *fat finger distance* (ajoj namiesto ahoj).
- Pri kategórii *všeobecný reťazec* odmena ak majú podobnú fonetiku.
- Pri kategórii *doménové meno* odmena za chybu pri separátore (domenacom namiesto domena.com).
- Pri kategórii *všeobecné číslo* odmena za zrkadlenie (123 namiesto 321).
- Odmena za kapitalizáciu (AHOJ namiesto ahoj). (15b)

5.4 Zhrnutie

V kapitole bol opísaný návrh systému na opravu preklepov a jeho časti. Na začiatku sa definovali typy dát s akými sa bude pracovať. Stanovili sa potrebné údaje do konfiguračných súborov a určil sa formát slovníkov.

Kvôli zjednoteniu rôznych formátov bude systém obsahovať konvertor, ktorého výstupom bude jednotný formát. S ním už ďalej bude pracovať časť programu pre detekciu a opravu chybného slova. Oprava sa bude vyberať z kandidátov na základe ohodnotenia vytvorenej heuristiky založenej na typoch chýb.

Kompletný návrh je možné vidieť na obrázku 5.9. Pre fázu učenia je pridaný výstup slovníka, ktorý sa využije pri aplikovaní heuristík a hľadaní kandidátov. Týmto spôsobom je možné zo správnych dát určiť aké dáta sa prirodzene v sieti používajú.



Obr. 5.9: Celkový návrh programu

Kapitola 6

Implementácia

V tejto kapitole bude opísaná implementácia navrhnutého programu. Implementačná časť je napísaná v skriptovacom programovacom jazyku Python 3.6.

Najprv bude vysvetlený spôsob spustenia programu a všetky jeho argumenty príkazového riadku v sekcii 6.1. V nasledujúcej sekcii 6.3 sú popísané kombinované kategórie, ďalej v 6.2 je vysvetlené fungovanie tried pre spracovanie vstupu na jednotný výstup. Sekcia 6.4 opisuje spôsob pri rozhodovaní akú dátovú štruktúru použiť pre vnútornú reprezentáciu slovníka. Sekcia 6.5 opisuje prácu so slovníkmi a ich triedy. Nakoniec sa kapitola venuje detekcii chybného slova a generovaniu kandidátov v sekcii 6.6 a výberu kandidátov v 6.7.

6.1 Spustenie programu

Program bude implementovaný ako konzolová aplikácia. Cesty k súborom s konfiguráciami a dátami je možné do programu dostať ako argumenty z konzoly. Vstupné dáta a formáty konfiguračných súborov sú podrobnejšie opísané v sekcii 5.1. Pri spustení programu sa špecifikuje či sa jedná o dáta pre naučenie slovníka alebo vyhľadávanie preklepov. Takisto je potrebné definovať typ dát, s ktorým sa bude pracovať. Je možné zvoliť medzi pcap, NetFlow a log súborom.

Program sa spúšťa pomocou súboru *ndsc.py*, názov je akronymom od *network data spelling corrector*. Nižšie sú uvedené všetky potrebné parametre.

ndsc.py [-log/-flow/-pcap] [-learn/-diag] [cesta k dátam] [cesta ku konf. súboru pre vstup] [cesta k primárnemu slovníku]

Ako prvý je prepínač pre typ dát NetFlow (-flow), súbor s paketmi (-pcap) alebo záznamy aktivity (-log). Potom určíme či chceme z bezchybných dát vytvoriť primárny slovník (-learn) alebo budeme chybu hľadať (-diag). Za tým si špecifikujeme cestu k vstupným dátam, môže odkazovať na priečinok, ktorý ich obsahuje vo viacerých súboroch. Už zostáva len cesta ku konfiguračnému súboru a primárnemu slovníku. Primárny slovník sa vo fáze učenia vytvorí a vo fáze diagnostiky sa môže, ale nemusí použiť.

Príklad na spustenie, v ktorom sa analyzujú vstupné dáta vo forme paketov z priečinka *pcap* a výstup je uložený do primárneho slovníka *dic2.txt*:

python3 ndsc.py -pcap -learn files/in/pcap files/config/confPcap.txt files/dic2.txt

V súbore *ndsc.py* je riešený celý beh programu, od spracovania argumentov po obdržanie výstupu. Vykonávajú sa v ňom volania potrebných tried. Okrem konfiguračného

súboru, ktorého cesta sa zadáva ako parameter programu existuje ďalší konfiguračný súbor. Obsahuje dodatočné informácie o slovníkoch. Je umiestnený vždy v priečinku so súborom *ndsc.py* a má názov *config.txt*. Je možné ho upravovať, ale nie premiestňovať. Vďaka tomu nie je potrebné zadávať cestu k nemu ako ďalší argument programu.

6.2 Konvertor

Modul konvertor obsahuje sadu tried, ktoré sa využívajú pri načítaní vstupov a konvertovaní ich obsahu do požadovanej formy. Jednotlivé triedy modulu a ich funkcionalita:

- Input - pomocou tejto triedy sa načíta vstup, či už konfiguračného súbora alebo dát vo formáte CSV alebo log.
- ConfigParser - spracuje konfiguračný súbor pre vstupné dáta a načíta ho do vnútornej reprezentácie.
- Converter - implementuje spoločné a základné metódy pre načítanie, pridávanie a získanie dát pre všetky typy konvertorov. Metóda na spracovanie dát je ponechaná na zadefinovanie v odvodených triedach.
- TsharkExport - slúži pre konvertovanie vstupného pcap súboru do JSON podoby. Pre tento úkon volá konzolovú aplikáciu TShark, ktorá dáta vyfiltruje podľa údajov z konfiguračného súboru pre vstup. Volanie externého programu je umožnené vďaka Python modulu subprocess[1], ktorý dovoľuje vytvárať nové procesy a obdržať z nich výsledok, napojiť sa na ich vstupy, výstupy a obdržať chybové kódy a hlášky. Metóda zobrazená na obrázku 6.1 má názov *terminal_cmd*. Na vstup prijme príkaz pre konzolu a vráti výsledok po ukončení procesu.

```
@staticmethod
def terminal_cmd(cmd):
    try:
        out = subprocess.check_output(cmd, shell=True, stderr=subprocess.STDOUT)

        if out is not None:
            out = out.decode()

        return out

    except subprocess.CalledProcessError as error:
        raise Exception('ERROR:' + str(error))
```

Obr. 6.1: Metóda pre vykonanie konzolového príkazu

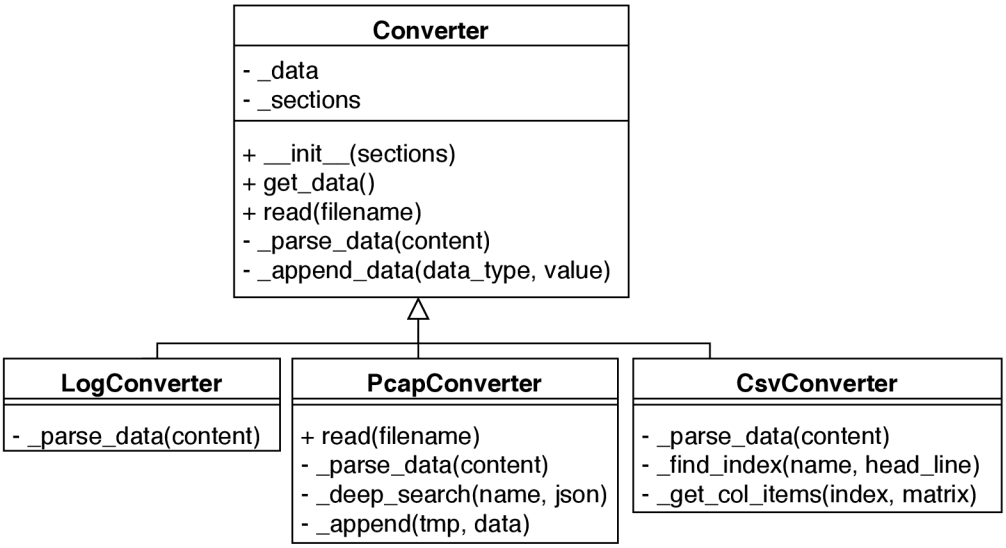
Pripravené vlastnosti a správanie z triedy Converter implementujú triedy:

- LogConverter - spracuje vstup z log súboru na základe regulárnych výrazov v konfiguračnom súbore. Pre prácu s regulárnymi výrazmi je použitý Python modul *re*.
- PcapConverter - využíva statickú metódu triedy *TsharkExport* pre konvertovanie pcap súboru do JSON.

[1]Dokumentácia pre modul subprocess https://docs.python.org/3.6/library/subprocess.html

- CsvConverter - spracuje vstup z NetFlow dát vo formáte CSV. Pre určenie použitého oddeľovača vo vstupnom súbore je použitá trieda Sniffer z Python modulu csv. Tá zanalyzuje vstupné dáta a určí, ktorý znak je oddeľovač hodnôt.

Závislosť a dodatočné metódy týchto tried vidíme na obrázku 6.2.



Obr. 6.2: UML triedny diagram pre triedy typu konvertor

6.3 Kombinované kategórie

V podsekcii 5.1.3 bol popísaný návrh pre zadefinovanie kombinovaných kategórii v konfiguračnom súbore. Súbor bol navrhnutý vo formát YAML. Na prácu s týmto formátom je použitá knižnica PyYAML, ktorá načíta konfiguračný súbor do vnútorných štruktúr. Trieda *MainConfig* obsluhuje úkony okolo konfiguračného súbora.

Kvôli kombinovaným kategóriám vznikla trieda *CombinedCategory*. Jej úlohou je zo zadaného slova určiť z akých kategórii sa skladá. Keď už jednotlivé časti majú kategóriu priradenú upravujú sa samostatne. Kategórie sa určujú na základe zadefinovaných formátov a separátorov. V jednotlivých formátoch sa vyhľadá separátor a ak sa tam nachádza, tak sa to v tom mieste rozdelí. S využitím regulárnych výrazov je možné rozlíšiť o akú kategóriu sa jedná ak má viac základných kategórii podobný formát. Pri vytváraní inštancií kombinovaných kategórii sa nové kategórie iba po dobu behu programu pridajú k základným kategóriám, preto pri práci so slovníkom, ktorý obsahuje kombinované kategórie je potrebné ich mať zapísané v konfiguračnom súbore.

6.4 Výber dátového typu pre slovník

Pri výbere dátového typu pre programovú reprezentáciu slovníka som využil rôzne existujúce knižnice a vykonal porovnanie. Porovnával som bloom filter, ternárny vyhľadávací strom, štruktúry ponúkané programom Python a to zoznam (list) a množinu (set), ktorá je implementovaná pomocou hash tabuľky. Ako zdroj slov do slovníka som použil dokument s 253 886 jedinečnými reťazcami.

Dátový typ	Priemerný čas inicializácie [ms]	Priemerný čas potrebný pre vyhľadanie [ms]
Ternárny vyhľadávací strom	13190.0	56.62
Bloom filter	7943.2	36.30
Množina	120.4	0.39
Zoznam	95.8	39.53

Tabuľka 6.1: Porovnanie dátových typov

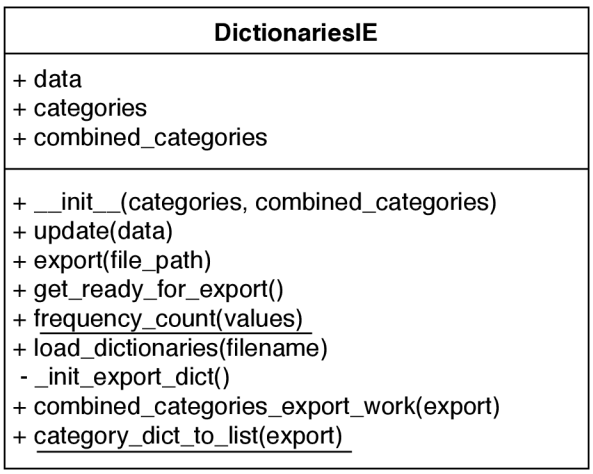
V tabuľke 6.1 je zobrazený výsledok porovnávania spomenutých dátových štruktúr. Bolo vykonaných 50 iterácií inicializácie dátového typu a vyhľadania v dátovej štruktúre. Vyhľadávalo sa 1,179 slov a pri inicializácii sa menil zdrojový súbor, aby sa predišlo urýchľovaniu pomocou cache procesora. Časové trvanie iterácií bolo zmerané a spriemerované.

V tabuľke sú zaznamenané najlepšie hodnoty pre možnosť vyhľadania v danej dátovej štruktúre. To znamená, že napríklad pre množinu bol vykonaný prienik množín a tým sa určilo čo sa nachádza v slovníku a druhým spôsobom je iteratívne vyhľadávanie v množine, ale to bolo o niečo pomalšie, tak uvádzam čas len prvej varianty. Vo výsledku je zahrnutý aj čas potrebný pre vytvorenie novej množiny z hľadaných slov. Takisto pre zoznam bolo vykonaných viac variácii vyhľadávaní, no najviac sa osvedčilo binárne vyhľadávanie. V čase pre vyhľadávanie v zozname je zahrnuté zoradenie zoznamu.

Priestorová zložitosť medzi jednotlivými štruktúrami nebola porovnaná, ale bola skontrolovaná iba pre časovo najrýchlejšiu množinu. Vyskúšal som súbor o objeme slov 235 886, ktoré mali vo formáte txt veľkosť 2,5MB. Po načítaní v jazyku Python do štruktúry množina použitím modulu *memory_profiler* na overenie priestorovej zložitosti sa ukázalo, že výsledná štruktúra zaberá 38.8MB. Na súčasné parametre PC komponentov to nie je hodnota, ktorá by ma odradila od jej použitia. Na základe týchto zistení som zvolil množinu ako dátový typ pre implementáciu slovníka.

6.5 Slovníky

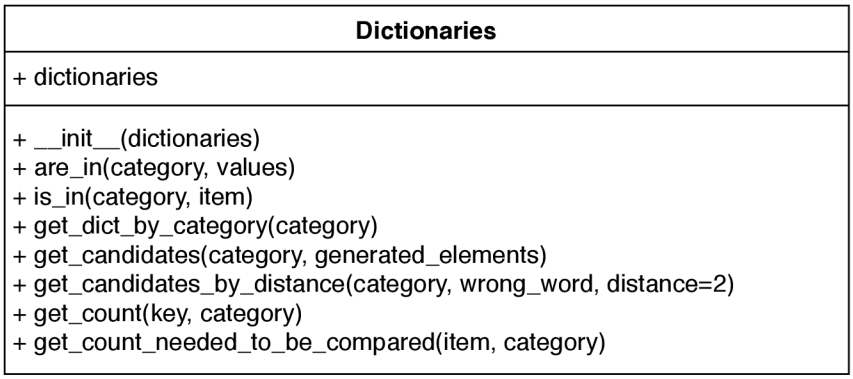
Modul *dictionaries.py* obsahuje triedy na prácu s primárnymi a sekundárnymi slovníkmi. Primárne slovníky sú obsluhované triedou *DictionariesIE*. UML diagram triedy so všetkými



Obr. 6.3: UML triedny diagram triedy DictionariesIE

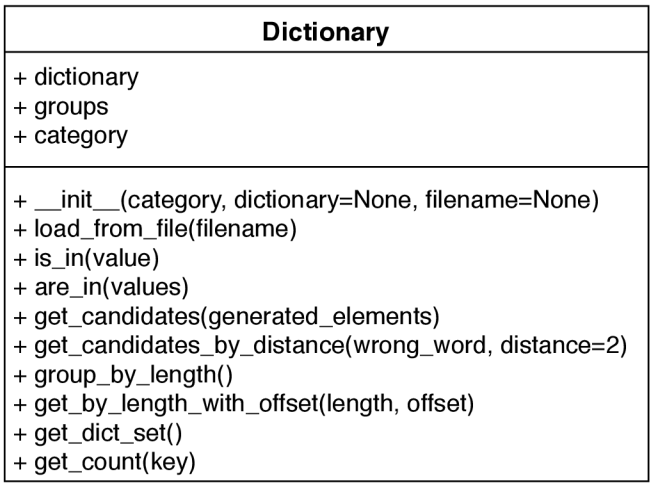
atribútmi a metódami je na obrázku 6.3. Jej úlohou je vytvorenie primárneho slovníka vo fáze učenia.

Počas načítavania dát z rôznych súborov sa ich výstupy zbierajú v atribúte *data*. Predtým než sa dáta exportujú do súboru sa vykoná kontrola na frekvenčný výskyt. Táto informácia sa využije pri výbere vhodného kandidáta a je uložená v primárnom slovníku spolu s príslušným slovom. Dáta sú uložené vo formáte JSON. Vo fáze diagnostiky, je možné pomocou tejto triedy načítať primárne slovníky do vnútornej reprezentácie a tou je trieda *Dictionary*. Pri načítavaní sa používa externý Python modul *json*, ktorý deserializuje informácie zo súboru.



Obr. 6.4: UML diagram pre triedu Dictionaries

Aby nebolo potrebné vždy riešiť hľadanie konkrétneho slovníka vznikla trieda *Dictionaries*. Vlastnosti tejto triedy sú znázornené na obrázku 6.4. Jej jediný atribút je vlastne zoznam slovníkov a to buď primárnych alebo sekundárnych. To znamená, že program vo fáze diagnostiky bude obsahovať 2 inštancie triedy *Dictionaries*. Napríklad pri potrebe vykonať operáciu nad jedným z primárnych slovníkov ho vyberie na základe kategórie, zavolá potrebnú operáciu a vráti výsledok.



Obr. 6.5: UML diagram pre triedu Dictionary

Slovník obsahuje informáciu do akej kategórie spadá a aké dáta obsahuje. Dáta sú uložené v dátovom type množina a pre vyhľadanie či sa v ňom dáta nachádzajú sa vykoná

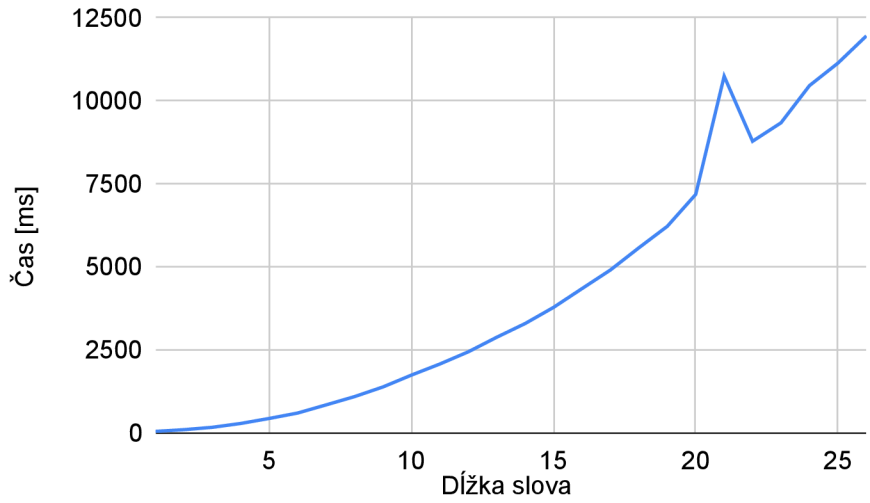
prienik množiny slovníka a slov, ktoré sa kontrolujú. Na tento účel slúžia metódy *is_in a are_in*. Intuitívne jedno kontroluje či jeden prvok sa nachádza v slovníku, druhé kontroluje prítomnosť viacerých prvkov naraz. Celý opis triedy je obsiahnutý v obrázku 6.5.

6.6 Detekcia chybného slova a získanie kandidátov

Detekcia chybného slova je riešená pomocou triedy *Dictionaries*. V nej sa pristúpi k správnemu slovníku a overuje sa prítomnosť slov operáciou prieniku množín. V prípade ak sa slová nenašli v primárnom slovníku nasleduje kontrola na prítomnosť v sekundárnom.

Ďalej je potrebné nájsť kandidátov na náhradu za chybné slovo. Najprv som pre generovanie kandidátov vytvoril funkciu, ktorá zo vstupu zoberie chybné slovo a vykoná nad ním reverznú operáciu pre všetky možné kombinácie štyroch základných chýb - vloženie, zmazanie, nahradenie znaku a zámena dvoch znakov. Výsledkom sú kandidátne slová v editačnej vzdialenosti 1. Pre každé slovo vo výsledku sa opäť vykoná generovanie slov a zlúčením všetkých týchto vzniknutých slov máme kandidátov do editačnej vzdialenosti 2. Ďalej sa činnosť už neopakuje, lebo väčšina chýb sa stane do tejto vzdialenosti. Na druhej strane je časovo náročné generovať slová do väčšej vzdialenosti.

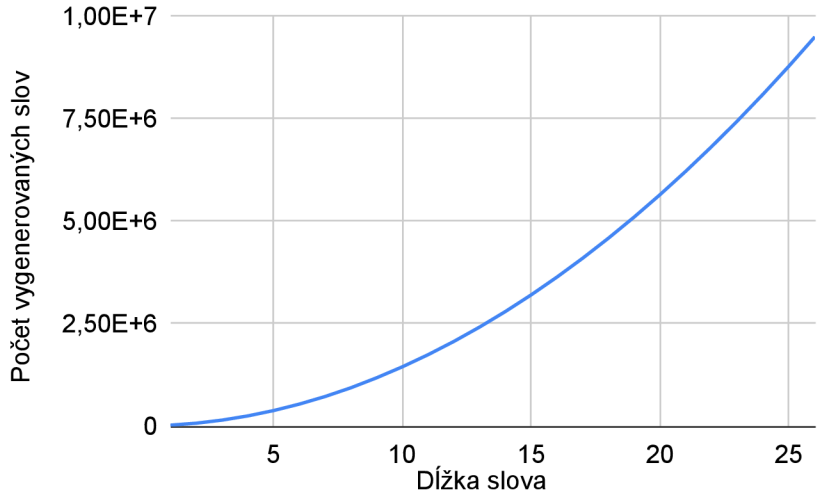
Tento spôsob generácie kandidátov bol opísaný Norvigom[31]. V niektorých článkoch uvádzali toto riešenie ako referenčné a porovnávali s ním svoje namerané hodnoty[13].



Obr. 6.6: Závislosť času od dĺžky slova pre vygenerovanie kandidátov

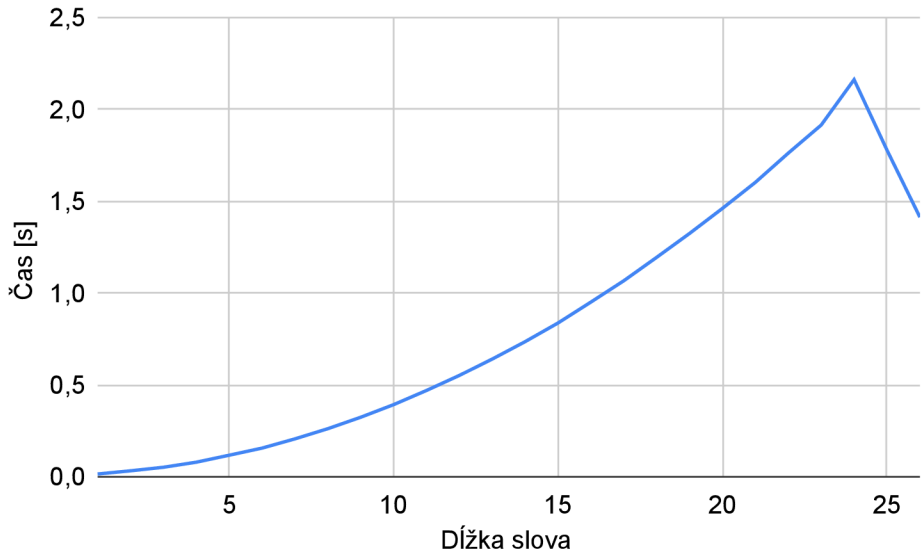
Po vygenerovaní kandidátov je potrebné overiť, ktorí sú valídni kontrolou prítomnosti v jednom zo slovníkov. Po tomto kroku už máme k dispozícii zoznam kandidátov a môžeme vyberať toho najvhodnejšieho. Problém tejto metódy je, že už pri slovách o dĺžke 7 vygenerovanie kandidátov trvá sekundu. Čím bolo slovo dlhšie, tým viac kandidátov vzniklo, lebo bolo možné vykonať viac operácii. Na obrázku 6.6 vidíme, že slovo o dĺžke 25 znakov potrebuje na vygenerovanie kandidátov približne 11 sekúnd. Pre takisto dlhé slová sa vytvorí skoro milión možných kandidátov, to je znázornené na obrázku 6.7.

Z týchto dôvodov nebol tento spôsob dostačujúci a musel som nájsť iné riešenie. Rozhodol som sa, že slovník bude rozdelený podľa dĺžky slov a kandidáti sa budú hľadať iba v určitom dĺžkovom rozmedzí. Táto funkcionalita je implementovaná v triedach pre prácu so slovníkmi a výsledok dostaneme zavolaním metódy s názvom *get_candidates_by_distance*.



Obr. 6.7: Závislosť počtu vygenerovaných kandidátov od dĺžky slova

Je možné nastaviť prípustnú vzdialenosť nepovinným parametrom, ktorý je prednastavený na hodnotu 2. Trieda *Dictionary* v atribúte *groups* obsahuje slová rozdelené podľa dĺžky. Nad ním sa bude získavať výsledok. Po vypočítaní editačných vzdialeností dostaneme kandidátov. Na určenie podobnosti slov sa používa Damerau-Levenshtein algoritmus spomenutý v sekcii 3.4.2.



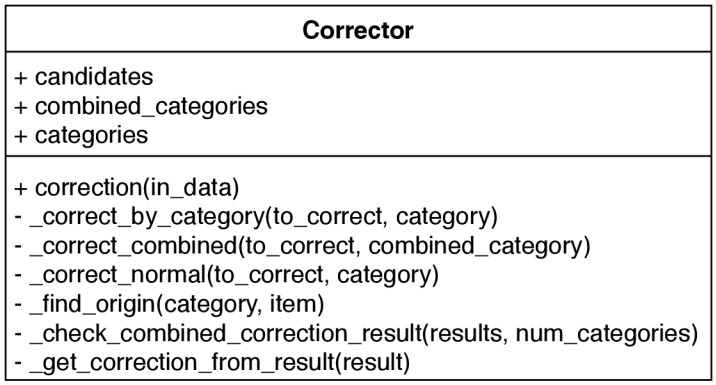
Obr. 6.8: Závislosť času a dĺžky slova pre nájdenie kandidátov v usporiadanom slovníku

Pre porovnanie jednotlivých spôsobov hľadania kandidátov je na obrázku 6.8 graf s novými hodnotami vyhľadania kandidátov. Ukazuje, že slovo o dĺžke 24 znakov má nájdených kandidátov za približne 2 sekundy. V prvej verzii v najhoršej možnej vyobrazenej situácii táto operácia trvala okolo 11 sekúnd. Dĺžka použitého slovníka bola 26000 slov s rovnomerným rozložením podľa dĺžky. To znamená, že každá dĺžka slova od 1 po 26 mala zastúpenie

1000 slov. Rýchlosť tohto riešenia závisí od dĺžky slova a počtu slov podobnej dĺžky v slovníku. Zoskupenie slov podľa dĺžky je rýchle a nepovšimnuteľné na dĺžke behu programu. Na základe týchto informácii je možné skonštatovať, že toto riešenie je oveľa rýchlejšie pri použití menšieho slovníka. Na konci grafu je časový pokles z dôvodu, že sa kontrolujú aj kratšie aj dlhšie slová do vzdialenosti 2. Slovo o dĺžke 26 nemalo k dispozícii slová o dĺžke 27 a slová o dĺžke 28.

Hľadanie chybných slov, generovanie kandidátov a nájdenie vhodného kandidáta je v réžii triedy *Corrector*, ktorá využíva pre jednotlivé úkony príslušné triedy. Vďaka triede *Candidates* obdrží potencionálne náhrady nájdené v primárnych a sekundárnych slovníkoch. Pre vytvorenie inštancií triedy *Candidate* sa používa trieda *CandidateFactory*. Vytvárajú sa obyčajní kandidáti alebo kandidáti z primárneho slovníka. Pri kandidátoch z primárneho slovníka vieme, že boli niekedy použitý v sieťovej komunikácii a počet, koľko krát sa v nej vyskytli. Túto informáciu využijeme pri odmeňovaní kandidátov.

Na obrázku 6.9 je trieda *Corrector*. Obsahuje viacero metód, z ktorých najdôležitejšou je *correction*. Jej vstupom je jednotný výstup z konvertora a pomocou nej sa obdržia chybné slová s opravou.



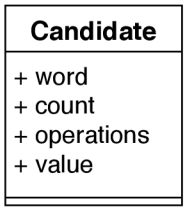
Obr. 6.9: UML diagram triedy Corrector

Opravu slova kombinovanej kategórie je potrebné rozlíšiť od obyčajnej kategórie, to zabezpečuje metóda *correct_by_category*. Je to potrebné riešiť z jednoduchého dôvodu, kombinovaná kategória pozostáva z viacerých obyčajných kategórii a oprava funguje princípom rozdelenia slova na časti. Každá časť patrí do príslušnej základnej kategórie a hľadá sa pre ňu oprava. Na tento účel slúži metóda *_find_origin*, ktorá si vygeneruje kandidátov a volá ďalšiu triedu pre aplikovanie heuristík. Ak sa našli nejaký kandidáti, výsledkom je najlepšie ohodnotená oprava.

6.7 Ohodnotenie kandidátov

V module *candidates.py* sa nachádzajú triedy *Candidate* a *CandidateSelection*. Prvá trieda je reprezentáciou kandidáta na opravu. Je tvorená atribútmi pre slovo, frekvenčný výskyt, ohodnotenia a zoznamu základných operácii, ktoré sa museli na chybnom slove vykonať, aby vznikol kandidát, obrázok 6.10.

CandidateSelection pracuje so zoznamom takýchto kandidátov. Na začiatok je potrebné určiť vykonané operácie pre každého kandidáta. Tie sú určené pomocou triedy *WordDistance* z iného modulu, ktorá okrem určenia operácii implementuje rôzne spôsoby na vyjadrenie podobnosti 2 slov číslom. Na základe týchto operácii a navrhnutému systému odmien



Obr. 6.10: UML diagram triedy Candidate

v podsekcii 5.3.1 sú kandidáti ohodnotení. Trieda CandidateSelection obsahuje metódu *apply_metrics*, ktorá ohodnotí každého kandidáta a vráti buď jedného alebo v prípade zhodného najvyššieho ohodnotenia viacerých ako výsledok. Na kontrolu fonetiky sa využíva Python knižnica Metaphone, ktorá vychádza z algoritmu Double Metaphone.

Podobne vyzerajúce znaky sú v predpripravenej triede prístupné pomocou Python zoznamu zoznamov. Vnútorný zoznam obsahuje podobne vyzerajúce prvky. Nie je použitá implementáciu *fat finger distance* ako ho opísali v pôvodnom článku [29]. Namiesto toho sa využíva Python dátovú štruktúru dictionary, kde kľúčom je znak a hodnota je zoznam okolitých znakov. Ak daná operácia bola z okolia znaku, tak sa navýši hodnotenie kandidáta.

Kapitola 7

Testovanie

Táto kapitola sa venuje testovaniu vytvoreného systému. Cieľom je potvrdiť správnosť vytvoreného riešenia a ukázať funkcionalitu na vybranej vzorke dát, overiť rýchlosť riešenia. V prípade odhalenia nedostatkov je možné ich zapracovanie.

V prvej sekcii 7.1 je opísaný postup pri rozhodovaní či podporovať filtrovanie v TShark. V ďalšej sekcii 7.2 je overovaná implementovaná heuristika, kde sa overuje či sa správa tak ako bola navrhnutá. Pri implementácii sa udiali zmeny v generovaní kandidátov voči návrhu a preto sa v sekcii 7.3 podrobnejšie testuje nová aj predchádzajúca metóda. Na konci kapitoly sekcia 7.4 zobrazuje prehľad behu programu pre rôzne prípady.

7.1 Podpora filtra pre TShark

Oproti ostatným typom zdrojových dát sú pcap súbory veľké. Obsahujú veľké množstvo informácii a tak ich spracovanie môže dlho trvať. Jednou z možností ako túto činnosť urýchliť je použitie filtrovania dát v programe TShark, ktorý sa používa na spracovanie týchto súborov.

Použitie filtrovania	Veľkosť pcap	Čas behu TShark [s] programu [s]	Beh zvyšku	Čas spolu [s]
nie	40MB	3,46	0,19	3,65
	111MB	7,9	0,51	8,41
	159MB	10,98	0,82	11,8
	164MB	11,63	0,83	12,46
	216MB	15,05	1,08	16,13
áno	40MB	0,93	0,002	0,94
	111MB	2,045	0,003	2,05
	159MB	2,9	0,002	2,9
	164MB	3,27	0,002	3,27
	216MB	4,13	0,002	4,13

Tabuľka 7.1: Meranie času behu programu s filtrom a bez neho

Pre rozhodnutie či pre urýchlenie rýchlosti spracovania podporovať filtrovanie týchto dát pomocou filtrov, ktoré si program definuje boli vykonané merania. Prvé meranie spustilo program bez filtrovania. Program sa spúšťal na piatich pcap súboroch od veľkosti 40MB po 216MB v móde učenia. Meranie je zaznamenané v tabuľke 7.1 pod záznamom bez použitia

filtrovania. Časovo to boli vždy sekundy behu programu. Pri najmenšom súbore TShark spracovával dáta 3,46 sekúnd a pri najväčšom 15,05 sekúnd.

Druhé meranie bolo vykonané s tým, že TShark dostal filter pre ignorovanie vybraných dát ako parameter. Meranie je zaznamenané v tabuľke 7.1 na riadku s použitým filtrovaním. Program trval od necelej sekundy do 4,13 sekúnd. To je v porovnaní s prvým meraním viac ako trojnásobné zlepšenie. Z tohto dôvodu bola pridaná podpora pre zadanie filtra. Najlepšie je pustiť program na predfiltrovaných dátach, ale tam sa zaberie čas zvlášť filtráciou.

7.2 Overenie heuristiky

Táto sekcia je zameraná na overenie správnosti vytvorenej heuristiky pre ohodnotenie kandidátov na opravu slova. Na začiatok sa overí či vytvorená heuristika sa správa podľa očakávaní a ohodnotí vytvorené preklepy tak ako sa od nej očakáva. Cieľom nie je analyzovať hodnotu samotného ohodnotenia, ale poradie jednotlivých kandidátov. Preto boli aplikované rôzne chyby na slová a nad nimi sa spustila metóda pre ohodnotenie.

Počet operácii	Operácie	Chybné slovo	Ohodnotenie
X	Kapitalizácia	GOOGLE.COM	15,58
1	Náhrada za podobný znak	g0ogle.com	11,62
1	Chyba v súvislosti s bodkou	googlecom	10,5
1	Náhrada v FF vzdialenost	giogle.com	10,22
1	Náhrada s posunutím	ggogle.com	9,62
1	Vloženie s pridaním opakovaného znaku	gooogle.com	9,5
1	Zmazanie a nezdvojenie	gogle.com	9,5
1	Vloženie a zdvojenie	ggoogle.com	8,5
1	Vloženie a slovo je prefix	google.comp	8,5
1	Zmazanie	goole.com	8,5
1	Vloženie v FF vzdialenosti	goople.com	8,22
1	Náhrada nie v FF vzdialenosti	gwogle.com	8,22
1	Transpozícia	ogogle.com	7,6
1	Vloženie nie v FF vzdialenosti	gooqgle.com	7,5
2	Náhrada v FF vzdialenosti	giigle.com	6,44
2	Vloženie v FF vzdialenosti	gopople.com	5
2	Náhrada, 1 v FF vzdialenosti a 1 nie	gpqgle.com	4,44
2	Vloženie, 1 v FF vzdialenosti a 1 nie	ghoogwle.com	3
2	Zmaž	ggle.com	3
2	Náhrada nie v FF vzdialenosti	gyygle.com	2,44
2	Transpozícia	oggole.com	1,2
2	Vloženie nie v FF vzdialenosti	goqoqgle.com	1

Tabuľka 7.2: Ohodnotenie rôznych chýb pre doménu google.com použitím vytvorenej heuristiky

Prvý test heuristiky je zameraný na ohodnotenie viacerých kandidátov jedného chybného slova. Cieľom je zistiť, či poradie ohodnotených kandidátov adekvátne reflektuje pravdepodobnosť typu preklepu, ktorý nastal. Takže napríklad, jeden vložený znak z fat finger vzdialenosti by mal mať lepšie ohodnotenie ako zmazanie dvoch znakov. Výsledok je za-

znamenaný v tabuľke 7.2, ktorá zobrazuje počet operácii, typ operácie vykonanej na slove, chybné slovo a jeho ohodnotenie. Položky v tabuľke sú zoradené podľa ohodnotenia. Na prvom mieste je kapitalizácia, ktorá má "X" namiesto počtu operácii z dôvodu, že nie je overovaná podľa editačnej vzdialenosti. Na posledných miestach medzi chybami s jednou a dvomi operáciami je transpozícia. To z dôvodu stanoveného pomeru, ktorý ju ťahá nadol. Je možné konštatovať, že ohodnotenie je zoradené v súlade s navrhnutou heuristikou.

Druhým testom bolo overenie či vytvorená heuristika vyberá správnych kandidátov. Použili sa dáta sprístupnené od Roger Mitton na jeho webovej stránke[1]. Nachádza sa tam viacero súborov obsahujúcich správne slová s ich chybnými verziami zistených z rôznych zdrojov. V prípade dát, ktoré som použil, pochádzali z wikipédie a obsahovali 2455 preklepov 1922 slov.

Počet slov	1000
Počet nenájdených v slovníku	991
Počet určených za správne	9
Percento určených za správne	0,9%
Počet správne opravených	971
Počet prípadov s viacerými možnosťami	4
Percento správne op. z nenájdených slov	97,98%
Percento správne op. celkovo	97,1%
Priemerný počet kandidátov na slovo	2,31
Veľkosť použitého slovníka	797641 domén

Tabuľka 7.3: Testovanie heuristiky

Pre tento test som vytvoril skript, ktorý na zoznam známych domén aplikoval chyby z preklepov v bežnom texte. Pre domény sa hľadalo správne napísané slovo v zdroji dát, ktoré bolo ich podreťazcom. Správne napísané slová majú zoznam svojich chybne napísaných verzií. Potom bol podreťazec nahradený jednou z chybných podôb slova. Napríklad slovo *guard* malo uvedený preklep *gaurd*. Po aplikovaní preklepu na doménu *theguardian.com* vznikla nová doména s preklepom *thegaurdian.com*. Týmto spôsobom bolo vytvorených 1000 rôznych domén obsahujúcich preklep a uložených do súboru. Slovník pre kontrolu či je doména správna alebo nie obsahoval 797641 domén. Chybné slová obsahujú iba chyby do editačnej vzdialenosti 2. Prvý krát, sa generovali slová nezávisle na editačnej vzdialenosti. V dôsledku to znamenalo, že 2% slov nemohli byť opravené a 6,7% vytvorených slov bolo inou validnou doménou, takže sa hneď našli v slovníku.

V tabuľke 7.3 sú výsledky po spustení programu. Kvôli veľkosti slovníku, v ktorom sa hľadali kandidáti existovalo 9 domén s vytvoreným preklepom. To znamená, že z jednej správnej domény aplikovaním preklepu vzniklá iná správna doména. Zo zvyšných 991 slov bolo správne opravených 971 prípadov. Z tohto celkového počtu majú iba 4 chybné slová viac ako jednu ponúknutú náhradu. To sa stane v situácii, keď je viacero kandidátov rovnako ohodnotených. Na jedno chybné slovo priemerne vychádza 2,31 kandidátov. Je možné, že slovo malo iba jedného kandidáta a to bol výsledok. Zvyšných 20 domén bolo chybne opravených. Napríklad doména *udemy.com* mala preklep *udemi.com*, ktorý bol opravený na *udemu.com* z dôvodu, že nahradenie písmena *u* písmenom *i* je viac pravdepodobné, keďže sa nachádza na QWERTY klávesnici hneď vedľa.

[1] http://www.dcs.bbk.ac.uk/~ROGER/corpora.html

Počet slov	1000
Počet nenájdených v slovníku	991
Počet určených za správne	9
Percento určených za správne	0,9%
Počet správne opravených	891
Počet prípadov s viacerými možnosťami	44
Percento správne op. z nenájdených slov	89,9%
Percento správne op. celkovo	89,1%
Priemerný počet kandidátov na slovo	8,8
Veľkosť použitého slovníka	797641 domén

Tabuľka 7.4: Testovanie heuristiky na Birkbeck dátach

Podobné meranie s vytvorenými chybnými doménami bolo vykonané s inými zdrojmi preklepov. Taktiež pochádzali z rovnakej stránky od Rogera Mittona s názvom *Birkbeck*. Zdrojové dáta s preklepmi obsahujú 36133 chýb 6136 slov. Výsledok je zaznamenaný v tabuľke 7.4. Nevýhodou týchto dát je, že nevznikli písaním na klávesnici, ale boli zozbierané z písaných testov, z chýb vzniknutých voľným písaním. Celková úspešnosť je 89,1%. Priemerný počet kandidátov na slovo bol v tomto prípade 8,8. Keď porovnám prvé vytvorené chybné domény z preklepov z wikipédie s týmito, tak bolo použitých 128 rovnakých domén, ale zhodujú sa iba v dvoch chybných slovách.

Pre názornú ukážku vytvorenej chybnej domény a jej nájdenej opravy bolo vybratých pár správne opravených domén. Výber je možné vidieť v tabuľke 7.5. Boli vyberané s cieľom ukázať rôzne chyby, ktoré sú ohodnotené. V dvoch prípadoch ide o znak z fat finger vzdialenosti a to raz pri vložení a raz pri náhrade. V jednom prípade sa jedná o transpozíciu a často sa v dátach vyskytovalo zdvojenie znaku.

Oprava	Chybné slovo	Ohodnotenie
twitch.com	twich.tv	Odmena za jednu chybu, sankcia za zmazanie znaku, 8.5 (-4? vsetky)
panda.tv	panbda.tv	Odmena za jednu chybu, odmena za Fat finger pri vložení znaku, sankcia za operáciu, 9.5
chase.com	chasr.com	Odmena za jednu chybu, odmena za Fat finger pri náhrade znaku, sankcia za operáciu, 10.22
ilovepdf.com	ilvoepdf.com	Odmena za jednu chybu, sankcia za transpozíciu, 7.6
weather.com	weatherr.com	Odmena za zdvojenie pri vložení, sankcia za operáciu vloženia, 8.5
google.co.kr	google.comkr	Chyba na bodke, sankcia za to že náhrada nie je z Fat finger vzdialenosti a za operáciu samotnú, 10.22
redd.it	redd.itt	Odmena za jednu chybu, opravné slovo je prefix, zdvojenie pri vložení a sankcia za operáciu vloženia, 10.5

Tabuľka 7.5: Príklad vybranej vzorky chybných domén s opravou

Počas overovania heuristík sa mi podarilo odhaliť niektoré chyby. Napríklad kontrola na fat finger neoverovala vzdialenosti na numerickej klávesnici. Celkovo je náročnejšie overiť

takto vytvorenú heuristiku. Bola vytvorená na základe vyskytujúcich sa chýb a štatistických znalostí. Na prvých dátach obstála na 97%, v druhom prípade to bolo 89% a v nezaznamenanej situácii keď boli akceptované slová s väčšou editačnou vzdialenosťou to bolo 89,5% celkovo a 95% úspešnosť z nenájdených slov.

Posledný test funkčnosti bol zameraný na detekciu a opravu preklepov v ďalších kategóriách dát. V nasledujúcej tabuľke 7.6 sú zobrazené príklady preklepov a ich opráv aj pre iné kategórie ako je doména. V tomto teste sa nepozerá na poradie jednotlivých kandidátov, ale či najlepší kandidát odpovedá správnemu slovu. V každom z prípadov ohodnotenia bola udelená sankcia za operáciu, ktorá v tabuľke nie je uvedená. Zaujímavou hodnotou je dvojica slov "reel" a "real", ktoré zastupujú kategóriu všeobecného reťazca. V tejto kategórii sa porovnáva taktiež fonetika a za to bol kandidát odmenený. Inak je ešte zaujímavá hodnota používateľského mena, kde v chybnom prípade slovo "xbohus01" nekončí číslom jeden, ale písmenom "l".

Kategória	Oprava	Chybné slovo	Ohodnotenie:
IP	192.168.0.1	192.168.20.1	Odmena za jednu chybu, Fat finger pri vložení, 9.5
	147.229.100.1	147.229.10.1	Odmena za zmazanie opakujúceho sa znaku, jednu chybu, 9.5
Všeobecný reťazec	reel	real	Odmena za podobnú fonetiku a jednu chybu, 10.22
Používateľské meno	xbohus01	xbohus02	Odmena za jednu chybu vo Fat finger vzdialenosti, 10.22
	xbohus01	xbohus0l	Odmena za jednu chybu, znak v matici zameniteľných znakov, sankcia za Fat finger, 9.62
Port	6535	65535	Odmena za zdvojenie pri vložení, jedna chyba, 9.5

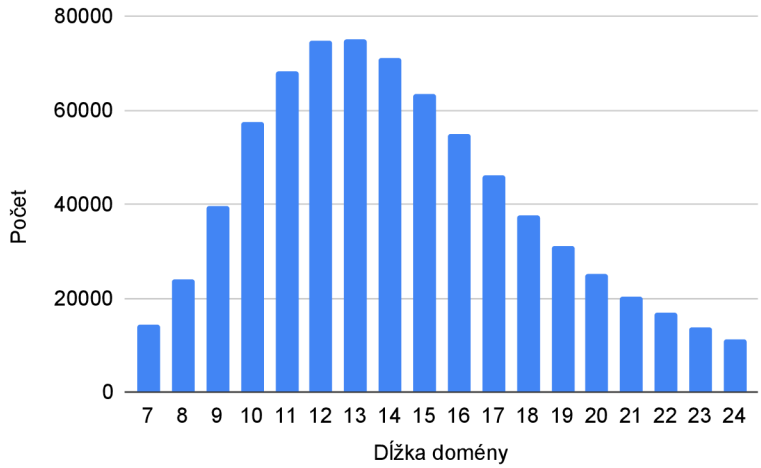
Tabuľka 7.6: Príklady preklepov a ich zvolených opráv aj pre iné kategórie

V prípade tohto testovania sa ukázalo, že heuristika je schopná ohodnotiť aj ostatné kategórie. Pravidlá sa používajú tie isté, ale napríklad pri všeobecnom reťazci je možné overiť navyše fonetiku. Preklepy boli vytvorené mnou a neprehľadával sa slovník pre nájdenie opravy, ale vyberalo sa zo zadaných kandidátov. Boli ukázané aj iné typy chýb, ktoré môžu nastať.

7.3 Overenie rýchlostí vytvorenia kandidátov

Kvôli rôznym implementáciám vytvorenia kandidátov je potrebné podrobnejšie porovnať jednotlivé metódy a overiť ich rýchlosť. Pre tvorbu kandidátov sa najprv používal prístup generovania, kde sa na slovo s preklepom reverzne aplikovali všetky možné kombinácie jednej z operácii vloženia, zmazania, nahradenia alebo transpozície. Po prvej aplikácii všetkých operácii vznikli slová o editačnej vzdialenosti 1 a po druhej aplikácii operácii na vzniknuté slová mali nové slová editačnú vzdialenosť 2. Pri dlhších slovách toto generovanie začalo zaberať značný čas a chcel som to urýchliť. Preto som sa rozhodol pre iný spôsob, ktorý tvorí kandidátov porovnaním editačnej vzdialenosti. Pre urýchlenie porovnávania je slovník

rozdelený podľa dĺžky slov. V rozdelenom slovníku sa hľadajú slová do editačnej vzdialenosti 2 v rovnako dlhých slovách a v slovách v rozmedzí do plus a mínus 2 znaky. V sekcii 6.6 bola metóda generovania porovnaná s metódou vyhľadania na slovníku o veľkosti 26000 slov s rovnomerným rozložením pre slová od dĺžky 1 po 26. V takejto situácii jasne dominovala metóda na základe porovnania editačnej vzdialenosti.

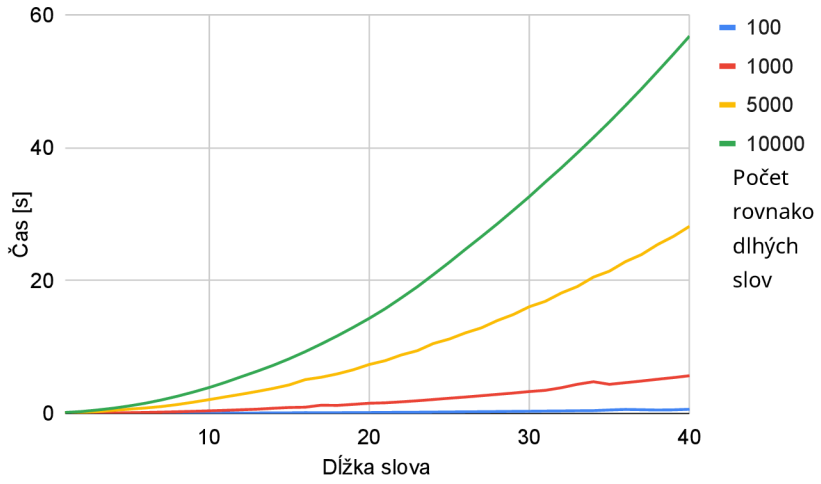


Obr. 7.1: Počet výskytu domén podľa ich dĺžky v zozname takmer milión domén

Pri overovaní heuristiky a práci so slovníkov o veľkosti 797641 domén vyhľadanie kandidátov novou metódou trvalo dlhý čas. Najhorší potrebný čas pre nájdenie kandidátov bol 52 sekúnd a preto chcem zistiť príčinu tak dlhej doby spracovania. Z dôvodu, že na tento spôsob vyhľadávania kandidátov má vplyv dĺžka slova a počet rovnako dlhých slov bolo analyzované aké rozloženie podľa dĺžky domén má súbor, na ktorom sa overovali heuristiky. Súbor bol zverejnený službou Amazon Alexa pred pár rokmi, ale to na meranie nebude mať vplyv. Výsledok je zaznamenaný v grafe na obrázku 7.1. Ukázalo sa, že domény o dĺžke 10 až 17 majú najväčšie zastúpenie. Všetky majú výskyt vyšší ako 40000. Keď sa hľadali kandidáti pre doménu o dĺžke 13 znakov bolo potrebné vykonať až 352000 porovnaní. To kvôli tomu, že sa hľadajú kandidáti aj v kratších a dlhších slovách. Na tejto vzorke dát bola zistená priemerná dĺžka domény 15 znakov. Ak by sa opravovali domény priemernej dĺžky, stále je potrebné pre jednu doménu vykonať 310000 porovnaní.

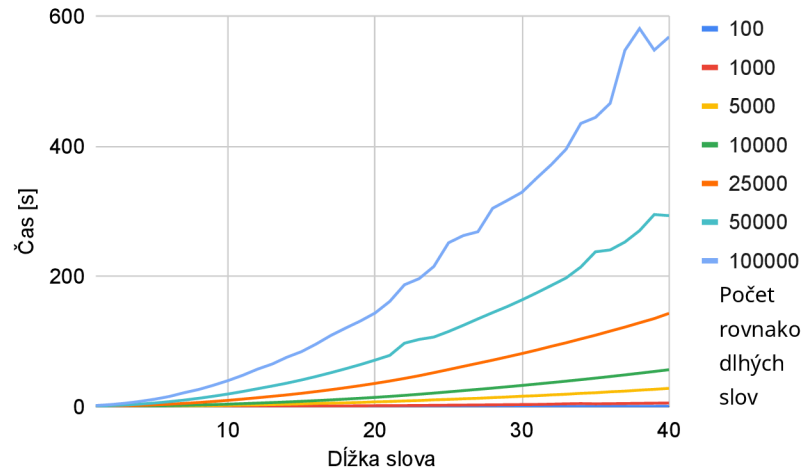
Na základe zistení koľko porovnaní bolo potrebné vykonať na priemerne dlhú doménu sa zisťovalo koľko času zaberie vyhľadanie kandidátov porovnávaním editačných vzdialeností v rôznom počte rovnako dlhých domén. Na tento účel bol vytvorený skript, ktorý naplnil slovník slovami o rovnakej dĺžke od 1 do 42 rovnakým počtom. Potom bol zmeraný čas potrebný pre vyhľadanie kandidáta. Pri rovnomernom naplnení slovníka to znamená, že kvôli spôsobu vyhľadávania je potrebné porovnať 5 krát viac slov ako je počet rovnako dlhých.

V grafe na obrázku 7.2 je zmeraný čas potrebný pre nájdenie kandidáta v rovnako dlhých slovách o počte 100, 1000, 5000, 10000. V poslednej situácii, keď počet rovnako dlhých slov bol 10000 ich slovník obsahoval dokopy 42000. Keďže sa vyhľadáva aj v kratších, aj dlhších slovách, aby nebol pokles na konci grafu pri slove o dĺžke 40 znakov boli pridané do slovníka slová aj s dĺžkou 41 a 42 znakov. Z nameraných dát je možné určiť, že pre priemernú dĺžku domény 15 znakov budú jednotlivé merania v závislosti na počte rovnako dlhých domén od najmenšieho po najväčší trvať 0,09, 0,9, 4,3 a 8,23 sekundy.



Obr. 7.2: Závislosť času na dĺžke slova a rôznom počte rovnako dlhých slov pri vyhľadávaní kandidátov podľa editačnej vzdialenosti v zoznamoch roztriedených podľa dĺžky

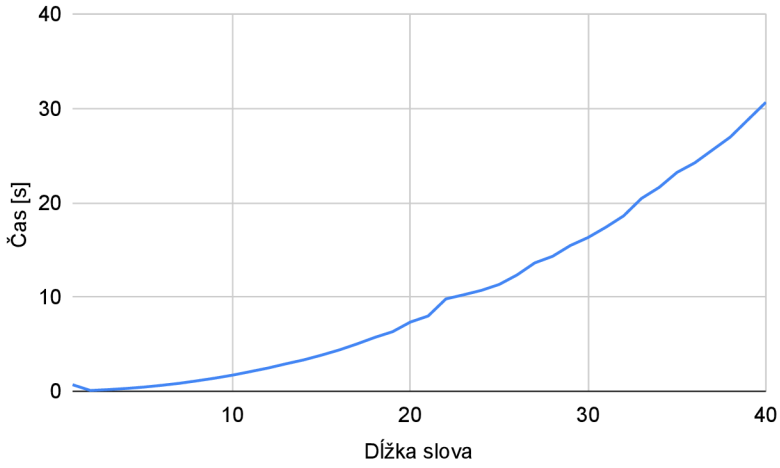
Z dôvodu, že v súbore s doménami bol počet rovnako dlhých domén až takmer 75000 bolo vykonané meranie aj na väčšom počte rovnako dlhých slov. Ďalej v odseku pri použití slova počet bude myslený počet rovnako dlhých slov. Je pridané meranie pre počty 25000, 50000 a 100000. Výsledky predchádzajúceho merania sú na samostatnom obrázku, lebo sú tak čitateľnejšie. Meranie s pridanými hodnotami je zaznamenané na obrázku 7.3. Pre priemernú dĺžku domény pri počte 25000 trvalo vyhľadanie kandidátov 20 sekúnd. Počet 50000 zabral 41 sekúnd a v prípade počtu 100000 to je 84 sekúnd. Z toho je možné usúdiť, že časová zložitosť je lineárna.



Obr. 7.3: Závislosť doby vyhľadania kandidátnych slov na dĺžke slova a rôznom počte rovnako dlhých slov

Pre porovnanie sa overil prvý spôsob získavania kandidátov. Generovali sa slová od dĺžky 1 po 40 a potom sa overila ich prítomnosť slovníku. V grafe na obrázku 7.4 je zobrazený výsledok merania. Vygenerovanie kandidátov z chybného slova o dĺžke 15 znakov trvá

3,8 sekundy. Je možné si všimnúť, že priebeh grafu je rovnaký ako na obrázku 7.2 pre 5000 rovnako dlhých slov.



Obr. 7.4: Závislosť času na dĺžke slova pri generovaní kandidátov

Dôležité zistenie je, že generovanie kandidátov trvá približne rovnako dlhý čas pre rôzne dĺžky slova ako zistenie kandidátov pomocou 25000 porovnaní editačnej vzdialenosti. Z toho je možné urobiť finálny záver, že prvotný pohľad na nové riešene kandidátov nebol dostatočne preverený. Nový spôsob nájdenia kandidátov má zmysel ak je celkový počet porovnaní do 25000. Preto bude rozumné prepínať medzi spôsobom nájdenia kandidátov na základe generovania a porovnávaním editačných vzdialeností.

Toto riešenie bolo doimplementované a teraz sú kandidáti získavaný jedným alebo druhým spôsobom podľa počtu slov, na ktorých je potrebné porovnanie editačných vzdialeností. Ak sa presiahne počet 25000 je použité generovanie slov inak sa hľadajú kandidáti v zoznamoch slov s rovnakou dĺžkou. Pred vylepšením riešenia trvalo nájdenie kandidátov pre 99 slov na slovníku s veľkosťou 797641 domén 4069 sekúnd. Po zistení problému a podpore oboch spôsobov zistenia kandidátov to trvá 423 sekúnd. To je v priemere potrebných 4,3 sekundy na zistenie kandidátov pre jedno chybné slovo. Pomocou kombinovania týchto dvoch spôsobov pre nájdenie kandidátov sa tak pôvodné riešenie pre tento prípad zlepšilo 9,6 krát.

7.4 Beh programu

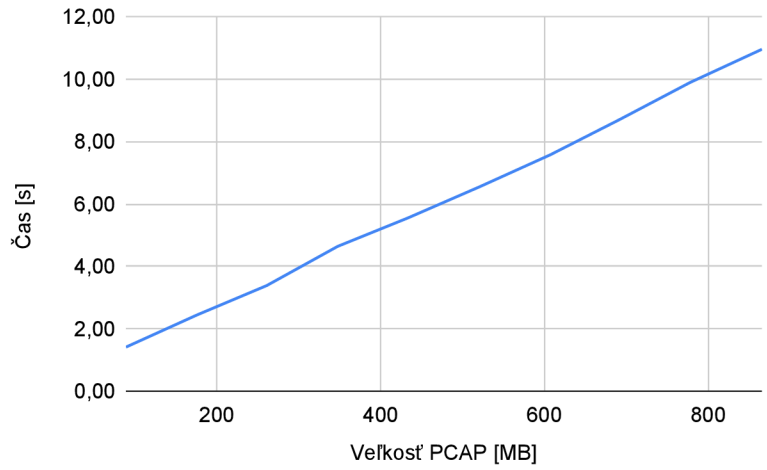
V tejto sekcii bude zmeraný čas behu programu pre rôzne vstupné dáta. Prvé meranie bude vykonané na rozdielne veľkých pcap súboroch. Týmto meraním sa sleduje vývoj dĺžky behu programu v závislosti na veľkosti súboru. Bol zmeraný čas behu programu a jeho častí v móde učenia pre dáta z pcap súboru. Výsledky sú zaznamenané v tabuľke 7.7.

Vstupne dáta obsahovali DNS dotazy, z ktorých sa vytiahli dotazované domény. Program bol spúšťaný s filtrom pre TShark, ktorý sa zadáva v konfiguračnom súbore pre beh programu. Vstupné súbory boli rôznych veľkostí od 88MB do 865MB a obsahovali od približne stotisíc paketov po milión paketov. Počet unikátnych domén je v najmenšom súbore 796 a v najväčšom 10443. Čas potrebný pre spracovanie najväčšieho súboru je skoro 11 sekúnd. Najviac času zaberá TShark, ktorý dáta filtruje a konvertuje do JSON formátu.

Veľkosť pcap v MB	Počet paketov	Počet slov	Počet unikátnych slov	TShark [s]	Konvertor [s]	Export [s]	Beh programu [s]
88,7	106843	861	796	1,41	0,003	0,003	1,41
175,1	213687	1959	1805	2,43	0,007	0,006	2,44
260,6	320530	3177	2916	3,37	0,009	0,009	3,39
347,3	427375	4338	3962	4,62	0,013	0,011	4,64
433,2	531218	5549	5075	5,52	0,017	0,014	5,55
520,7	641062	6602	6020	6,51	0,019	0,018	6,55
607,2	747907	7748	7064	7,53	0,020	0,020	7,58
692,2	854751	9004	8207	8,66	0,010	0,023	8,71
777,9	961594	10183	9305	9,83	0,029	0,033	9,89
865,5	1072089	11438	10443	10,88	0,035	0,037	10,95

Tabuľka 7.7: Program v móde učenia s rôznymi pcap súbormi na vstupe

V grafe na obrázku 7.5 je zobrazená výsledná závislosť času trvania programu od veľkosti vstupného pcap súboru. Časová zložitosť je v tomto prípade lineárna.



Obr. 7.5: Závislosť času trvania programu od veľkosti pcap súboru

K dátam využitým v móde učenia boli pridané tri pakety obsahujúce chybné domény a s takto vytvorenými súbormi bolo urobené meranie v móde diagnostiky. Bol použitý sekundárny slovník obsahujúci 1000 domén. Primárny slovník bol vždy vytvorený z príslušného bezchybného súboru, takže obsahoval toľko unikátnych slov ako je uvedené v tabuľke 7.7. Namerané výsledky sú zaznamenané v tabuľke 7.8. Vo všetkých prípadoch boli preklepy opravené.

V oboch režimoch, učenia a diagnostiky najviac času zaberie konverzia pcap do JSON. Inak s narastajúcimi vstupnými súbormi a slovníkmi rastie aj čas behu programu. Nájdenie chybných slov pri najväčšom slovníku trvalo 0,003 sekundy a nájdenie kandidátov 0,48 sekundy. Ohodnotenie kandidátov sa pohybovalo v tisícinách sekundy. Pri takto veľkých slovníkoch nebol problém s rýchlosťou nájdenia kandidátov.

TShark [s]	Konvertor [s]	Nájdenie chybných slov [s]	Nájdenie kandidátov [s]	Ohodnotenie kandidátov [s]	Beh programu [s]
1,39	0,12	0,0002	0,09	0,001	1,59
2,42	0,15	0,0004	0,14	0,001	2,66
3,58	0,24	0,0006	0,19	0,001	3,89
4,41	0,28	0,001	0,25	0,001	4,77
5,42	0,33	0,0013	0,29	0,001	5,83
6,59	0,38	0,0016	0,34	0,002	7,05
7,73	0,42	0,0019	0,38	0,002	7,23
8,99	0,47	0,0025	0,41	0,002	9,54
9,65	0,53	0,0028	0,45	0,002	10,26
11,39	0,57	0,003	0,48	0,002	12,04

Tabuľka 7.8: Program v móde diagnostiky

Pre porovnanie času spracovania s ostatnými typmi vstupných dát bol vytvorený log súbor obsahujúci páry emailovej adresy a IP adresy. Súbor obsahoval 10000 záznamov, ktoré vznikli kombináciou 12 emailových adries a 100 IP adries. Program v móde učenia zbehol za 0.03 sekundy, kde konvertor zabral 0,022 sekundy. Bolo potrebné aplikovať regulárne výrazy na vstupné záznamy a tým dostať požadované dáta. Export dát do primárneho slovníka trval 0,004 sekundy. Oproti spracovaniu pcap to je rýchlejšie, lebo sa pracuje so súborom v textovom formáte a nie je potrebné spracovať binárne dáta s toľkými informáciami. Súbor so záznamami má 317,7 kB.

Pre diagnostiku využitím log súboru boli vytvorené 3 IP adresy s preklepom a 2 emailové adresy s preklepom. Jedna chybná emailová adresa mala preklep v užívateľskom mene a druhá v doméne. Chybné dáta boli pridané k už vytvorenému log súboru, ktorý bol využitý pre naučenie. Použil sa vytvorený primárny slovník obsahujúci 100 IP adries, 12 emailových adries a 7 domén. Ako sekundárne slovníky boli použité súbory s 1000 doménami a 100 IP adresami. Program zbehol za 0,19 sekundy. Konvertor zabral viac ako polovicu programu, trval 0,11 sekundy.

Ako posledný prípad boli vytvorená dáta v CSV formáte do akého je možné exportovať záznamy NetFlow. Súbor obsahoval 2 stĺpce. Jeden s hodnotami portov a druhý s IP adresami. Bolo použitých 20 portov v kombinácii so 100 IP adresami pre vytvorenie 10000 riadkov dát. Beh programu trval 0,038 sekundy. Konvertor vyextrahoval dáta za 0,03 sekundy. Boli vytvorené 3 preklepy a pridané k vytvorenému súboru. Dva na porte a jeden na IP adrese. Program v móde diagnostiky skončil za 0.2 sekundy.

Dáta v CSV formáte a zázname neobsahovali žiadne údaje navyše. Ich čas je preto výrazne rýchlejší oproti spracovaniu pcap. Pri pcap je potrebné najprv zo všetkých tých zachytených informácii vybrať polia s údajmi, ktoré boli zadané v konfiguračnom súbore. Program bol spustený na rôznych vstupných dátach z rôznych kategórii. Bola použitá kombinovaná kategória pre email.

Kapitola 8

Záver

Cieľom práce bolo vytvoriť systém pre diagnostiku chýb v počítačových sieťach na základe hľadania a opravy preklepov. Systém má slúžiť na uľahčenie práce sieťových administrátorov, ktorí pri diagnostike siete používajú rôzne nástroje. Systém je zameraný na dáta, ktoré zadávajú priamo koncový používatelia a následne sa prenášajú sieťou alebo sa niekde ukladajú. Príkladom je konfigurácia emailového klienta, ktorú ručne vyplní používateľ.

Na začiatok sa identifikovalo v akých dátach má vôbec zmysel preklepy vyhľadávať. Z hľadiska dostupnosti a množstva dát obsahujúcich možné preklepy od používateľov sú najzaujímavejšie záznamy NetFlow, zachytené pakety vo formáte pcap a súbory zo záznamami. Každý typ dát má svoje výhody a nevýhody a tak nie je dobré sa zamerať iba na jeden typ dát. Napríklad súbory pcap môžu obsahovať najviac dát, avšak ich spracovanie je výkonovo náročnejšie a pri šifrovanej komunikácii sa obmedzí množstvo dostupných dát.

Na rozdiel od doterajšieho použitia vyhľadávania preklepov v písanom texte, táto práca aplikuje používané spôsoby na sieťové dáta. Vytvorený systém sa zameriava iba na detekciu neexistujúceho slova, takže ak preklepom v slove vzniklo iné správne slovo, vytvorený systém to nedeteguje. Pre overenie správnosti slova sú využité slovníky, ktoré kvôli zlepšeniu rýchlosti využívajú rôzne dátové štruktúry. To čo je v slovníku je správne a všetko čo sa v ňom nenájde je považované za chybu - preklep.

Pri písaní na klávesnici je možné urobiť rôzne preklepy. Základné operácie identifikoval Damerau a to vloženie, transpozíciu, náhradu a zmazanie znaku. Na základe týchto operácii je možné merať editačnú vzdialenosť medzi dvomi reťazcami a tak nájsť podobné slová, ktoré môžu byť opravou. V situácii, keď je viac kandidátov na opravu s rovnakou editačnou vzdialenosť je potrebné určiť najlepší výber. Preto bola vytvorená heuristika, ktorá zohľadňuje pravdepodobnosti výskytu jednotlivých operácii a vyhľadáva najčastejšie typy preklepov v kandidátoch, ktorí sú za to ohodnotení. Na základe najvyššieho ohodnotenia je vybraný kandidát ako náhrada. V prípade rovnako ohodnotených kandidátov môže byť aj viac ponúknutých opráv.

Vzniknutý systém funguje pod dvomi režimami. Prvý je režim učenia, ktorého cieľom je zo vstupných dát vytvoriť slovník správnych slov. Vstupom sú v tomto prípade dáta, o ktorých užívateľ systému rozhodne, že sú správne a môžu byť použité ako referenčné. V druhom režime diagnostiky využívajú predpripravené a naučené slovníky pre detekciu a opravu chyby. Program je možné spustiť aj bez fázy učenia, v tom prípade sa pracuje iba s ručne predvytvorenými slovníkmi.

V návrhu boli kandidáti generovaný pomocou reverzných operácii do editačnej vzdialenosti 2. Pri implementácii bolo zistené, že to zaberá značný čas a kandidáti boli hľadaní v zoradených zoznamoch podľa dĺžky a Damarau-Levenshtein editačnej vzdialenosti. Pri

testovaní systému sa odhalilo, že zlomový bod je 25000 porovnaní. Nad tento počet porovnaní sa oplatí použiť generovanie kandidátov, ktoré je v tomto prípade rýchlejšie. Ak sa ale pracuje so slovníkmi, kde počet rovnako dlhých slov neprevýši 1000, tak je hľadanie podľa editačnej vzdialenosti a dĺžky slova výrazne rýchlejšie.

Pre overenie funkčnosti celého programu bolo vykonané testovanie na vygenerovaných chybných slovách zo zdrojov, ktoré obsahovali reálne preklepy. Priemerne vo výsledku obstála vytvorená heuristika na 91%. V poslednej časti testovania bol program spúšťaný s rôznymi dátovými vstupmi, kde sa hľadali preklepy v rôznych kategóriach. Tieto informácie značia o tom, že systém je schopný diagnostikovať preklepy v sieťových dátach a ponúknuť opravu. Rýchlosť výsledného systému závisí od veľkosti vstupných dát, typu vstupných dát a na veľkosti slovníkov.

Možné rozšírenie práce je spracovávanie toku dát. To je možné vyriešiť postupným spúšťaním programu na častiach dát, dávkovaním. Postupne by sa aktualizoval primárny slovník. Problémom je určenie kedy nastala chyba a kedy sa odkladajú prijaté informácie. Pri väčšom počte hodnôt v primárnom slovníku, by nemalo zmysel v ňom držať slová, ktoré majú výskyt menší ako stanovená hranica.

Literatúra

[1] Ahmad, I., Parvez, M. A. a Iqbal, A. TypoWriter: A Tool to Prevent Typosquatting. In: IEEE. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. 2019, s. 423–432.

[2] Aho, A. a Corasick, M. Fast pattern matching: an aid to bibliographic search. *Communications of ACM*. 1975, roč. 18, č. 6, s. 333–340.

[3] Bao, Z., Kimelfeld, B. a Li, Y. A graph approach to spelling correction in domain-centric search. In: Association for Computational Linguistics. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. 2011, s. 905–914.

[4] Chatterjee, R., Athayle, A., Akhawe, D., Juels, A. a Ristenpart, T. PASSWORD tYPOS and how to correct them securely. In: IEEE. *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, s. 799–818.

[5] Chen, X., Huang, X., Mu, Y. a Wang, D. A Typo-Tolerant Password Authentication Scheme with Targeted Error Correction. In: IEEE. *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, s. 546–553.

[6] Christanti, V. M., Naga, D. S. et al. Fast and Accurate Spelling Correction Using Trie and Damerau-levenshtein Distance Bigram. *TELKOMNIKA*. Ahmad Dahlan University. 2018, roč. 16, č. 2, s. 827–833.

[7] Christen, P. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[8] Cisco Systems, I. *Cisco IOS NetFlow Version 9 Flow-Record Format* [online]. cisco.com, máj 2011 [cit. 2020-05-27]. Dostupné z: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.pdf.

[9] Cisco Systems, I. *IP SLAs Configuration Guide, Cisco IOS Release 15MT* [online]. cisco.com, apríl 2018 [cit. 2020-05-27]. Dostupné z: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipsla/configuration/15-mt/sla-15-mt-book/sla_overview-0.html.

[10] Claise, B., Trammell, B. a Aitken, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [Internet Requests for Comments]. STD 77. RFC Editor, September 2013. Dostupné z: http://www.rfc-editor.org/rfc/rfc7011.txt.

[11] Damerau, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM*. ACM. 1964, roč. 7, č. 3, s. 171–176.

[12] Damerau, F. J. a Mays, E. An examination of undetected typing errors. *Information Processing & Management*. Elsevier. 1989, roč. 25, č. 6, s. 659–664.

[13] De Amorim, R. C. a Zampieri, M. Effective spell checking methods using clustering algorithms. In: *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*. 2013, s. 172–178.

[14] Dhakal, V., Feit, A. M., Kristensson, P. O. a Oulasvirta, A. Observations on typing from 136 million keystrokes. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, s. 1–12.

[15] Gerhards, R. *The Syslog Protocol* [Internet Requests for Comments]. RFC 5424. RFC Editor, March 2009. http://www.rfc-editor.org/rfc/rfc5424.txt. Dostupné z: http://www.rfc-editor.org/rfc/rfc5424.txt.

[16] Gusfield, D. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge university press, 1997.

[17] Hamming, R. W. Error detecting and error correcting codes. *The Bell system technical journal*. Nokia Bell Labs. 1950, roč. 29, č. 2, s. 147–160.

[18] Han, Y., Zhao, X. a Li, J. Computer Network Failure and Solution. *Journal of Computer Hardware Engineering (TRANSFERRED)*. 2018, roč. 1, č. 1.

[19] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R. et al. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*. IEEE. 2014, roč. 16, č. 4, s. 2037–2064.

[20] Holkovič, M. a Rysavy, O. Network Diagnostics Using Passive Network Monitoring and Packet Analysis. In:. Jún 2019.

[21] Hussain, S. a Naseem, T. Spell checking. *Crulp, Nuces, Pakistan, www. crulp. org*. 2013.

[22] Kukich, K. Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*. ACM. 1992, roč. 24, č. 4, s. 377–439.

[23] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. 1966, s. 707–710.

[24] Luotonen, A., Nielsen, H. F. a Berners Lee, T. *Logging Control In W3C httpd*. W3C, Jul 1995. Dostupné z: https://www.w3.org/Daemon/User/Config/Logging.html#common_logfile_format.

[25] Martins, B. a Silva, M. J. Spelling correction for search engine queries. In: Springer. *International Conference on Natural Language Processing (in Spain)*. 2004, s. 372–383.

[26] Mauro, D. *Essential SNMP*. Sebastopol, Calif: O'Reilly, 2005. ISBN 0-596-00840-6.

[27] Mitton, R. Birkbeck spelling error corpus. *Retrieved 2009 from http://ota. ahds. ac. uk*. 1985.

[28] Mitton, R. *English Spelling and the Computer (Studies in Language Linguistics)*. Addison-Wesley Longman Ltd, dec 1995. Dostupné z: https://www.xarg.org/ref/a/0582234794/. ISBN 0582234794.

[29] Moore, T. a Edelman, B. Measuring the perpetrators and funders of typosquatting. In: Springer. *International Conference on Financial Cryptography and Data Security*. 2010, s. 175–191.

[30] Mullin, J. K. a Margoliash, D. J. A tale of three spelling checkers. *Software: Practice and Experience*. Wiley Online Library. 1990, roč. 20, č. 6, s. 625–630.

[31] Norvig, P. *How to Write a Spelling Corrector* [online]. Peter Norvig, február 2007 [cit. 2020-05-15]. Dostupné z: https://norvig.com/spell-correct.html.

[32] Odell, K. a Russell, R. Soundex phonetic comparison system. *US Patent*. 1918, roč. 1261167.

[33] Peterson, J. L. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*. ACM. 1980, roč. 23, č. 12, s. 676–687.

[34] Peterson, J. L. A note on undetected typing errors. *Communications of the ACM*. ACM. 1986, roč. 29, č. 7, s. 633–637.

[35] Pilar Angeles, M. del a Espino Gamez, A. Comparison of methods Hamming Distance, Jaro, and Monge-Elkan. In: *DBKDA 2015: the seventh international conference on advances in databases, knowl-edge and data applications*. 2015.

[36] Pollock, J. J. a Zamora, A. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*. ACM New York, NY, USA. 1984, roč. 27, č. 4, s. 358–368.

[37] QasemiZadeh, B., Ilkhani, A. a Ganjeii, A. Adaptive language independent spell checking using intelligent traverse on a tree. In: IEEE. *2006 IEEE Conference on Cybernetics and Intelligent Systems*. 2006, s. 1–6.

[38] Qiu, T., Ge, Z., Pei, D., Wang, J. a Xu, J. What happened in my network: mining network events from router syslogs. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, s. 472–484.

[39] Rimbar, H. THE INFLUENCE OF SPELL-CHECKERS ON STUDENTS'ABILITY TO GENERATE REPAIRS OF SPELLING ERRORS. *Journal of Nusantara Studies (JONUS)*. 2017, roč. 2, č. 1, s. 1–12.

[40] Sanders, C. *Practical packet analysis : using Wireshark to solve real-world network problems*. San Francisco: No Starch Press, 2007. ISBN 1-59327-149-2.

[41] Sun, Y.-C., Tang, D.-D., Zeng, Q. a Greenes, R. Identification of special patterns of numerical typographic errors increases the likelihood of finding a misplaced patient file. *Journal of the American Medical Informatics Association*. BMJ Group BMA House, Tavistock Square, London, WC1H 9JR. 2002, roč. 9, Supplement_6, s. S78–S79.

[42] TshwaneDJe, H. Sesotho sa leboa corpora, 2006. *Private Communication. September*. 2006.

[43] Ukkonen, E. Algorithms for approximate string matching. *Information and control*. Elsevier. 1985, roč. 64, 1-3, s. 100–118.

[44] Van, T., TRAN, H. A., SOUIHI, S. a MELLOUK, A. Network troubleshooting: Survey, Taxonomy and Challenges. In: IEEE. *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. 2018, s. 165–170.

[45] Van Berkelt, B. a De Smedt, K. Triphone Analysis: A Combined Method For The Correction Of Orthographical And Typographical Errors. In: *Second Conference on Applied Natural Language Processing*. 1988, s. 77–83.

[46] Wagner, R. A. a Fischer, M. J. The string-to-string correction problem. *Journal of the ACM (JACM)*. ACM. 1974, roč. 21, č. 1, s. 168–173.

[47] Wang, Y.-M., Beck, D., Wang, J., Verbowski, C. a Daniels, B. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. *SRUTI*. 2006, roč. 6, 31-36, s. 2–2.

[48] Yurcik, W. a Li, Y. Internet security visualization case study: Instrumenting a network for NetFlow security visualization tools. In: *21st Annual Computer Security Applications Conference (ACSAC)*. 2005.