VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

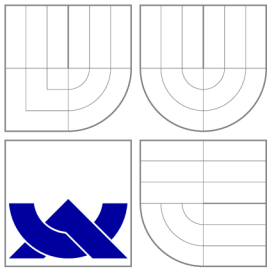
# ANOTACE SÍŤOVÉHO PROVOZU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

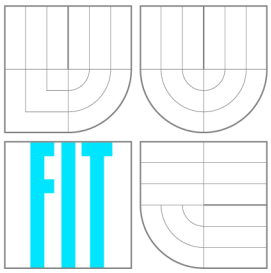
AUTOR PRÁCE
AUTHOR

JAN HOLAKOVSKÝ

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# ANOTACE SÍŤOVÉHO PROVOZU

ANNOTATION OF NETWORK TRAFFIC

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN HOLAKOVSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ŽÁDNÍK
SUPERVISOR

BRNO 2011

# Abstrakt

Tato práce se zabývá metodami rozpoznávání aplikačního protokolu v síťovém provozu. Jsou zde rozebírány a diskutovány jednotlivé metody z hledika efektivity. Dále tato práce popisuje vývoj nástroje pro podporu ruční anotace síťového provozu, který používá kombinaci vybraných metod. Na závěr tato práce obsahuje popis a výsledky experimentů provedených s vytvořeným nástrojem a diskutuje jeho efektivitu a možná rozšíření.



# Abstract

This thesis focuses on methods for network traffic classification. Furthermore the thesis analyzes the advantages and limitations of theese approaches. This work coveres development of a new tool for manual network traffic inspection and classification which uses a combination of selected classification approaches. At the end of this thesis results of conducted experiments are presented and some possible future improvements are proposed.



# Klíčová slova

rozpoznávání aplikačního protokolu, strojové učení, analýza síťového provozu, transportní statistiky, payload



# Keywords

application-layer protocol recognition, machine learning, network traffic classification, transport statistics, payload



# Citace

Jan Holakovský: Anotace síťového provozu, bakalářská práce, Brno, FIT VUT v Brně, 2011

# Anotace síťového provozu

# Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Martina Žádníka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Holakovský
17. května 2011

# Poděkování

Tímto bych chtěl poděkovat Ing. Martinu Žádníkovi za odbornou pomoc při tvorbě této práce. Další díky patří Lucu Salgarellimu za poskytnutí vzorků dat a Hyunchul Kimovi za poskytnutí softwaru pro zpracování těchto vzorků.

# Obsah

# Kapitola 1

# Úvod

Objem dnešní síťové komunikace dosáhl kolosálních rozměrů, a vzhledem k vývoji nových a neustálému zrychlování stávajících technologií je předpoklad, že bude i nadále růst. Vzhledem k tomuto faktu roste potřeba tento provoz analyzovat a třídit. Tato potřeba vyvstává hned z několika důvodů. Pomineme-li ty nezákonné nebo eticky sporné, je třeba sledovat síťový provoz například pro potřeby zajištění kvality služeb nebo včasného odhalování bezpečnostních rizik. Pro optimalizaci služeb poskytovaných na internetu a pro optimalizaci zpracování provozu, který tyto služby generují, je potřeba identifikovat typ provozu podle aplikace a následně tento provoz zpracovavat dle požadavků aplikace.

Pro tyto a další činnosti je třeba velmi přesných dat. Ovšem získat tato data je velmi pracné, neboť neexistuje mnoho spolehlivých způsobů ověřování výsledků. Výzkum je v tomto směru značně brzděn právě nedostatkem referenčních dat, která by bylo možno použít pro testování a validaci nových a lepších metod analýzy. Vzhledem k množství citlivých informací putujících po různých síťích se každý zdráhá svá data poskytnout i pro výzkumné účely kvůli obavám ze zneužití těchto dat a informací v nich obsažených.

Tato práce se zabývá metodami anotace dat odchycených na síťi a jejím cílem je aplikovat několik těchto metod při tvorbě nástroje, který bude sloužit ke zrychlení a zjenodušení ruční anotace síťového provozu.

Ve druhé kapitole jsou popsány základní metody klasifikace aplikačního protokolu a jsou diskutovány jejich výhody a nevýhody. Dále jsou v této kapitole představeny některé existující nástroje, které tyto metody využívají.

Třetí kapitola se zabývá návrhem vlastního nástroje. Obsahuje specifikaci požadavků na tento nástroj a popisuje navrhovanou základní strukturu této aplikace. Dále jsou v této kapitole navrhovány způsoby reprezentace výsledků.

Čtvrtá kapitola obsahuje popis implementace navrhovaného nástroje a zdůvodnění některých rozhodnutí, ke kterým došlo během vývoje. Jsou zde podrobně popsány a vysvětleny jednotlivé části aplikace a některé části použitých algoritmů. Na závěr této kapitoly jsou popsány optimalizace, které byly během vývoje provedeny.

Předposlední, pátá, kapitola obsahuje popis experimentů provedených s vytvořeným nástrojem. Dále jsou zde prezentovány výsledky těchto experimentů a závěry z nich vyvozené.

V závěru jsou hodnoceny dosažené výsledky, diskutovány klady a zápory vyvinuté aplikace a navrhována možná rozšíření a budoucí směry vývoje této aplikace.

# Kapitola 2

# Teoretický rozbor

Převážná většina dnešní síťové komunikace využívá nějaký síťový model, ať již klasický ISO/OSI model, nebo některý z jeho derivátů, například TCP/IP. Tyto modely se skládají z několika vrstev, kdy každá zajišťuje splnění určitých požadavků pro chod komunikace a ošetřují například i zotavení z chybových stavů.

## 2.1 TCP/IP model

Tato práce předpokládá TCP/IP model, který se skládá ze čtyř vrstev. Nejnižší – *vrstva síťového rozhraní* – není z pohledu této práce zajímavá, a proto nebude dále rozebírána.

Další vrstva – *síťová* – slouží k adresování jednotlivých bodů v síti a zakrývá implementační rozdíly mezi protokoly nižší vrstvy. Protokoly síťové vrstvy jsou zodpovědné za směrování paketů mezi jednotlivými podsítěmi a zajišťují kvalitu a plynulost služeb na nich běžících. Nejdůležitějšími představiteli těchto protokolů jsou IP verze 4 a 6 a jejich podmnožiny ICMP a IGMP.

Protokol IP verze 4 je již dnes zastaralý a jeho adresový prostor je skoro vyčerpán, ale stále představuje asi 80 % z veškeré světové síťové komunikace. Jeho nástupcem je protokol IP verze 6, který oproti svému předchůdci přináší řadu vylepšení. Z hlediska této práce je hlavní výhodou jednodušší hlavička, která má pevnou délku a dá se rychleji parsovat. Specifikace těchto protokolů lze nalézt zde [15] a zde [3].

Následující vrstva – *transportní* – zajišťuje spolehlivý přenos dat s požadovanou kvalitou a umožňuje navázání více spojení mezi entitami se stejnou adresou nižší vrstvy pomocí tzv. portů. Hlavními protokoly této vrstvy jsou TCP a UDP.

Protokol TCP je stavový a zaručuje bezztrátový přenos dat a kontroluje příjem fragmentovaných dat ve správném pořadí. Navázání spojení probíhá pomocí tzv. three-way handshake a zaručuje synchronizaci potvrzovacích a kontrolních mechanizmů. Ukončení spojení zajišťuje tzv. four-way handshake, který se stará o to, aby bylo spojení opravdu ukončeno až poté, kdy obě strany již nechtějí posílat žádná data, a obě koncové entity data úspěšně přijaly. Tyto vlastnosti jsou ale vykoupeny poměrně velkou režijí, a proto není protokol vhodný pro všechny aplikace. I přesto je ale dominantním protokolem transportní vrstvy. Jeho specifikaci je možné nalézt zde [16].

Oproti tomu je UDP bezstavový a nezaručuje správné pořadí ani kompletnost přijímaných dat, ale nabízí vyšší datovou prostupnost. Jeho specifikace je zde [14].

Poslední vrstvou je vrstva *aplikační*, která funguje jako rozhraní pro aplikace, které na ní pracují. Na této vrstvě pracuje nepřeberné množství protokolů, které se neustále mění,

a vznikají stále další. Jejich rozpoznávání z odchycené síťové komunikace je jádrem této práce.

## 2.2 Metody rozpoznávání aplikačního protokolu

Analýza aplikačního protokolu je poměrně složitý a výpočetně náročný problém. Proto se zpravidla neanalyzují odchycené pakety jednotlivě, ale nejprve se shlukují do větších celků – datových toků. Datové toky jsou jednoznačně identifikovány pomocí tzv. *five-tuple*, neboli pětice skládající se ze zdrojové a cílové adresy protokolu síťové vrstvy, čísla zdrojového a cílového portu protokolu transportní vrstvy a názvu transportního protokolu. Ačkoli to není moc časté, je teoreticky možné, aby byl datový tok uzavřen a vzápětí otevřen nový mezi stejnými hosty na stejných portech za použití stejného transportního protokolu, ale s jiným obsahem a aplikačním protokolem. Aby bylo možné tyto toky rozlišit, přidává se k *five-tuple* někdy ještě čas zahájení toku. S tím také souvisí nutnost zavedení mechanizmů, které detekují ukončení datového toku.

### 2.2.1 Ukončování datových toků

Datové toky lze ukončovat několika způsoby. Nejjednodušším je prohlásit po uplynutí stanoveného časového intervalu všechny toky za ukončené. Tento přístup je velmi jednoduchý a časově a paměťově nenáročný. Má však obrovskou nevýhodu v tom, že pokud je tok takto přeťat, bude jeho první část pravděpodobně analyzována úspěšně (pokud bylo odchyceno dostatečné množství dat pro analýzu), zatímco části následující s velkou pravděpodobností úspěšně identifikovány nebudou, protože jim budou chybět data z počátku toku.

Další možnost je zavedení časovače, který sleduje pro každý tok zvlášť dobu jeho neaktivity, a pokud tato doba přesáhne předem stanovený limit, tok ukončí. Výhoda této metody je, že dokáže správně ukončovat toky, které jsou zavírány chybným nebo nestandartním způsobem. Nevýhoda je, že není možné poznat, jestli aplikace spojení opravdu ukončila nebo ho nechala otevřené, ale už delší dobu neposílá ani nepřijímá žádná data. V takovém případě bude tok opět nesprávně rozdělen. Pokud ovšem toto dělá aplikace smysluplně, lze předpokládat, že tyto menší části toku budou obsahovat dostatek informací pro jejich úspěšnou identifikaci. Tato metoda předpokládá, že bude časový limit nastaven smysluplně a adekvátně topologii linky, ze které data pocházejí.

Poslední variantou je sledování stavu spojení (pouze u stavových transportních protokolů). Díky tomu je možné sledovat skutečný stav spojení a správně na něj reagovat. Problém ovšem nastává, pokud aplikace uzavírá spojení nestandardně nebo přímo chybně. Zpravidla pak dojde k předčasnému uzavření toku a vzniku několika „syrotků“ – paketů, které nenesou žádnou informaci použitelnou pro jejich identifikaci.

Po úspěšném rozdělení paketů do datových toků, lze konečně přistoupit k vlastní analýze aplikačního protokolu.

### 2.2.2 Analýza podle čísel portů

Tento přístup je nejstarší a nejjednodušší. Spočívá v porovnání čísel portů transportního protokolu se seznamy tzv. *Well Known Ports* a *Registered Ports*. To jsou seznamy obsahující čísla portů a jim přiřazené aplikace a protokoly. Toto rozdělení by mělo být dodržováno, ale v praxi se tak ne vždy děje. Jak je prokázáno v těchto pracech [8] [10], úspěšnost této metody se pohybuje mezi 70–90 %. Dále autoři prokazují, že tato metoda selhává hlavně

při rozpoznávání P2P komunikace a pasivního FTP. To přisuzují tomu, že tyto aplikace a protokoly využívají dynamických čísel portů nebo se maskují používáním čísla portu jiného protokolu. Na druhou stranu v článku [8] je prokázáno, že tato metoda je stále velmi platná a její výsledek může být významným vodítkem při identifikaci aplikačního protokolu. Dále autoři prokázali, že tato metoda má velmi dobré výsledky pro starší protokoly jako například HTTP, DNS, e-mailové protokoly, News, SNMP, NTP, Chat a SSH.

### 2.2.3 Analýza podle payloadu paketů

Tato metoda je také známá už poměrně dlouhou dobu a je hojně využívána množstvím různých aplikací. Její princip je také velice jednoduchý – spočívá ve vyhledávání známých vzorů v payloadu paketů. Předpokladem pro její úspěšné použití je správné sestavení obsahu fragmentovaného v jednotlivých paketech. To může být u protokolů běžících nad UDP problém, protože UDP nezajišťuje správné pořadí přijímaných paketů. Dalším předpokladem je existence kvalitních vzorů. Pro některé protokoly je velmi obtížné kvalitní vzory sestavit hned z několika důvodů:

- chybí nebo není dostupná specifikace protokolu – vzory jsou sestavovány podle empirických pozorování a nemusí být přesné
- protokol má několik verzí nebo variant a je těžké podchytit ve vzoru všechny – méně časté varianty bývají zanedbávány
- protokol neobsahuje žádný výrazný znak nebo se velmi podobá jinému protokolu – vzory jsou příliš obecné
- protokol používá několik datových toků najednou (např. FTP – řídicí a datový kanál) – toky obsahující data je prakticky nemožné identifikovat

Z těchto problémů vyplývají některé hlavní nevýhody tohoto přístupu. Vytvoření a odladění vzorů pro nově vzniklé protokoly je velmi pracné a tudíž je rozšiřitelnost analyzátorů tohoto typu velmi obtížná. Navíc je nutné udržovat vzory aktuální, jinak může spolehlivost analyzátoru rapidně klesnout. Kvalita vzorů má obrovský dopad na úspěšnost klasifikace – příliš obecné vzory mohou označit i jiné aplikace a protokoly a naopak příliš specifické vzory mohou správně identifikovat jen podmnožinu určitého protokolu.

Obrovskou nevýhodou této metody je velká časová náročnost, a to zejména u složitých vzorů, nebo pokud se hledaný vzor nachází velmi hluboko. Tento problém se dá částečně kompenzovat zjednodušením vzorů, ale zpravidla na úkor přesnosti. Další poměrně výrazný problém je, že tato metoda si neporadí se šifrovanými daty.

Na druhou stranu, pokud je analyzátor vybaven kvalitními vzory, může dosáhnout poměrně slušné přesnosti. Tato metoda je dnes považována za standard a např. v [8] je použita jako referenční.

### 2.2.4 Analýza podle statistik datových toků

Tato metoda je předmětem značného vědeckého zájmu a jejím variantám je věnováno mnoho prací. Spočívá v klasifikování datového toku na základě jeho transportních (a jiných) statistik bez nutnosti zkoumat payload paketů. K tomu jsou zpravidla používány různé varianty strojového učení.

#### Clustering

Jednou z možností je použití tzv. *unsupervised machine learningu*, konkrétně *clusteringu*. Tato metoda je založena na rozdělení daného datového souboru do skupin s podobnými statistikami transportní vrstvy. Tři základní algoritmy použité při clusteringu a jejich výhody a nevýhody jsou popsány zde [4]. Přesnost těchto algoritmů při klasifikaci datových toků se podle autorů pohybuje mezi zhruba 70–95 %. Přesnost klasifikace je značně závislá na kvalitě vzorových dat použitých při výstavbě modelu. Nevýhodou tohoto přístupu je většinou nemožnost identifikace konkrétního protokolu, ale pouze zařazení do určité skupiny (např. WWW, P2P, multimédia apod.). Toto chování ale může být v některých případech výhodné – nové nebo neznámé protokoly mohou být úspěšně klasifikovány, aniž by byly zahrnuty v původním modelu.

#### Supervised Machine Learning

Další možností je použití tzv. *supervised machine learningu*. Na rozdíl od clusteringu tato metoda pouze nerozděluje toky do skupin, ale podle podobnosti statistik určuje konkrétní protokoly. Používaný klasifikátor je třeba nejprve natrénovat na cvičné datové množině, na kterou jsou podle použitého algoritmu kladeny různé nároky. Opět existuje řada algoritmů, které lze použít. Obecně se dají rozdělit na dvě skupiny – deterministické a probabilistické. Deterministické algoritmy přiřadí vyhodnocovanému vzorku pouze jeden výsledek, zatímco probabilistické mu přiřadí několik výsledků s různou pravděpodobností. Efektivita různých algoritmů je měřena v článku [8]. Z testu jednoznačně nejlépe vycházejí *Support vector machines*, které prokazují úspěšnost 95–99 %. SVM navíc pro dosažení těchto výsledků stačí mnohem menší tréningová množina. U všech těchto algoritmů hraje značnou roli nastavení parametrů konkrétního klasifikátoru. Jak je vidět na příkladu z [11], zvolení správných parametrů a vyladění klasifikátoru pro konkrétní tréningovou množinu může zvednout úspěšnost i o 40 %.

Hlavní nevýhodou použití supervised machine learningu jsou vysoké nároky na kvalitu a množství tréningových dat a velmi náročné odladění klasifikátoru. Navíc každá tréningová množina vyžaduje jiné nastavení, takže neexistují žádné „univerzální“ hodnoty parametrů klasifikátorů. Trénování klasifikátoru je navíc velmi časově náročné.

Použití metod klasifikace využívajících statistiky datových toků se jeví jako nadějné řešení, ale stále je ještě třeba dořešit mnoho problémů s těmito metodami spojených. Největšími problémy jsou zjištění směrodatných a neredundantních znaků pro jednotlivé protokoly a zajištění dostatečně reprezentativního vzorku dat pro tréning klasifikátoru.

### 2.2.5 Analýza podle chování hostů

Poměrně zajímavá metoda je popsána v článku [7]. Tento analyzátor pozoruje a identifikuje vzory v chování hostů na transportní vrstvě. Jeho autoři studují chování hostů na třech úrovních:

- sociální úroveň – sleduje se počet spojení s ostatními hosty
- funkcionální úroveň – podle chování se rozlišuje, zda se host pro danou službu chová jako server nebo jako klient
- aplikační úroveň – jednotlivé služby jsou rozděleny podle transportních statistik toků

Analyzátor je potom schopen rozlišit, zda se host chová jako server nebo klient. Pokud je nalezen server, zkoumá se podle statistik, jaké služby poskytuje a na základě toho jsou potom všechny toky mezi ním a klienty na daném portu označeny.

Autoři vidí hlavní přínosy tohoto přístupu v možnosti správně detekovat nové protokoly s podobným chováním a v možnosti objevit potenciálně nebezpečnou nebo nevyžádanou komunikaci, která se snaží maskovat za jiné služby (její chování vybočuje ze vzorů). Uváděná úspěšnost tohoto analyzátoru je mezi 80–95 %.

V článku [8] byl tento systém podroben testování a byly zjištěny následující nedostatky:

- Systém musí být nasazen na topologicky vhodném místě, kde bude mít přístup k maximu údajů o chování hostů. Nejvhodnějšími místy jsou hrany vnitřních a vnějších sítí, kde může systém zkoumat chování celé vnitřní sítě. Z toho vyplývá, že tento systém příliš nefunguje na páteřních linkách, kde není k dispozici dost údajů o celkovém chování hostů a navíc díky asymetrickému směrování mohou chybět údaje o jednom směru toků.
- Systém vyžaduje určité množství toků náležících jednomu hostu, aby mohl vyhodnotit jeho chování. Toky, které jsou pod tímto prahem, nejsou vyhodnoceny.

Další nevýhodou tohoto přístupu je velmi obtížná tvorba nových vzorů a pracné vyladění analyzátoru pro konkrétní místo, kde je nasazen.

Samozřejmě existuje ještě celá řada dalších více či méně exotických přístupů pro klasifikaci aplikačních protokolů, ale jedná se už spíše o okrajové záležitosti. Protože každá zde popsaná metoda má nějaké nevýhody a nejlépe funguje jen pro určité protokoly, jeví se jako nejlepší použití vhodné kombinace jednotlivých přístupů.

## 2.3 Nástroje pro analýzu síťového provozu

### 2.3.1 Wireshark

Asi základním nástrojem pro analýzu síťového provozu je `Wireshark` [19]. Tento nástroj zkoumá jednotlivé pakety a parsuje je pomocí tzv. *dissectorů*. Jednotlivé aplikační protokoly jsou rozpoznávány především podle čísel portů, ale jednotlivé dissectory mohou dělat různé syntaktické a sémantické kontroly, takže je někdy možné objevit protokoly, které se snaží maskovat.

Hlavním kladem Wiresharku je jeho relativní jednoduchost. Má poměrně intuitivní grafické uživatelské rozhraní, takže práce s ním je v celku pohodlná.

Bohužel jednoduchot tohoto nástroje je zároveň jeho hlavním záporem – je vhodný pouze ke zkoumání malého množství dat. Protože neumí agregovat pakety do datových toků nebo jiných větších celků, je pro analýzu většího objemu dat prakticky nepoužitelný. Kvalita jednotlivých dissectorů se navíc dost různí, většinou se jedná pouze o velmi jednoduché parsery, které skutečný obsah paketu příliš nekontrolují. Pokud je třeba se pouze podívat do payloadu paketu (například při ruční anotaci datových toků), je Wireshark adekvátním nástrojem, ovšem pouze s ním se anotace síťového provozu dělat nedá.

### 2.3.2 Snort

Zajímavým nástrojem pro sledování síťového provozu je `Snort`. Patří do skupiny tzv. *IDS*, neboli *Intrusion Detection System*. Tyto nástroje zkoumají síťový provoz a hledají anomálie,

které by mohly být pokusy o narušení bezpečnosti. Pracuje na principu porovnávání vzorů v paketech, ale obsahuje i sadu preprocesorů, které mu umožňují hlubší analýzu payloadu.

Tento nástroj disponuje velmi výkonným enginem pro porovnávání vzorů, ovšem aby dosáhl potřebného výkonu pro filtrování komunikace v reálném čase, kontroluje pouze pakety, které jsou podezřelé. Pokud by byla přidána pravidla, která by kontrolovala veškerý provoz, jeho výkonnost by výrazně poklesla.

`Snort` obsahuje vlastní jazyk pro tvorbu nových pravidel a nástroje pro tvorbu a zapojení přídavných modůlu a preprocesorů.

Ačkoli se `Snort` na první pohled může jevit jako vhodný nástroj pro anotaci síťového provozu, není tomu tak hned z několika důvodů:

1. Bylo by nutné vytvořit velké množství nových vzorů v podstatě pro všechny síťové protokoly.

2. Pro správnou identifikaci některých protokolů by bylo třeba upravit nebo vytvořit preprocesory.

3. `Snort` neposkytuje možnost kontrolovat data po větších celcích – aby neprováděl analýzu po paketech, bylo by nutné vytvořit modul, který by pakety sdružoval.

4. `Snort` neposkytuje vhodnou formu výpisu výsledků a bylo by nutné pro tuto funkci také napsat modul.

Vezmeme-li v úvahu výše uvedené nedostatky, znamenalo by upravení tohoto nástroje do podoby vhodné pro anotaci síťového provozu, téměř kompletní přestavění celého systému. Vzhledem k tomu, že by bylo nutné obejít řadu jeho vnitřních principů, pravděpodobně by byla obětována jeho efektivita, protože původní optimalizace by nebyly pro nové použití vhodné.

`Snort` lze ale s úspěchem použít k jeho původnímu účelu, a to vyhledávání podezřelé komunikace a pravděpodobných útoků. Byl by proto zajímavým doplňkem jiného systému, ale vzhledem k tomu, že útoky a komunikace podobného typu tvoří pouze zlomek celkového síťového provozu, byl by jeho přínos spíše okrajový.

### 2.3.3 Bro

Tento nástroj [1] také patří mezi *IDS* a s trochou nadsázky by se dal nazvat starším bratrem `Snort`u. Oproti němu však disponuje velmi silným skriptovacím jazykem pro psaní nových modulů. Pracuje na velmi podobném principu – obsahuje sadu pravidel, podle kterých kontroluje síťovou komunikaci a na základě nich spouští skripty, které vykonávají patřičné akce.

Pro použití k anotaci síťového provozu není tento nástroj příliš vhodný z téměř stejných důvodů jako `Snort`, ale díky jeho vestavěnému skriptovacímu jazyku by pravděpodobně bylo o něco jednodušší jej k tomuto účelu upravit. `Bro` také obsahuje řadu sofistikovanějších vestavěných modulů než `Snort`, které by mohly být při anotaci síťového provozu nápomocné. Za zmínku stojí poměrně robustní analyzátor `http`, který je schopen odahlit jiné protokoly maskující se za `http` nebo protokoly přes něj tunelované. Dále `Bro` obsahuje statistický analyzátor a analyzátory schopné odhalit některé P2P protokoly.

### 2.3.4 L7-filter

`L7-filter` [9] je v podstatě rozšířením `netfilter/iptables`. I tento nástroj pracuje na principu vyhledávání vzorů v payloadu paketů. Obsahuje značné množství více či méně kvalitních vzorů, které jsou více či méně kvalitně otestovány.

Práce s tímto nástrojem není příliš intuitivní, protože primárně funguje jako rozšíření firewallu. Aby bylo možno tento nástroj použít, je třeba odchycená data přesměrovat do nějakého virtuálního síťového rozhraní například přes rozharní `localhost`. Navíc je poměrně obtížné jej nainstalovat, protože musí být zkompilován spolu s jádrem. Existuje i tzv. *user-space* verze, kterou není nutno kompilovat s jádrem, ale ta vyžaduje nastavení specifických voleb v `iptables`, takže je stejně nutné překompilovat jádro.

Ačkoliv má samotná aplikace tyto nevýhody, vzory v ní obsažené lze snadno použít v jiném nástroji, protože jsou psané pomocí regulárních výrazů. Užité implementace regulárních výrazů v *kernel* a *user-space* verzích se nepatrně liší, takže jejich chování je mírně odlišné.

### 2.3.5 Hippie

Tento nástroj [6] je určitá alternativa k `L7-filter`u. Také pracuje jako rozšíření `iptables`, ale neporovnává pouze vzory v paketech, ale implementuje systém *dissectorů* podobně jako `Wireshark`. Každý *dissector* je vlastně parser, který zkoumá daný protokol a kontroluje jeho obsah z fukcionální stránky.

Díky podobnosti s `L7-filter`em, přejímá tento nástroj stejné nevýhody. Navíc se jeho vývoj asi před dvěmi lety zastavil, takže nepodporuje nová jádra a díky mnoha chybám s nimi nejde zkompilovat. Stránky projektu byly pravděpodobně zrušeny, takže dnes není dostupná žádná dokumentace k tomuto nástroji.

### 2.3.6 OpenDPI

`OpenDPI` [13] je Open Source varianta systému `PACE` od firmy ipoque. Je to pokračovatel projektu `IPP2P`, který se zaměřoval na klasifikaci P2P komunikace. Momentálně je k dispozici buď wrapper pro `netfilter/iptables` nebo knihovna pro jazyk C.

Tento nástroj pracuje opět na bázi *pattern matchingu* a dokáže kromě běžných protokolů identifikovat několik P2P, běžné instant messengery a některé streamingové protokoly.

### 2.3.7 GTVS

Tento systém popsaný v [2] kombinuje několik přístupů ke klasifikaci datových toků. Je zde využito porovnávání vzorů v payloadu, analýza čísel portů, statistiky datových toků a analýza chování hostů. To vše je zabaleno do grafického uživatelského rozhraní a vybaveno několika heuristikami pro urychlení anotace.

`GTVS` je postaven na technologii `Click modular router` a `MySQL`, které jsou propojeny několika skripty a nad nimiž stojí GUI.

Tento systém je přímo navržen pro podporu a urychlení mauální anotace síťového provozu. Jakkoli se jeví jako ideální kandidát pro použití při anotaci, prakticky neexistující dokumentace a velmi obtížná instalace a nastavení služeb, které systém využívá, ho činí prakticky nepoužitelným.

# Kapitola 3

# Návrh řešení

Cílem této kapitoly je navrhnout nástroj pro podporu a urychlení ruční anotace síťového provozu, která je při použití stávajících nástrojů velmi náročná, nepřesná nebo přinejmenším zdlouhavá. Dále by měl být navržen vhodný formát pro uložení a reprezentaci výsledků.

## 3.1 Specifikace systému

Navrhovaný systém by měl splňovat následující požadavky:

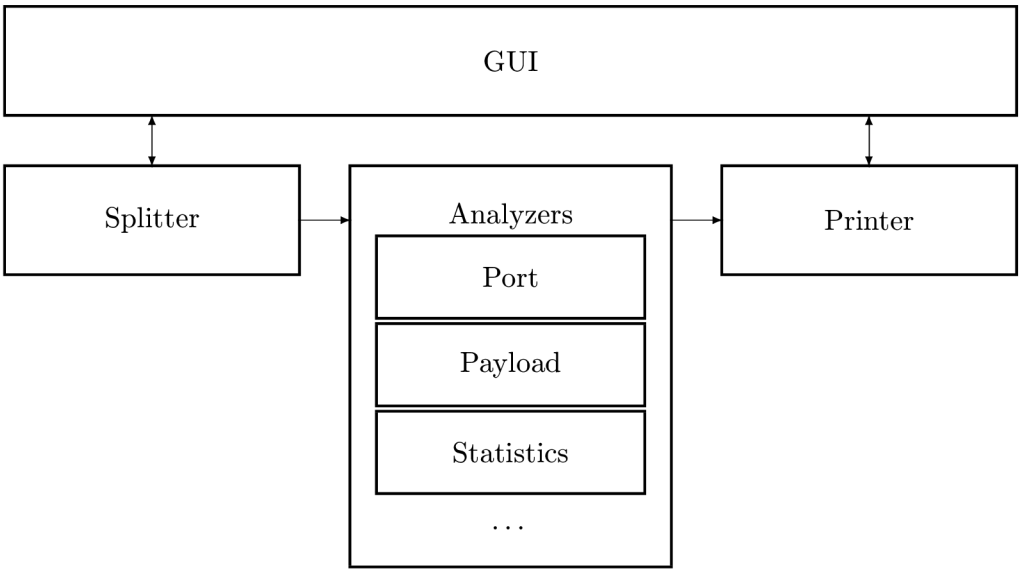
- možnost provádět anotaci nad většími celky než jednotlivými pakety – systém by měl umět sdružovat pakety do datových toků a provádět anotaci nad nimi.
- zapojení několika způsobů klasifikace datových toků – systém by měl obsahovat tři nejrozšířenější metody klasifikace síťového provozu (analýza čísel portů, analýza payloadu a analýza transportních statistik toků)
- modularita – výsledný nástroj by měl být snadno rozšiřitelný o další analyzátory
- přenositelnost – systém by měl fungovat na různých systémech (alespoň UNIX/Linux a MS Windows)
- konfigurovatelnost – klíčové parametry systému by mělo být možné pohodlně nastavovat
- jednoduché používání – systém by měl obsahovat jednoduché uživatelské rozhraní

## 3.2 Návrh systému

Navrhovaný systém se skládá z několika modulů, které mezi sebou vzájemně komunikují viz Obrázek 3.1.

### 3.2.1 Modul `Splitter`

Tento modul se stará o sdružování paketů do obousměrných datových toků na základě jejich *five-tuple* a její obrácené verze. Zajišťuje správné ukončování toků podle uživatelem definovaných parametrů. Pro ukončování toků používá dva timery (samostatný timer pro TCP a UDP) a u TCP navíc ještě sleduje stav spojení a náležitě na něj reaguje. Také se stará o vytřídění nezajímavých paketů (ARP, ICMP, IGMP apod.). Modul by měl být




Obrázek 3.1: Základní struktura systému

schopen zpracovat pakety užívající protokoly IP verze 4 a 6. Dále by si měl poradit se základními formami enkapsulace (IP in IP). Také se stará o výpočet transportních statistik toků, které jsou pak použity v modulu `Statistics`. Jeho poslední úlohou je sběr celkových statistik o analyzovaném souboru.

Pro správné ukončování TCP toků není třeba sledovat všechny stavy spojení, proto tento modul sleduje pouze resety spojení a *four-way handshake*. Sledování *four-way handshake* zajišťuje jednoduchý konečný automat, který vychází ze stavů TCP popsaných v [16]. Tento automat je ale podstatně zjednodušen, protože sleduje stav spojení jako celek a ne oba stavy koncových bodů (ty se mohou částečně lišit podle rozdělení úloh klient – server). Jednoduchý graf přechodů tohoto automatu je znázorněn na Obrázku 3.2. Tento graf je pro přehlednost zjednodušen, protože kromě TCP flagů je při přechodech ještě nutno kontrolovat potvrzovací čísla paketů. Stavy automatu jsou pojmenovány shodně se stavy TCP, ale významově se odlišují. Hrany grafu jsou popsány TCP flagy dalšího příchozího paketu.

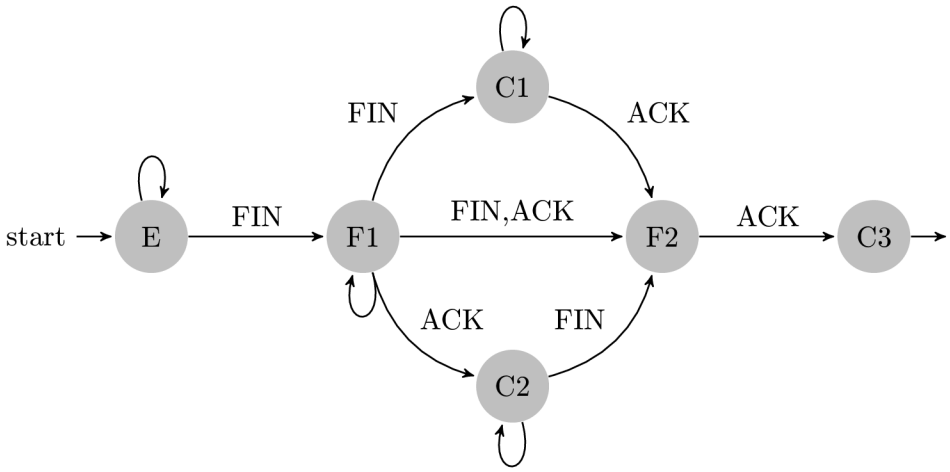
### 3.2.2 Modul `Analyzers`

Tento modul slouží jako rozhraní pro jednotlivé analyzátory a tvoří kolem nich jakousi obálku. Stará se o registraci, zapnutí a spouštění jednotlivých analyzátorů. Také zjišťuje jaká nastavení jednotlivé analyzátory potřebují, a kontroluje, zda jsou daná nastavení k dispozici. Zajišťuje, aby každý analyzátor implementoval tyto funkce:

1. inicializace – V tomto kroku by si měl každý analyzátor načíst data potřebná pro klasifikaci a připravit si potřebné struktury a proměnné.

2. vlastní analýza – V tomto kroku analyzátor klasifikuje datové toky a ukládá si mezivýsledky

3. finalizace – Toto je závěrečný krok, kdy může analyzátor provést dokončení klasifikace (např. pokud je analyzátor víceprůchodový, v předchozím kroku si pouze z jednotli-




Obrázek 3.2: Graf přechodů konečného automatu pro TCP. E=ESTABLISHED, F1=FIN-WAIT-1, F2=FIN-WAIT-2, C1=CLOSING, C2=CLOSE-WAIT, C3=CLOSED

vých toků nasbírá potřebná data a až v tomto kroku vykoná analýzu v potřebném počtu iterací). Všechny analyzátory by měly na konci tohoto kroku předat své výsledky modulu `Printer`.

Dále by měly všechny analyzátory obsahovat nastavení váhy svého výsledku, které může uživatel změnit. Pokud dosáhnou součty vah stejných výsledků určitého prahu, je datový tok automaticky označen daným výsledkem.

#### Modul analyzátoru `Port`

Tento analyzátor by měl provádět jednoduchou analýzu datových toků na základě čísel portů. Seznam portů by měl být uložen v externím souboru, aby ho mohl uživatel snadno upravovat. Základ tohoto souboru by měl být tvořen seznamem portů registrovaných u organizace IANA.

#### Modul analyzátoru `Payload`

Tato část navrhovaného systému zajišťuje analýzu dat pomocí kontroly payloadu. Užívá vzory převzaté z nástroje `L7-filter` [9], které vyhledává v payloadu jednotlivých toků. Tyto vzory by opět měly být uloženy v externím souboru, aby mohly být uživatelem snadno modifikovány. Tento modul také obsahuje mechanismus pro reassembling dat přenášených přes TCP – data každého TCP toku jsou seřazena podle sekvenčních čísel. Délka prohledávaných dat by měla být nastavitelná.

#### Modul analyzátoru `Statistics`

Tento analyzátor funguje na principu porovnávání podobnosti transportních statistik datových toků. Protože podle [8] je výhodnější použití jednosměrných datových toků, jsou analyzátoru předány statistiky vypočítané pro každý směr zvlášť a použit je výsledek s větší pravděpodobností. Porovnávací algoritmus by měl být realizován pomocí *Support Vector*

*Machine*, které ve výše zmíněném článku vykazovaly nejvyšší přesnost analýzy za použití nejmenší tréningové množiny. Jako vhodná tréningová množina byla zvolena anotovaná data z [5]. Algoritmus SVM by měl vracet skupinu, do které vzorek s nejvyšší pravděpodobností spadá. Dále by měl analyzátor obsahovat nějaký uživatelsky nastavitelný práh pravděpodobnosti, který by určoval, zda se má daný výsledek použít, nebo zda je jeho pravděpodobnost příliš nízká.

### 3.2.3 Modul `Printer`

Tento modul by se měl starat o uchovávání výsledků vrácených analyzátory a o jejich předání v potřebné podobě dalším částem systému. Také by měl zajišťovat výpis výsledkových souborů ve formátu popsaném v kapitole 3.4.

### 3.2.4 Grafické uživatelské rozhaní

GUI by mělo stát vedle zbytku systému a pouze se starat o nastavení jednotlivých částí systému, spouštění nebo ukončování analýzy a výpis výsledků po úspěšném dokončení analýzy. Systém by měl jít spustit i bez grafického uživatelského rozhraní pouze jako konzolová aplikace.

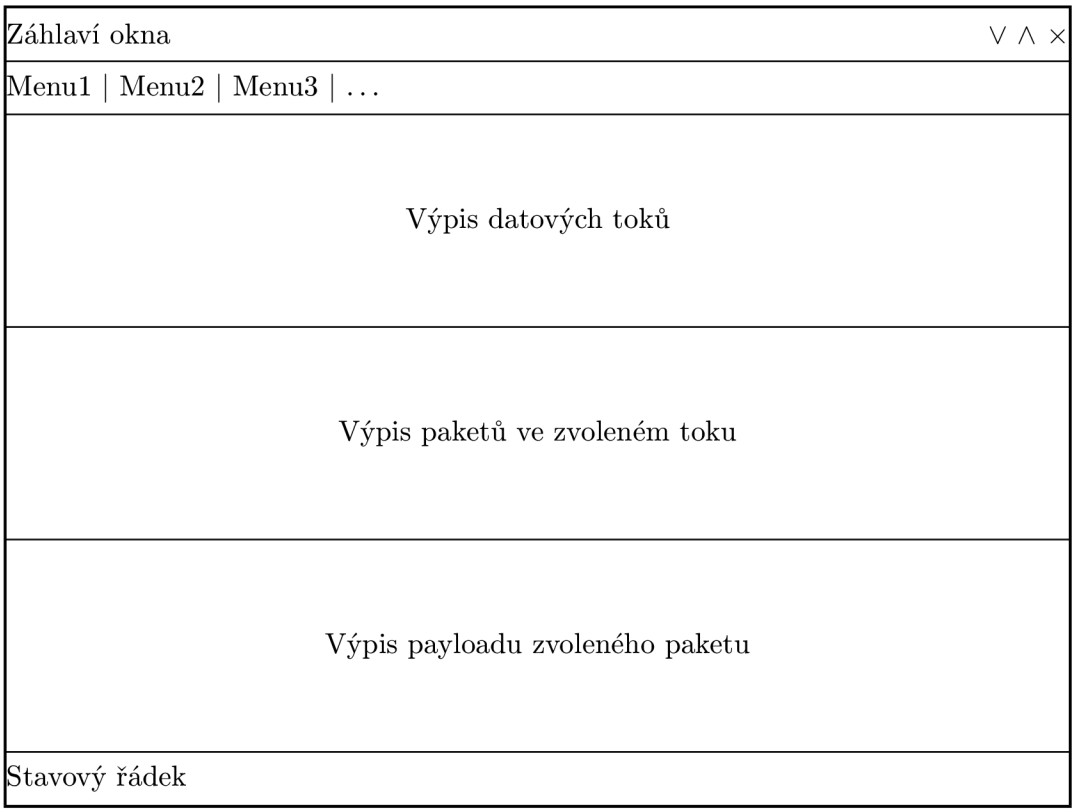
## 3.3 Návrh grafického uživatelského rozhraní

Navrhované GUI by mělo bý co možná nejjednodušší, aby jeho používání bylo pokud možno intuitivní. Proto byl zvolen model hlavního okna obsahujícího hlavní nabídku, tři podokna pro zobrazení výsledků a stavová lišta. Toto rozvržení je velmi podobné například GUI z nástroje `Wireshark` [19], kde se velmi osvědčilo.

Rozložení hlavního okna je naznačeno na Obrázku 3.3. Záhlaví okna by mělo obsahovat běžné ovládací prvky a případně jméno otevřeného souboru. V menu by se měly objevit běžné ovládací prvky pro otevření a uložení souboru, zobrazení statistik analyzovaného souboru, spuštění analýzy, zobrazení nastavení apod. V prvním podokně by měla být zobrazena jednoduchá tabulka obsahující informace o jednotlivých datových tocích, výsledky jednotlivých analyzátorů a pole pro zápis výsledku zadaného uživatelem. Druhé podokno by mělo, opět formou jednoduché tabulky, zobrazovat pakety ve zvoleném datovém toku. Poslední podokno by mělo sloužit k zobrazení payloadu vybraného paketu a to jak ve formě textu, tak pomocí hexadecimálních hodnot jednotlivých bytů payloadu. Ve stavovém řádku by měly být zobrazovány informace související s momentální činností aplikace.

Veškerá nastavení systému by měla být zobrazena v jednom okně přístupném pomocí hlavního menu. Toto okno by mělo jednak obsahovat nastavení pro vestavěné části systému, a jednak by mělo zobrazovat nastavení pro jednotlivé (i uživatelem přidané) analyzátory. Aby bylo možnot vykreslit části okna nutné pro zadání nastavení přidaných modulů, je nutné, aby dané moduly definovaly vyžadovaná nastavení (více podrobností viz Dokumentace v příloze B).

Zobrazované výsledky analýzy by měly obsahovat následující údaje: pořadové číslo datového toku, jeho začátek a konec, five-tuple, pakety obsažené v datovém toku v jednom a druhém směru, výsledky analýz jednotlivých analyzátorů a pole pro celkový výledek anotace, které by mělo bý editovatelné.




Obrázek 3.3: Návrh rozložení hlavního okna aplikace

## 3.4 Navrhovaný formát výsledků anotace

Výsledky anotace by měly být uloženy ve výsledkových souborech nebo zobrazeny v grafickém uživatelském rozhraní. Pro každý datový tok by měly být uvedeny alespoň tyto údaje: čas počátku a konce toku, adresa a port zdroje, adresa a port cíle, transportní protokol, pakety obsažené v daném toku, dílčí výsledky a konečný výsledek. Navrhované formáty výsledkových souborů jsou prostý text nebo XML.

### 3.4.1 Textový formát

Každý datový tok je uložen na jednom řádku ve formátu:

```
<s> <k> <z_IP> <z_p> <c_IP> <c_p> <pr> [<k_p>] [<s_p>] {<v>} <k_v>
```

Tabulka č. 3.1 popisuje význam jednotlivých polí. Na konci výpisu mohou být umístěny stručné statistiky.

### 3.4.2 XML formát

Druhá varianta je výpis výsledků do XML souboru se strukturou naznačenou na následujícm příkladu (DTD výsledkového XML je v příloze C.):

```
<?xml version="1.0" encoding="utf-8"?>
<results>
```

| pole | význam | poznámka |
|---|---|---|
| <s> | čas počátku toku | ve formátu RRRR-MM-DD HH:MM:SS.MSMSMS |
| <k> | čas konce toku | ve stejném formátu jako <s> |
| <z_IP> | IP adresa zdroje | ve formátu odpovídajícímu IP verze 4 a 6 |
| <z_p> | zdrojový port | |
| <c_IP> | IP adresa cíle | ve stejném formátu jako <z_IP> |
| <c_p> | cílový port | |
| <pr> | transportní protokol | TCP nebo UDP |
| <k_p> | seznam čísel paketů od klienta | položky odděleny čárkou bez mezer |
| <s_p> | seznam čísel paketů od serveru | stejné jako <k_p> |
| <v> | seznam dvojic výsledků | '<jméno_analyzátoru>': '<výsledek>' (položky odděleny čárkami a mezerami) |
| <k_v> | konečný výsledek | |

Tabulka 3.1: Význam jednotlivých polí v textové reprezentaci výsledků

```
<flow_list>
  <flow>
    <start_time/>
    <end_time/>
    <five_tuple/>
    <client_packet_list>
      <packet/>
      ...
    </client_packet_list>
    <server_packet_list/>
    <result_list>
      <result/>
      ...
    </result_list>
    <user_result>
  </flow>
  ...
</flow_list>
</results>
```

Tabulka 3.2 popisuje význam atributů jednotlivých elementů

Zobrazení výsledků v grafickém uživatelském rozhraní je popsáno v kapitole 3.3.

| **atribut** | **elementy obsahující tento atribut** | **poznámka** |
|---|---|---|
| `seconds` | `<start_time>`<br>`<end_time>`<br>`<packet>` | počet sekund od 1. 1. 1970 |
| `microseconds` | `<start_time>`<br>`<end_time>`<br>`<packet>` | počet mikrosekund po `seconds` |
| `formated_string` | `<start_time>`<br>`<end_time>`<br>`<packet>` | čas ve stejném formátu<br>jako v textové reprezentaci |
| `id` | `<packet>` | pořadové číslo paketu |
| `source_address` | `<five_tuple>` | zdrojová IP adresa |
| `destination_address` | `<five_tuple>` | cílová IP adresa |
| `source_port` | `<five_tuple>` | zdrojový port |
| `destination_port` | `<five_tuple>` | cílový port |
| `transport_protocol` | `<five_tuple>` | transportní protokol |
| `code` | `<result>`<br>`<user_result>` | výsledek |
| `analyzer` | `<result>` | jméno analyzátoru |
| `count` | všechny podelementy<br>elementu `statistics` | počet paketů |
| `bytes` | `total_packets`<br>`ipv4_packets`<br>`ipv6_packets`<br>`tcp_packets`<br>`udp_packets` | počet bytů |

Tabulka 3.2: Možné atributy jednotlivých elementů v XML reprezentaci výsledku

# Kapitola 4

# Implementace

## 4.1 Zvolený programovací jazyk

Z důvodu přenositelnosti programu byl jako implementační jazyk zvolen Python, který je platformně nezávislý (stačí mít na dané platformě nainstalován jeho interpret). Později se tato volba ale ukázala jako ne příliš šťastná zejména z důvodu rychlosti použitých algoritmů. Reimplementace celého systému v jiném jazyce se však ukázala jako velmi časově náročná a proto byl nakonec systém ponechán v Pythonu. Aby byla alespoň částečně kompenzována rychlost, bylo v maximální možné míře využito knihoven pro Python napsaných v jiném jazyce (např. C). Protože většina těchto knihoven nepodporuje python verze 3.x, bylo nutno použít starší verzi 2.7.

## 4.2 Implementace jednotlivých částí systému

Systém byl implementován podle návrhu uvedeného v kapitole 3.2 pouze s drobnými změnami na místech, kde se návrh ukázal jako nevyhovující.

### 4.2.1 Modul `Splitter`

Tento modul využívá knihovnu `pcapy` pro čtení zdrojových souborů ve formátu `pcap`. Postupně načítá jednotlivé pakety ze souboru a postupně z nich parsuje potřebná data a ukládá je do objektů. Na základě získaných dat jsou pakety tříděny podle five-tuple a přiřazovány do objektů reprezentujících datové toky. Ty jsou potom ukládány do slovníku a jejich five-tuple jsou použity jako klíče. Slovníky v jazyce Python jsou vysoce optimalizovány a pracují poměrně efektivně, pokud je v nich potřeba vyhledávat a obsahují velké množství záznamů, což je přesně případ tohoto modulu.

Pokud je některý datový tok předem stanovenou dobu nečinný nebo v něm dojde k uzavření TCP spojení, je ukončen a předán modulu `analyzers`, kde je zpracován.

### 4.2.2 Modul `Analyzers`

Tento modul obsahuje třídu `Analyzer`, která slouží jako základní model analyzátoru a všechny ostatní třídy reprezentující vlastní analyzátory jsou od této třídy odvozeny. Tím je zajištěno, že každý analyzátor implementuje potřebné rozhraní a lze tak vytvářet nové analyzátory jednoduchým naděděním této třídy a překrytím příslušných metod, které definují chod vlastního analyzátoru. Tvorba nových modulů je podrobně popsána v příloze B.

Dále tento modul zajišťuje správné předávání a typovou kontrolu nastavení, která jednotlivé analyzátory potřebují. Kontrola konkrétních hodnot nastavení, např. intervaly, do kterých musí spadat číselné hodnoty, jsou však už v režiji samotných analyzátorů.

Tento modul se stará o inicializaci všech analyzátorů před začátkem vlastní analýzy a zajišťuje i jejich správné ukončení potom, co proběhne analýza.

Původně bylo zamýšleno spouštět metody analyzátorů pro klasifikaci jednotlivých toků paralelně jako vlákna, ale ukázalo se, že tyto metody běží velice krátkou dobu, ale jsou mnohokrát spouštěny, a proto je režije vzniklá vytvořením jednotlivých vláken větší než časová úspora z běhu analyzátorů paralelně.

Do tohoto modulu jsou postupně zasílány datové toky, které předává všem analyzátorům. Ty provádějí vlastní analýzu a postupně si ukládají výsledky. Poté co modul `Splitter` zavolá metodu `Finalize`, proběhne ukončení všech analyzátorů. Po ukončení vrátí analyzátory výsledky, které jsou spojeny do jedné struktury a ta je předána modulu `Printer`.

### 4.2.3 Modul `Statistics`

V tomto modulu je implementován analyzátor, který využívá ke klasifikaci datových toků strojové učení – konkrétně algoritmus Support Vector Machine. Tento algoritmus bylo původně zamýšleno implementovat pomocí použití knihovny PyML, ale během vývoje se ukázalo, že tato knihovna neimplementuje funkce potřebné pro klasifikaci, ale slouží spíše pro testování samotných SVM. Proto byla nakonec využita knihovna libSVM, konkrétně její implementace algoritmu C-SVC s použitím jádra RBF. Optimální parametry ($C = 128$, $\gamma = 0,5$) klasifikátoru byly zjištěny za použití *cross-validace* pomocí skriptu `easy.py`, který je součástí balíku libSVM. Klasifikátor je užíván v pravděpodobnostním módu, tzn. že pro každou třídu vrací pravděpodobnost, že do ní klasifikovaný prvek spadá. Díky tomu lze porovnávat výsledky pro oba směry datových toků a vybrat ze dvou výsledků ten pravděpodobnější.

Jako tréningová množina byla použity data z [5]. Pro získání statistik z datových toků byl použit nástroj `crl_pay` od autorů práce [8]. Dále byly z těchto statistik vybrány pouze čísla portů, transportní protokol, maximální, minimální a průměrná velikost paketu, směrodatná odchylka velikostí paketů, počet jednotlivých TCP flagů obsažených v toku a velikosti prvních deseti paketů podle zjištění uvedených ve výše zmíněné práci. Z této tréningové množiny bylo poté vybráno asi 10 000 vzorků toků tak, aby byly jednotlivé protokoly zastoupeny pokud možno rovnoměrně. Tyto vzorky byly poté použity k natrénování klasifikátoru.

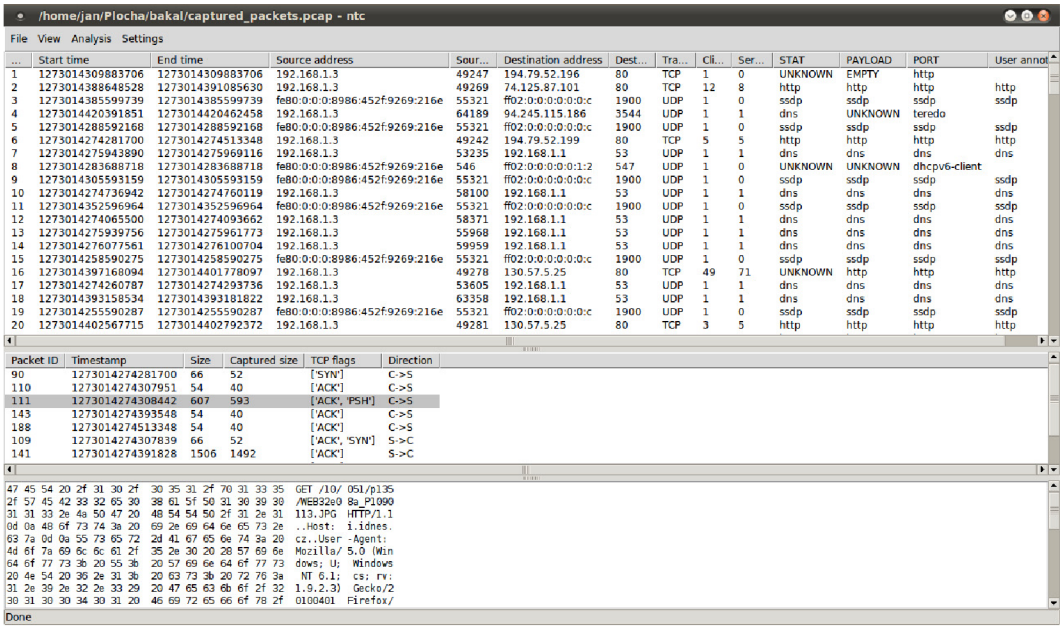
Tato tréningová množina se ale ukázala jako nereprezentativní, protože obsahuje jen 10 různých protokolů, což je poměrně málo. Navíc počet vzorků některých protokolů byl velmi nízký (méně než 100). Proto bylo přistoupeno k doplnění tréningové množiny o ručně anotované vzorky získané z kolejní sítě.

### 4.2.4 Modul `Payload`

Při implementaci analyzátoru payloadu byly podle doporučení v [2] vypuštěny některé vzory převzaté z nástroje `l7-filter` (např. vzor pro protokol finger), protože tyto vzory jsou známé generovámín mnoha tzv. *false positives*. Navíc tyto protokoly nejsou příliš časté, takže je možné je bez větší ztráty přesnosti vypustit. Většinou se podle poznatků ve výše citovaném článku po tomto kroku přesnost analýzy naopak zvýší.

### 4.2.5 Grafické uživatelské rozhraní

Pro implementaci GUI byly zvoleny knihovny `tkinter` a `ttk`, které jsou vestavěnou součástí jazyka Python. Tyto knihovny jsou obálky pro Tk/Tcl. Aby bylo dosaženo nepatrně lepšího vzhledu aplikace, bylo využito tzv. *themingu*, který umožňuje použití nativního vzhledu prvků použitého v operačním systému. Bohužel tato funkce pracuje hlavně na Windows a Mac OS, na Linuxu jsou využívána vestavěná témata, která nemají nativní vzhled. I tak ovšem vypadají nepatrně lépe, než klasický Motif. Tato funkce je bohužel podporována pouze u Tk/Tcl verze 8.5, která ještě běžně není ve standardních distribucích operačních systémů a je třeba ji doinstalovat. Na obrázku 4.1 je vidět hlavní okno uživatelského rozhraní v systému Linux. Tabulky v prvních dvou podoknech jsou implementovány pomocí obálky




Obrázek 4.1: Ukázka grafického uživatelského rozhraní aplikace

`TableListWrapper` [17], která je založena na `TableList` [12] widgetu pro Tk. Tento widget zobrazuje tabulku, kterou lze řadit podle sloupců a umožňuje vybírat záznamy v řádcích jako celek. Navíc umožňuje editaci některých polí v tabulce, takže je možné, aby uživatel ručně zadal výsledky anotace.

## 4.3 Nastavení nástroje

Vytvořený nástroj obsahuje několik možných nastavení, která zásadně ovlivňují jeho práci a výsledky. Všechna hlavní nastavení jsou uložena v souboru `config.cfg`. Ukázka tohoto souboru je v příloze B. Konfigurační soubor využívá strukturu podobnou INI souborům na MS Windows. Nastavení jsou rozdělena do sekcí, jejichž jména jsou uzavřena mezi znaky `[]` a měla by začínat velkým písmenem. Každá tato sekce pak obsahuje na jednotlivých řádcích nastavení ve tvaru:

```
<jméno> = <hodnota>
```

Jméno nastavení může obsahovat alfanumerické znaky a nemělo by obsahovat mezery. Hodnota nastavení může být typu `Integer`, `Float`, `String` nebo `Boolean`. Tyto typy jsou v konfiguračním souboru zapsány jako řetězce a vlastní typová kontrola se řídí pravidly Pythonu pro převod do daného typu.

Jednotlivé sekce v konfigurační souboru jsou: `Splitter` pro nastavení týkající se sdružování paketů do datových toků, `Analyzers` pro nastavení aktivních analyzátorů, `Printer` pro nastavení týkající se výsledků a jedna samostatná sekce pro každý analyzátor.

Sekce `[Analyzers]` je poněkud specifická – udává, jaké analyzátory jsou zapnuty. Jména nastavení musí bý shodná s názvy modulů implementujících analyzátory. Hodnoty jednotlivých nastavení jsou typu Boolean a indikují, zda je daný analyzátor aktivován. Pro každý záznam v sekci `[Analyzers]` musí existovat samostatná sekce se jménem analyzátoru, která obsahuje záznam `weight` a jiná nastavení registrovaná v modulu analyzátoru.

Jednotlivá nastavení, jejich význam a jejich očekávané hodnoty jsou popsány v příloze B:

Každý analyzátor používá ještě vlastní soubor s nastaveními. Pro modul `Port` je to soubor `ports.txt`, který obsahuje seznam čísel portů a jim přiřazených protokolů. Na každém řádku je záznam ve formátu:

```
<jméno_protokolu>/<číslo_portu>/<tcp|udp>
```

Podobně modul `Payload` používá soubor `patterns.txt`, který na každém řádku obsahuje jméno protokolu a regulární výraz, definující hledaný vzor ve formátu:

```
<jméno_protokolu>;;<regulární výraz>
```

Syntaxe regulárního výrazu je shodná se syntaxí vzorů použitých v nástroji `l7-filter` [9].

Modul `Statistics` používá soubor `model`, který obsahuje definici klasifikátoru ve formátu libSVM.

## 4.4 Optimalizace

Během vývoje bylo nutno kvůli neúměrně dlouhé době analýzy přistoupit k několika optimalizacím. Nejdůležitější z nich bylo reimplementování datových struktur uchovávajících mezivýsledky analýzy. Ty byly původně implementovány jako seznamy, ale vzhledem k faktu, že během rozřazování paketů do datových toků jsou v těchto seznamech vyhledávány již existující toky a je tudíž třeba je sekvenčně procházet, byla asymptotická časová složitost algoritmu kvadratická. Díky reimplementaci těchto struktur jako slovníků, které jsou v Pythonu navíc optimalizovány, které není třeba sekvenčně procházet, bylo pro tuto část algoritmu dosaženo lineární asymptotické časové složitosti. To vedlo k výrazné časové úspoře, protože zkoumaná vstupní data běžně obsahují několik milionů paketů.

Další časové úspory bylo dosaženo při parsování dat v jednotlivých paketech. Protože při tomto procesu je třeba přistupovat k některým datům na úrovni bitů, byla původně zvolena knihovna `bitstring` pro procházení těchto dat. Později se ovšem ukázalo, že tato knihovna reprezentuje binární data jako seznamy a proto je práce s nimi opět velmi pomalá. Proto byla tato funkce reimplementována pomocí vestavěné knihovny `struct`, která umožňuje přistupovat k datům po bytech. Jednotlivé bity pak byly pak získávány pomocí jednoduché aritmetiky.

Další časové úspory bylo dosaženo přidáním možnosti omezit délku payloadu, ve kterém jsou vyhledávány vzory, protože ve většině případů jsou tyto vzory obsaženy na začátku payloadu a pokud nalezeny nejsou, je zbytečné procházet celý obsah paketu.

Drobných paměťových úspor bylo dosaženo přidáním mechanismu, který po zpracování několika paketů vyhledá ukončené datové toky a pošle je na zpracování. Díky tomu již není třeba udržovat v paměti informace o těchto tocích a dochází k jejímu uvolnění. Nastavení četnosti těchto kontrol je ponecháno na uživateli, protože příliš časté kontroly jsou časově náročné a zdržují běh algoritmu.

# Kapitola 5

# Experimenty

S vytvořeným nástrojem byla provedena řada experimentů zaměřená na zjištění časové a paměťové náročnosti analýzy ve vztahu ke vstupním datům. Dále byly provedeny experimenty s cílem ověřit úspěšnost jednotlivých analyzátorů (ve smyslu počtu identifikovaných toků v poměru k celkovému počtu toků) a jejich přesnost (nakolik jsou výsledky úspěšně identifikovaných toků pravdivé).

Testy prováděné na nástroji byly provedeny na třech různých vzorcích dat:

**vzorek 1** Tento vzorek obsahuje velmi malé množství paketů (asi 1 400) a byl pořízen na kolejní síťi VUT 5. 5. 2010. Byl odchytáván necelé 4 minuty a obsahuje kolem 250 datových toků s plným payloadem.

**vzorek 2** Tento vzorek obsahuje asi 130 000 paketů a také pochází z kolejní síťě VUT. Byl pořízen 5. 5. 2011, trvá zhruba 16 minut a obsahuje asi 11 500 datových toků s plným payloadem

**vzorek 3** Tento vzorek je výřezem z dumpu, který pochází z WIDE backbone network [18] ze vzorkového bodu F z 30. 12. 2010. Vzorek 3 obsahuje prvních 1 000 000 paketů z tohoto dumpu. Pakety v tomto vzorku neobsahují payload (kromě DNS paketů, které mají malou část payloadu ponechánu) a jsou rozděleny do zhruba 65 000 datových toků.

## 5.1 Časová náročnost

Časová náročnost analýzy byla měřena na konzolové verzi nástroje, aby nedošlo ke zkreslení výsledků vláknem grafického uživatelského rozhaní. Pro vlastní měření byla použita vestavěná knihovna `cProfile`. Tato knihovna měří tzv. procesorový čas, tj. pouze čas, po který je procesu skutečně přidělen procesor. Tím je značně omezena chyba měření, ovšem samotné měření proces nepatrně zpomaluje a mírně tak zkresluje výsledky. Nástroj byl během měření spouštěn s následujícími nastaveními modulu `Splitter` (ostatní moduly čas výpočtu neovlivňují): `killintervat = 1000`, `tcptimeout = 1000000` a `udptimeout = 1000000`.

Tabulka 5.1 ukazuje výsledky jednotlivých měření. Z výsledků je patrné, že asymptotická časová složitost algoritmu je lineární, což bylo předpovídáno na základě analýzy implementovaného algoritmu.

| | Vzorek 1 | | Vzorek 2 | | Vzorek 3 | |
|---|---|---|---|---|---|---|
| **Forma výpisu výsledků** | **text** | **XML** | **text** | **XML** | **text** | **XML** |
| Celkový čas běhu nástroje | 1,836 | 2,468 | 61,226 | 100,134 | 292,556 | 669,799 |
| Celkový čas analýzy | 0,921 | 0,921 | 59,334 | 59,334 | 282,955 | 282,955 |
| Čas analýzy portů | 0,053 | 0,053 | 11,120 | 11,120 | 40,768 | 40,768 |
| Čas analýzy payloadu | 0,348 | 0,348 | 22,220 | 22,220 | 0 | 0 |
| Čas analýzy statistik | 0,229 | 0,229 | 8,548 | 8,548 | 48,627 | 48,627 |
| Čas zápisu výsledků | 0,017 | 0,679 | 0,765 | 40,496 | 5,749 | 379,792 |

Tabulka 5.1: Výsledky měření časové náročnosti analýzy

## 5.2 Paměťová náročnost

Paměťová náročnost analýzy byla opět zjišťována na konzolové verzi nástroje. Paměťové nároky grafické verze nástroje jsou ještě o něco vyšší, protože je třeba držet v paměti data, která jsou zobrazena v obou `TableList` widgetech. Data uložená v paměti během analýzy jsou několikrát větší než vstupní data, protože ačkoli neobsahují tolik informací, nejsou, narozdíl od dat vstupních, efektivně uložena v přímé posloupnosti bytů, ale v instancích různých objektů.

Navíc, aby byla zachována možnost vytvářet přídavné moduly, které implementují víceprůchodové analyzátory, je třeba, aby každý analyzátor vracel všechny výsledky a až po dokončení své činnosti a ne průběžně, takže se paměťová náročnost násobí počtem spuštěných analyzátorů.

Měření paměťové náročnosti bylo provedeno za použití knihoven `pympler` a `PySizer`. Tyto knihovny byly použity tak, aby měřily velikost použitých objektů v momentě, kdy je předpokládaná paměťová náročnost nejvyšší – tj. těsně před a po skončení vlastní analýzy a před počátkem vypisování výsledků.

Naměřené hodnoty jsou prezentovány v tabulce 5.2. Ze zjištěných hodnot vyplývá, že asymptotická prostorová složitost je také lineární. I tak ale paměťové nároky nástroje vzhledem k počtu analyzovaných paketů nepříjemně rychle rostou.

| | Vzorek 1 | Vzorek 2 | Vzorek 3 |
|---|---|---|---|
| Maximální paměťová náročnost | 8 MB | 248 MB | 1 542 MB |

Tabulka 5.2: Výsledky měření paměťové náročnosti

## 5.3 Úspěšnost analyzátorů

Pojmem úspěšnost je v tomto kontextu myšlen počet datových toků, pro které analyzátor vrátil jakýkoli výsledek. U statistického analyzátoru tato hodnota značně zavisí na nastavení hodnoty `probability`. Pokud se tato hodnota nastaví příliš vysoká, je úspěšnost analyzátoru velmi nízká, a pokud se naopak nastaví příliš nízká, značně klesne přesnost analyzátoru, protože začne vznikat mnoho tzv. *false positives*.

Úspěšnost analyzátoru čísel portů je obecně poměrně vysoká, ale sama o sobě není tato metoda dostatečně přesná a může tak také vznikat mnoho *false positives*.

Tabulka 5.3 obsahuje procentuální hodnoty úspěšnosti pro jednotlivé analyzátory, pro statistický analyzátor je vždy ještě uvedena hodnota `probability`, pro kterou byla data testována.

| **Analyzátor** | | **Vzorek 1** | **Vzorek 2** | **Vzorek 3** |
|---|---|---|---|---|
| Port | | 100,0 % | 93,3 % | 83,5 % |
| Payload | | 84,4 % | 60,3 % | Netestováno |
| Statistics | 0,9 | 63,2 % | 60,1 % | 36,4 % |
| | 0,8 | 90,4 % | 71,8 % | 50,7 % |
| | 0,7 | 92,0 % | 81,2 % | 56,6 % |
| | 0,6 | 92,4 % | 86,9 % | 68,9 % |

Tabulka 5.3: Výsledky měření úspěšnosti jednotlivých analyzátorů

## 5.4 Přesnost analyzátorů

Přesnost je brána jako počet správně klasifikovaných toků oproti ručně ověřeným výsledkům. Vzorek 3 neobsahuje payload paketů, a proto nebylo možné ověřit přesnost na něm testovaných výsledků. Tabulka 5.4 ukazuje procentuální hodnoty přesnosti jednotlivých analyzátorů.

| **Analyzátor** | | **Vzorek 1** | **Vzorek 2** |
|---|---|---|---|
| Port | | 99,6 % | 97,4 % |
| Payload | | 99,5 % | 87.8 % |
| Statistics | 0,9 | 84,8 % | 55,9 % |
| | 0,8 | 91,2 % | 60,4,% |
| | 0,7 | 91,7 % | 62,6 % |
| | 0,6 | 92,2 % | 63,1 % |

Tabulka 5.4: Výsledky měření přesnosti jednotlivých analyzátorů

## 5.5 Úspěšnost a přesnost automatické anotace

Z předchozích testů bylo empiricky zjištěno, že nejvýhodnější nastavení pro automatickou anotaci jsou: `probability = 0,8`, `Port weight = 40`, `Payload weight = 60`, `Statistics weight = 30` a `threshold = 60`. Jednotlivé vzorky byly za použití těchto nastavení oanotovány. Úspěšnost a přesnost anotace ukazuje tabulka 5.5. Tabulka 5.6 obsahuje preocentuální zastoupení jednotlivých protokolů v anotované části vzorků.

| | **Vzorek 1** | **Vzorek 2** | **Vzorek 3** |
|---|---|---|---|
| Úspěšnost | 87,6 % | 65,3 % | 10,9 % |
| Přesnost | 99,5 % | 90,6 % | Netestováno |

Tabulka 5.5: Výsledky automatické anotace

| Protokol | **Vzorek 1** | **Vzorek 2** | **Vzorek 3** |
|---|---|---|---|
| aim/icq | X | 0,01 % | X |
| bittorrent | X | 0,12 % | X |
| dhcp | X | 0,04 % | X |
| dns | 44,29 % | 30,91 % | 87,11 % |
| edonkey | X | 0,15 % | X |
| http | 23,29 % | 3,04 % | 12,84 % |
| jabber | X | 0,03 % | X |
| kugoo | X | 0,11 % | X |
| nbns | X | 12,22 % | 0,05 % |
| ntp | X | 0,26 % | X |
| pop3 | X | 0,07 % | X |
| qq | X | 0,05 % | X |
| skype | 0,46 % | 9,34 % | X |
| smb | 0,46 % | 5,86 % | X |
| ssdp | 31,50 % | 37,67 % | X |
| ssl | X | 0,12 % | X |

Tabulka 5.6: Procentuální zastoupení jednotlivých protokolů v úspěšně anotovaných tocích

# Kapitola 6

# Závěr

V rámci této práce byl navržen, vytvořen a otestován nástroj pro anotaci síťového provozu. Časové a paměťové nároky tohoto nástroje, i přes snahu o optimalizace, jsou poměrně vysoké a značně tak omezují jeho použití. Za současného stavu je třeba analyzované soubory rozdělit na menší části, které neobsahují více než milion paketů nebo 500 MB dat.

Toto nepříjemné omezení by se dalo vyřešit reimplementací modulu sdružujícího pakety do datových toků v nějakém nižším programovacím jazyce (např. C), kde by byla data uchovávána v efektivnějších strukturách, a použitím relační databáze pro ukládání výsledků (odpadne tak nutnost držet v paměti velké množství dat).

Kromě výše zmíněných možných rozšíření a vylepšení by bylo vhodné nástroj doplnit o další analyzátory pracující na jiných principech, které by výrazně přispěly ke zvýšení kvality automatické anotace. Tato rozšíření by mělo být velmi snadné do nástroje přidat díky připravenému rozhraní pro nové moduly.

Úspěšnost automatické anotace je značně ovlivněna použitými nastaveními a je třeba je vyladit pro konkrétní vstupní data. Optimální hodnoty pro data z páteřní linky, kde vystupuje velké množství hostů, chybí část dat v tocích kvůli asymetrickému směrování a celkově je hustší provoz, budou zcela jistě jiné než pro data pocházející z malé domácí sítě, kde je celkový objem dat několikanásobně nižší, a kde vystupuje jen několik málo hostů.

Z provedených experimentů je patrné, že analyzátor payloadu paketů je klíčová komponenta celého nástroje, a při použití nástroje na anonymizovaných datech, kde tato analýza není možná, je celková úspěšnost mizivá. Také je patrné, že tento analyzátor dosahuje velmi vysoké přesnosti, i přes generování mnoha false positives některými vzory.

Dále bylo zjištěno, že úspěšnost i přesnost statistického analyzátoru je poměrně nízká. To je pravděpodobně způsobeno nekvalitními tréningovými daty, která obsahují velmi málo protokolů (pouze 18) a navíc pro některé z nich obsahují nedostatečné množství vzorků. To způsobuje, že dané protokoly buď nejsou nalezeny vůbec, nebo naopak produkují velké množství false positives. Tento problém by mělo být možné výrazně omezit přetrénováním analyzátoru na kvalitních tréningových datech.

Bohužel se ukázalo, že získat kvalitní a reprezentativní vzorová data je velmi obtížné. Jednou z možností získání těchto dat, je itaretivně rozšiřovat vzorové sady nástroje. Tento postup by znamenal shromáždění maxima dotupných vzorů, provedení analýzy za použití těchto vzorů, ruční kontrolu a opravení chyb ve výsledcích a použití získaných výsledků jako nových vzorů.

Největším přínosem vytvořeného nástroje je možnost jeho použití při ruční anotaci síťového provozu, kterou velmi zjednodušuje, protože většinu potřebných dat sdružuje na

jednom místě, a může ji i postatně urychlit automatickým klasifikováním datových toků

Z podobných existujících projektů lze tento srovnat pouze se systémem GTVS, který je jako jediný také zaměřen na podporu ruční anotace. Jeho úspěšnost je sice mnohonásobně vyšší, ale jeho velmi složitá instalace a konfigurace velmi znesnadňují jeho nasazení. Instalace vytvořeného nástroje je o poznání jednodušší.

# Literatura


[1] *Bro Intrusion Detection System* [online]. rev. May 5, 2011 [cit. 9. května 2011]. Dostupné na: <http://www.bro-ids.org/>.

[2] Canini, M., Li, W., Moore, A. W. et al. GTVS: Boosting the Collection of Application Traffic Ground Truth. In Papadopouli, M., Owezarski, P. a Pras, A. (ed.). *Traffic Monitoring and Analysis*. Berlin Heidelberg: Springer-Verlag, 2009. S. 54–63. Lecture Notes in Computer Science, sv. 5537. Dostupné na: <http://www.springerlink.com/content/74096w567m49m5w6/fulltext.pdf>.

[3] Deering, S. a Hinden, R. *RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification* [online]. rev. December 1998 [cit. 6. května 2011]. Dostupné na: <http://www.faqs.org/rfcs/rfc2460.html>.

[4] Erman, J., Arlitt, M. a Mahanti, A. Traffic Classification Using Clustering Algorithms. In *MineNet '06 Proceedings of the 2006 SIGCOMM workshop on Mining Network* [online]. New York, USA: [b.n.], 2006 [cit. 11. listopadu 2010]. Dostupné na: <http://portal.acm.org/ft_gateway.cfm?id=1162679&type=pdf&CFID=22236327&CFTOKEN=32669154>.

[5] Gringoli, F., Salgarelli, L., Dusi, M. et al. GT: Picking Up the Truth From The Ground For Internet Traffic: The UNIBS Anonymized 2009 Internet Traces. In *SIGCOMM Computer Communication Review* [online]. New York, USA: [b.n.], 2009, rev. Mar 18, 2010 [cit. 2. dubna 2011]. S. 13–18. Dostupné na: <http://www.ing.unibs.it/ntw/tools/traces>.

[6] *HiPPIE* [online]. rev. 2009-07-17 [cit. 9. května 2011]. Dostupné na: <http://sourceforge.net/projects/hippie/>.

[7] Karagiannis, T., Papagiannaki, K. a Faloutsos, M. BLINC: Multilevel Traffic Classification in the Dark. In *SIGCOMM '05 Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communactions* [online]. New York, USA: [b.n.], 2005 [cit. 11. listopadu 2010]. Dostupné na: <http://portal.acm.org/ft_gateway.cfm?id=1080119&type=pdf&CFID=22236327&CFTOKEN=32669154>.

[8] Kim, H., Claffy, K., Formenkov, M. et al. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practises. In *CoNEXT '08 Proceedings of the 2008 ACM CoNEXT Conference* [online]. New York, USA: [b.n.], 2008 [cit. 9. listopadu 2010]. Dostupné na: <http://portal.acm.org/ft_gateway.cfm?id=1544023&type=pdf&CFID=22236327&CFTOKEN=32669154>.

[9] *L7-filter* [online]. rev. January 25, 2011 [cit. 9. května 2011]. Dostupné na: <http://l7-filter.clearfoundation.com/>.

[10] Moore, A. W. a Papagiannaki, K. Toward the Accurate Identification of Network Applications. In Dovrolis, C. (ed.). *Passive and Active Network Measurement* [online]. Berlin Heidelberg: Springer-Verlag, 2005 [cit. 11. listopadu 2010]. S. 41–54. Lecture Notes in Computer Science, sv. 3431. Dostupné na: <http://www.springerlink.com/content/re7ej0uj7eep2htl/fulltext.pdf>.

[11] Moore, A. W. a Zuev, D. Internet Traffic Classification Using Bayesian Analysis Techniques. In *SIGMETRICS '05 Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* [online]. New York, USA: [b.n.], 2005 [cit. 11. listopadu 2010]. Dostupné na: <http://portal.acm.org/ft_gateway.cfm?id=1064220&type=pdf&CFID=22236327&CFTOKEN=32669154>.

[12] Nemethi, C. *The Multi-Column Listbox Package TableList* [online]. rev. January 9, 2011 [cit. 20. dubna 2011]. Dostupné na: <http://www.nemethi.de>.

[13] *OpenDPI* [online]. rev. 8. 10. 2010 [cit. 9. května 2011]. Dostupné na: <http://opendpi.org/>.

[14] Postel, J. *RFC 768 - User Datagram Protocol* [online]. rev. 28 August 1980 [cit. 6. května 2011]. Dostupné na: <http://www.faqs.org/rfcs/rfc768.html>.

[15] *RFC 791 - Internet Protocol* [online]. rev. September 1981 [cit. 6. května 2011]. Dostupné na: <http://www.faqs.org/rfcs/rfc791.html>.

[16] *RFC 793 - Transmission Control Protocol* [online]. rev. Septemner 1981 [cit. 6. května 2011]. Dostupné na: <http://www.faqs.org/rfcs/rfc793.html>.

[17] Walzer, K. *TableListWrapper* [online]. rev. 2010-07-26 [cit. 20. dubna 2011]. Dostupné na: <http://tkinter.unpythonic.net/wiki/TableListWrapper>.

[18] *MAWI Working Group Traffic Archive* [online]. [cit. 25. dubna 2011]. Dostupné na: <http://mawi.wide.ad.jp/mawi/samplepoint-F/2010/201012301400.html>.

[19] *Wireshark* [online]. rev. 2011 [cit. 22. března 2011]. Dostupné na: <http://www.wireshark.org/>.

# Příloha A

# Obsah CD

Přiložené CD obsahuje:

- `install` – externí knihovny
- `ntat` – vyvinutý nástroj NTAT
- `results` – výsledkové soubory
- `testing_data` – vstupní soubory se vzorky
- `TeX` – zdrojové soubory dokumentace a této práce
- `training_data` – tréningová data, skript pro tréning klasifikátoru a soubor `range`, který byl použit při scalingu tréningových dat.
- `Dokumentace.pdf` – dokumentace
- `README` – stručné instrukce
- `bp-xholak01.pdf` – tato práce

# Příloha B

# Dokumentace

## B.1 Úvod

Program NTAT neboli Network Traffic Annotation Tool slouží k urychlení a zjednodušení manuální anotace síťového provozu. Obsahuje tři zálkadní analyzátory pro automatickou anotaci, které používají analýzu čísel portů, payloadu a transportních statistik datových toků, a rozhraní umožňující tvorbu a zapojení dalsích modulů. Dále tento nástroj nabízí uživatelské rozhraní, které umožňuje manuální prohlídku obsahu jednotlivých datových toků. Výsledky anotace je možné uložit jako prostý text nebo XML.

## B.2 Požadavky

- Python verze 2.7 – http://www.python.org/download/releases/2.7/
- knihovna libpcap nebo WinPcap – http://www.tcpdump.org/ nebo http://www.winpcap.org/
- knihovna pcapy – http://oss.coresecurity.com/projects/pcapy.html
- knihovna libSVM a její rozhraní pro Python – http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Pro používání grafického uživatelského rozhraní je navíc třeba:

- Tk/Tcl verze 8.5 – http://www.tcl.tk/software/tcltk/8.5.html
- moduly TableList a TableListWrapper – http://www.nemethi.de/ a http://tkinter.unpythonic.net/wiki/TableListWrapper

## B.3 Instalace

Samotný program instalaci nevyžaduje, ale používá řadu knihoven, které je třeba nainstalovat.

1. nainstalujte knihovnu libpcap nebo Wincap
2. nainstalujte Tcl a Tk

3. nainstalujte Python 2.7 s podporou Tk/Tcl
4. přeložte libSVM
5. přeložte libSVM Python interface
6. nainstalujte pcapy pomocí skriptu `setup.py` (na Windows je třeba ještě stáhnout a rozbalit Developer's pack pro WinPcap (http://www.winpcap.org/devel.htm a rozbalit ho do adresáře `C:\devel\`)
7. nainstalujte doplněk Tcl TableList podle přiložených instrukcí
8. zkopírujte soubor `TableList.py` do adresáře `site-packages` v adresáři Pythonu

## B.4 Použití programu

Program lze spustit buď jako konzolovou aplikaci nebo s grafickým rozhraním. Pokud je program spuštěn bez parametrů, zapne se grafické uživatelské rozhraní a všechny potřebné volby se nastaví v něm.

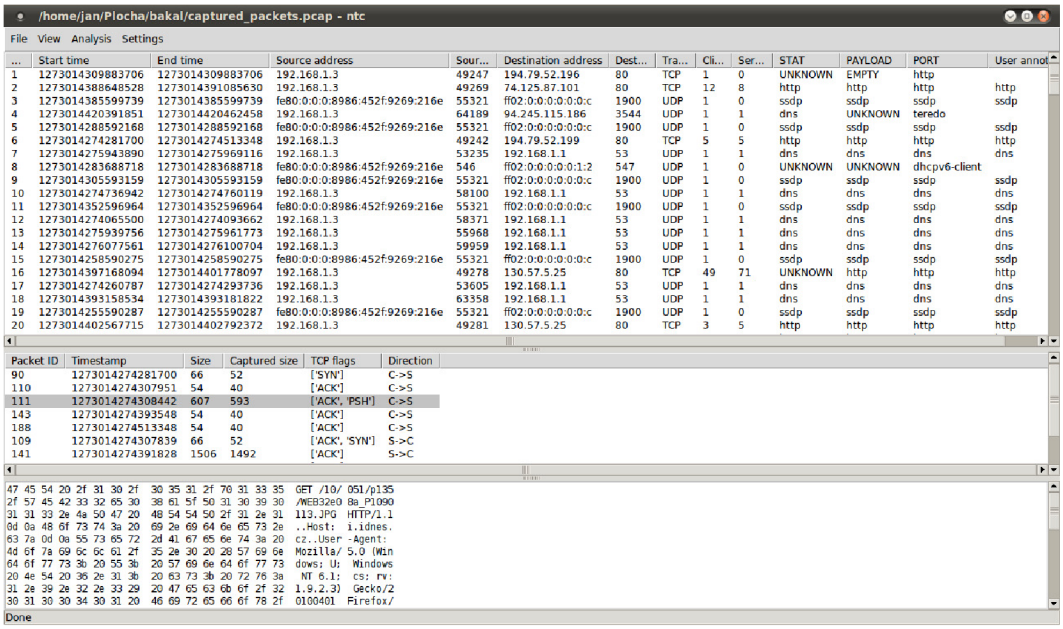
Pokud chcete spustit program v příkazové řádce, musíte jako parametr zadat alespoň jméno vstupního souboru. Všechny parametry pro spuštění v příkazové řádce jsou:

```
./ntat.py [volby] vstupní_soubor

volby:
 -h --help           zobrazí nápovědu
 -o --output=SOUBOR  zapíše výsledky do souboru SOUBOR
 -x --xml            vrátí výsledky ve formátu XML
 -s --statistics     vypíše stručné statistiky vstupního souboru
```

Vstupní soubor je očekáván ve formátu `pcap`. Pokud je místo vstupního nebo výstupního souboru zadáno `-`, je použit `<stdin>` nebo `<stdout>`.

Uživatelské rozhraní je velmi jednoduché – obsahuje hlavní nabídku a tři podokna pro zobrazení výsledků. V prvním jsou zobrazeny datové toky, ve druhém jednotlivé pakety ve vybraném toku a ve třetím je zobrazován payload zvoleného paketu.




Hlavní nabídka obsahuje následující submenu: `File`, `View`, `Analysis` a `Settings`. Podnabídka `File` umožňuje otevřít a zavřít vstupní soubor, uložit soubor s výsledky a ukončit aplikaci. Pomocí položky `View->Statistics` je možné zobrazit stručné statistiky dat ve vstupním souboru. Vlastní analýza je spuštěna pomocí `Analysis -> Run`. Volba `Settings -> Preferences` otevře okno s nastaveními.

Po dokončení analýzy jsou v prním podokně zobrazeny výsledky obsahující informace o datovém toku, výsledky vrácené jednotlivými analyzátory a pole pro výslednou anotaci. Pokud bylo u datového toku dosaženo nějakým analyzátorem prahové úspěšnosti, obsahuje toto pole celkový výsledek. Tento výsledek lze libovolně editovat.

## B.5 Nastavení

Aplikace používá řadu nastavení která jsou uložena v hlavním konfiguračním souboru. Ten obsahuje sekce `[Splitter]` pro nastavení rozdělování paketů do jednotlivých datových toků, `[Analyzers]` pro zapínání jednotlivých analyzátorů, `Printer` pro nastavení výpisu výsledků a samostanou sekci pro každý instalovaný analyzátor.

Jednotlivá nastavení:

`killinterval` počet paketů, po kterých se vyhledávají mrtvé toky podle timeru, 0 znamená nikdy nevyhledávat.

`tcptimeout` doba nečinnosti v mikrosekundách, po které je datový tok považován za skončený (timer pro TCP)

`udptimeout` význam podobný jako u `tcptimeout` (timer pro UDP)

`weight` váha přikládaná výsledku daného analyzátoru

`threshold` práh pro automatickou anotaci

`tcpmaxsize` počet bytů payloadu, které jsou prohledávány u TCP, 0 znamená prohledávání celého payloadu

`udpmaxsize` počet bytů payloadu, které jsou prohledávány u UDP, 0 znamená prohledávání celého payloadu

`probability` práh pravděpodobnosti pro zobrazení výsledku statistického analyzátoru

Hodnoty nastavení musí být literály jazyka Python pro typy Int, Float, String nebo Bool a mohou nabývat následujících hodnot.

| **jméno** | **typ** | **očekávané hodnoty** |
|---|---|---|
| `killinterval` | Integer | $\langle 0; \infty \rangle$ |
| `tcptimeout` | Integer | $\langle 0; \infty \rangle$ |
| `udptimeout` | Integer | $\langle 0; \infty \rangle$ |
| `weight` | Integer | $\langle 0; \infty \rangle$ |
| `threshold` | Integer | $\langle 0; \infty \rangle$ |
| `tcpmaxsize` | Integer | $\langle 0; \infty \rangle$ |
| `udpmaxsize` | Integer | $\langle 0; \infty \rangle$ |
| `probability` | Float | $\langle 0; 1 \rangle$ |

Hlavní nastavení je možné upravovat i za použití GUI (`Settings -> Preferences`).

Čísla portů použitá pro analýzu lze upravovat v souboru `ports.txt` a vzory pro analýzu payloadu je možné modifikovat v souboru `patterns.txt`. Natrénovaný klasifikátor je uložen v souboru `model` ve formátu libSVM. Pro tvorbu nového nebo úpravu stávajícího klasifikátoru lze použít nástroje obsažené v knihovně libSVM.

Příklad konfiguračního souboru je zde:

```
[Splitter]
killinterval = 1000
tcptimeout = 1000000
udptimeout = 1000000

[Analyzers]
payload = true
port = true
stat = true

[Payload]
tcpmaxsize = 2048
udpmaxsize = 4096
weight = 30

[Port]
weight = 30

[Stat]
threshold = 0.8
weight = 30

[Printer]
threshold = 60
```

## B.6 Tvorba nových analyzátorů

Aplikace NTAT poskytuje rozhraní pro tvorbu nových analyzátorů a jejich snadné zapojení. Nový modul se tvoří naděděním třídy `Analyzer` a překrytím její metody `Behavior`. Jméno vytvořené třídy je použito jako název analyzátoru a mělo by začínat velkým písmenem. Jméno modulu musí odpovídat jménu třídy a smí obsahovat pouze malá písmena. Modul musí být umístěn v adresáři `modules` a je třeba ho zaregistrovat do souboru `__init__.py`, který je umístěn tamtéž.

Pokud ke své funkci analyzátor potřebuje nějaká nastavení, musí být uvedena v třídní proměnné `required_settings`, která je reprezentována jako slovník, jehož klíče jsou názvy nastavení a hodnoty jsou datové typy těchto nastavení (int, float, str nebo bool). Každý analyzátor musí povinně obsahovat v této proměnné dvojici `"weight": "int"`. Aby byl analyzátor aktivován, musí být v konfiguračním souboru v sekci `[Analyzers]` hodnota se jménem shodným se jménem modulu, která je typu bool. Dále musí být v konfiguračním souboru obsažena sekce shodná se jménem třídy modulu, ve které jsou uložena všechna potřebná nastavení.

### B.6.1 Třída `Analyzer`

Třída `Analyzer` obsahuje následující metody:

`Initialize()`
: V této metodě je možné provést inicializaci analyzátoru. Nastavení analyzátoru načtená z konfiguračního souboru jsou dostupná v proměnné `self.settings`, která obsahuje seznam dvojic `(jméno_nastavení, hodnota)`. Na hodnotách je provedena typová kontrola, ale všechny jsou předány jako řetězec. Pro převod řetězce na bool lze použít metodu `toBool(str)`

`Behavior(flow)`
: Tato metoda je volána pro každý datový tok, který je jí předán v proměnné `flow`. Objekt `Flow`, který je v této proměnné, je podrobně okomentován ve zdrojovém souboru `mytypes.py` v adresáři `core`. Tato metoda musí vracet slovník obsahující jeden prvek, jehož klíč je jméno analyzátoru a hodnota je řetězec obsahující výsledek analýzy (jméno aplikačního protokolu) nebo `"EMPTY"`, pokud tok neobsahuje žádná data, nebo `"UNKNOWN"`, pokud nebyla analýza úspěšná.

`Finalize()`
: V této mětodě je možné provést finalizaci analyzátoru. To umožňuje implementovat víceprůchodový analyzátor, který během provádění metody `Behavior` nasbírá všechna potřebná data a v této metodě provede jejich analýzu v libovolném počtu průchodů. Výsledky je pak třeba zapsat do proměnné `self.results`, která obsahuje slovník objektů typu `Result`, jehož klíče jsou řetězce získané konkatenací five-tuple a počátečního času datového toku (`str(Flow.fivetuple) + " " + str(Flow.stime)`). Podrobnosti viz zdrojový soubor `mytypes.py`.

`Error(jméno, zpráva)`
: Tato metoda ukončí analýzu v případě chyby a vypíše chybovou hlášku s hlavičkou `jméno` a tělem zprávy `zpráva`.

Následuje příklad jednoduchého pseudo-analyzátoru, který si před zahájením analýzy načte nastavení, na jejich základě vrací výsledky a nakonec vypíše zprávu na `<stdout>`.

```
#! /usr/bin/env python2.7
# -*- coding: utf-8 -*-

from core.analyzer import Analyzer
from core.mytypes import Ftuple, Packet, Flow

class Example(Analyzer):
  #vyžadovaná nastavení
  required_settings = {'weight': 'int', 'set1': 'bool', 'set2': 'str'}

  def Initialize(self):
    #načtení nastavení
    for i~in self.settings:
      if (i[0] == 'set1'):
        #převod řetězce na bool
        try:
          self.set1 = self.toBool(i[1])
        except:
          self.Error("chyba:", "Někde se stala chyba!")
      elif (i[0] == 'set2'):
        self.set2 = i[1]
    #nějaká činnost

  def Behavior(self, flow):
    #analýza
    vysledek = self.set2
    if (self.set1):
      vysledek += "!!!"
    return({'EXAMPLE': 'výsledek' + " " + vysledek})

  def Finalize(self):
    print "Analýza dokončena"
```

Po odebrání modulu je třeba odstranit jeho záznamy z nastavení, jinak dojde k chybě.

# Příloha C

# Struktura výsledkového XML

Výsledkový soubor ve formátu XML má strukturu definovanou následujícím DTD:

```
<!ELEMENT results (flow_list, statistics?)>
<!ELEMENT flow_list (flow*)>
<!ELEMENT flow (start_time, end_time, five_tuple,
  client_packet_list, server_packet_list, result_list,
  user_result)>
<!ELEMENT start_time EMPTY>
  <!ATTLIST start_time seconds CDATA #REQUIRED
    microseconds CDATA #REQUIRED
    formated_string CDATA #IMPLIED>
<!ELEMENT end_time EMPTY>
  <!ATTLIST end_time seconds CDATA #REQUIRED
    microseconds CDATA #REQUIRED
    formated_string CDATA #IMPLIED>
<!ELEMENT five_tuple EMPTY>
  <!ATTLIST five_tuple source_address CDATA #REQUIRED
    destination_address CDATA #REQUIRED
    source_port CDATA #REQUIRED
    destination_port CDATA #REQUIRED
    transport_protocol CDATA #REQUIRED>
<!ELEMENT client_packet_list (packet*)>
<!ELEMENT server_packet_list (packet*)>
<!ELEMENT packet EMPTY>
  <!ATTLIST packet seconds CDATA #REQUIRED
    microseconds CDATA #REQUIRED
    formated_string CDATA #IMPLIED
    id CDATA #IMPLIED>
<!ELEMENT result_list (result+)>
<!ELEMENT result EMPTY>
  <!ATTLIST result analyzer CDATA #REQUIRED
    code CDATA #REQUIRED>
<!ELEMENT user_result EMPTY>
  <!ATTLIST user_result code CDATA #REQUIRED>
<!ELEMENT statistics (total_packets, total_flows,
```

```
  arp_packets, unsupported_packets, ipv4_packets,
  ipv6_packets, icmpv4_packets, icmpv6_packets,
  igmp_packets, tcp_packets, udp_packets)>
<!ELEMENT total_packets EMPTY>
  <!ATTLIST total_packets count CDATA #REQUIRED
    bytes CDATA #IMPLIED>
<!ELEMENT total_flows EMPTY>
  <!ATTLIST total_flows count CDATA #REQUIRED>
<!ELEMENT arp_packets EMPTY>
  <!ATTLIST arp_packets count CDATA #REQUIRED>
<!ELEMENT unsupported_packets EMPTY>
  <!ATTLIST unsupported_packets count CDATA #REQUIRED>
<!ELEMENT ipv4_packets EMPTY>
  <!ATTLIST ipv4_packets count CDATA #REQUIRED
    bytes CDATA #IMPLIED>
<!ELEMENT ipv6_packets EMPTY>
  <!ATTLIST ipv6_packets count CDATA #REQUIRED
    bytes CDATA #IMPLIED>
<!ELEMENT icmpv4_packets EMPTY>
  <!ATTLIST icmpv4_packets count CDATA #REQUIRED>
<!ELEMENT icmpv6_packets EMPTY>
  <!ATTLIST icmpv6_packets count CDATA #REQUIRED>
<!ELEMENT igmp_packets EMPTY>
  <!ATTLIST igmp_packets count CDATA #REQUIRED>
<!ELEMENT tcp_packets EMPTY>
  <!ATTLIST tcp_packets count CDATA #REQUIRED
    bytes CDATA #IMPLIED>
<!ELEMENT udp_packets EMPTY>
  <!ATTLIST udp_packets count CDATA #REQUIRED
    bytes CDATA #IMPLIED>
```