VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCE DYNAMICKÝCH SÍŤOVÝCH APLIKACÍ

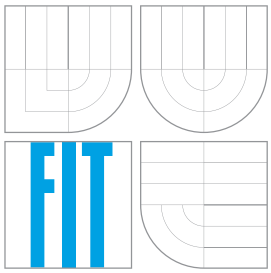
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR OROŠ

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

DETEKCE DYNAMICKÝCH SÍŤOVÝCH APLIKACÍ

DETECTION OF DYNAMIC NETWORK APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR OROŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ TOBOLA
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zabývá detekcí dynamických síťových aplikací. Popisuje metody obecné detekce libovolného protokolu aplikační vrstvy a automatické vyhledávání pravidel pro detekci. Praktická část se zabývá obecnou detekcí protokolů na základě automaticky generovaných vzorů. Zvláštní část práce je věnována detekci bittorent protokolu.

Abstract

This work deals with the detection of dynamic network applications. It describes a general method for detection of any application layer protocol and automatically searches for the detection rules. The practical part deals with general detection protocol to the automatically generated designs. A special section is devoted detect bittorrent protocol.

Klíčová slova

P2P, peer-to-peer, TCPDump, L7-filter, detekce, BitTorrent

Keywords

P2P, peer-to-peer, TCPDump, L7-filter, detection, BitTorrent

Citace

Petr Oroš: Detekce dynamických síťových aplikací, bakalářská práce, Brno, FIT VUT v Brně, 2011

Detekce dynamických síťových aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jiřího Toboly

.....
Petr Oroš
17. května 2011

Poděkování

Děkuji za vstřícné jednání a odborné vedení vedoucímu práce Ing. Jiřímu Tobolovi.

Obsah

Kapitola 1

Úvod

V rozmanitém prostředí moderních počítačových sítí mezi sebou v každé chvíli komunikují obrovské počty různých aplikací. Chceme-li takové komunikace uskutečňovat, musíme dodržovat jistá pravidla. Tato pravidla se nazývají protokoly. Protokolem je myšlen soubor pravidel, na jejichž základě si jsou schopny komunikující strany rozumět a pochopit obsah přijímaných a vysílaných dat.

Stejně jako se vyvíjí aplikace, procházejí řadou změn, vylepšení a rozšíření, tak se vyvíjí i komunikační protokoly které používají. Chceme-li vytvořit detekční systém pro identifikaci jednotlivých protokolů, musíme brát ohled i na tento vývoj a rozlišovat mezi sebou nejen různé protokoly ale i jejich konkrétní verze.

Vytvořit komplexní systém pro analýzu a detekci různorodých protokolů není jednoduché. Hlavní příčinou jsou nedostupné dokumentace spousty proprietárních protokolů a rychlý vývoj. Proprietární protokoly, využívané třeba jen jednou aplikací, se mohou měnit kdykoliv je to nutné. Toto rozhodnutí záleží pouze na vývojáři daného protokolu/aplikace. Pak jen stačí, aby vývojář změnil např. řetězec v hlavičce protokolu na jehož základě protokol detekujeme a detekce přestane fungovat.

Cílem této práce je navrhnout a implementovat takový detekční systém, který dokáže nejen rozlišit mezi sebou různé protokoly, ale dokáže i v co největším počtu případů automaticky určit o jaký protokol se jedná. Cílem je také do dopodrobna popsat principy dynamických protokolů jako je BitTorrent a implementovat prototyp aplikace, která dokáže tyto protokoly detekovat.

Práce je členěna do kapitol. Druhá kapitola čtenáře uvede do problematiky protokolů v počítačových sítích, nutných k bezproblémovému pochopení dalšího textu. Dále bude v kapitole probrána terminologie a principy P2P sítí, protokolu BitTorrent a Skype. Třetí kapitola bude zaměřena na principy obecné detekce libovolného protokolu. Zvláštní část třetí kapitoly bude věnována detekci protokolu BitTorrent. Ve čtvrté kapitole budou probrány schopnosti a vlastnosti v současnosti používaných nástrojů pro klasifikaci síťového provozu a analýzu obsahu paketů. Pátá kapitola bude věnována implementaci prototypu aplikace pro detekci dynamických síťových aplikací, možnostem jejího rozšíření a porovnání vlastností s L7-filterem.

Kapitola 2

Teoretický rozbor

2.1 Model ISO/OSI

Model ISO/OSI je vrstvený referenční model, vypracovaný organizací ISO v roce 1984 [1]. Cílem vzniku tohoto modelu bylo, vytvořit normy pro účely propojování systémů. V tomto modelu každý protokol patří do určité vrstvy. Protokoly vyšších vrstev jsou při přenosu dat po síti baleny do protokolů nižších vrstev a přitom na sobě nejsou jednotlivé vrstvy přímo závislé. Protokolům na vyšších vrstvách nezáleží na tom, jakým způsobem a jakým protokolem jsou přenášeny po síti, protože po přenosu bude protokol vyšší vrstvy opět „rozbalen“ a předán aplikaci tak, jak byl odeslán aplikací na druhé straně.

7.	Aplikační
6.	Prezenční
5.	Relační
4.	Transportní
3.	Síťová
2.	Linková
1.	Fyzická

Tabulka 2.1: Vrstvy modelu ISO/OSI

Vyvíjená aplikace bude detekovat protokoly poslední, tedy aplikační vrstvy. Důležité jsou i informace uložené v nižších vrstvách a to proto, abychom mohli identifikovat tzv. Flow. Flow je datový tok, který patří jednomu konkrétnímu spojení a veškeré pakety v něm jsou součástí jedné probíhající relace mezi aplikacemi. Dále v textu jsou často zmiňovány síťové vrstvy (l3, l4, ...), tím jsou myšleny právě vrstvy výše zmíněného modelu.

2.2 Protokol IP

Protokol IP je jedním ze základních protokolů používaným pro přenos dat po síti. Popis v RFC 791 pochází z r. 1981. Vrstva na které pracuje IP protokol, umožňuje adresování koncových uzlů na globální úrovni. IP nezaručuje bezpečný a spolehlivý přenos dat, slouží pouze pro doručení bloku dat od jednoho uzlu k druhému.[6]

Pro účely detekce protokolů aplikační vrstvy, je IP důležitý především pro základní určování flow a to podle IP adresy zdrojového a cílového koncového uzlu. Z hlavičky IP

paketu je pro určování síťového toku klíčová právě IP adresa, která jednoznačně určí 2 uzly, které mezi sebou komunikují, což je první nezbytný předpoklad pro rozhodnutí, zda jde o komunikaci jednoho konkrétního protokolu aplikační vrstvy.

Koncové uzly, a všechny ostatní aktivní síťové prvky v internetu potřebují pro svoji činnost jednoznačnou adresu, podle které mohou být identifikovány a adresovány. K tomu slouží tzv. IP adresa. IP adresa má pevně daný formát, určený verzí protokolu. Pro IPv4 má délku 32b. Používá se notace xxx.xxx.xxx.xxx, kde xxx odpovídá číslu od 0 do 255. Pro IPv6 má adresa délku 128b a notace vypadá takto xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx, kde xxxx odpovídá hexadecimální reprezentaci 16b.

2.3 Protokol TCP

Protokol TCP je protokol transportní vrstvy detailně popsaný v RFC 793. Jeho účelem je spolehlivé doručení dat mezi jednotlivými procesy. Hlavička tohoto protokolu již může obsahovat informace užitečné pro detekci konkrétního protokolu aplikační vrstvy. Hlavička vypadá následovně:[6]

Bits	**0-3**	**4-7**	**8-15**	**16-31**
0	Source Port			Destination Port
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Reserved	Flags	Window
128	Checksum			Urgent Pointer
160	Options (optional)			
160/192+	Payload			

Tabulka 2.2: Hlavička protokolu TCP

Zdrojový/Cílový port

Tato dvě 16b čísla slouží pro adresování na úrovni transportní vrstvy. Zdrojový port zvolí u navazovaných spojení náhodně operačním systém a je portem na klientské straně. Cílový port na straně serveru, je klíčový pro adresaci a připojení ke konkrétní službě. Cílový port nastavuje administrátor serveru.

Většina nejpoužívanějších služeb dnešního internetu používá tzv. Standardní čísla portů a velice často podle něj lze určit konkrétní službu. Toto bohužel není pravidlem a administrátoři porty, na kterých služby běží mění. Nejčastějším důvodem je bezpečnost. Pokud se potenciální útočník snaží na serveru najít jemu známou zranitelnou službu, bude ji s největší pravděpodobností hledat právě na standardním portu. Pro účely detekce protokolů aplikační vrstvy je tato skutečnost komplikací a nelze se na číslo cílového portu spoléhat.

Sekvenční číslo

Sekvenční číslo je klíčové pro vlastní flow. Jeho hodnota se s každým paketem zvyšuje o vlastní velikost. Podle sekvenčního čísla může přijímací strana naprosto přesně určit, jak jdou pakety po sobě a zda se nějaký neztratil a také zda dorazil v celku. Hodnota

sekvenčního čísla se při navázání spojení nastavuje z bezpečnostních důvodů náhodně. Náhodná hodnota umožňuje zabránit únosu spojení a útoku přehráváním na úrovni transportní vrstvy. Neexistuje totiž způsob, jak dopředu odhadnout hodnotu sekvenčního čísla.

Acknowledgment

Nastavuje jej především přijímající strana. Jedná se o potvrzení, že paket skutečně dorazil v pořádku, a že není třeba původní odeslaný paket posílat znovu. Ack obsahuje hodnotu sekvenčního čísla zvětšenou o délku posledního přijatého paketu.

Offset dat

Položka určující offset, kde začíná payload. Lze z ní odvodit délku hlavičky TCP.

Příznaky

V 8b poli příznaků má každý bit specifický význam. Důležité jsou především příznaky SYN, ACK, RST a FIN. Podle toho, který z příznaků je právě aktivní, můžeme zjistit o jaký paket se jedná. Pakety, které mají nastaven jeden z těchto příznaků jsou signalizační a v datové části obvykle nenesou žádné informace užitečné pro detekci protokolu aplikační vrstvy. Významy příznaků:

- SYN – první paket v komunikaci signalizující navazování spojení
- ACK – obsahuje informace potvrzující správné doručení datového paketu
- RST – signalizující reset spojení
- FIN – ukončení komunikace

Checksum

Kontrolní součet paketu. Pokud tato hodnota nesouhlasí s kontrolním součtem vypočítaném z těla paketu, signalizuje to chybu při přenosu. Jestli bude paket zahozen nebo dál zpracován a předán vyšším vrstvám, záleží pouze na konkrétní implementaci TCP v operačním systému. Pro účely detekce protokolu aplikační vrstvy je paket s chybným kontrolním součtem problém a je lepší ho ignorovat. Takový paket může obsahovat nesmyslná data, porušená při přenosu a nelze na něj brát zřetel.

Urgent pointer

Ukazatel signalizující urgentní data v payloadu paketu.

Payload

Payload obsahuje vlastní data protokolu aplikační vrstvy, který budeme detekovat.

2.3.1 Navázání spojení

Protokol TCP neumožňuje prosté zaslání paketu s daty v payloadu. TCP nejprve vyžaduje navázat spojení tzv. handshake neboli podáním ruky. U TCP se používá 3-way handshake. Podání ruky probíhá následovně:

- Klientská strana navazující spojení pošle packet SYN
- Serverová strana přijímající spojení pošle paket SYN/ACK
- Klientská strana odpoví paketem ACK

Podání ruky slouží k synchronizaci a nastavení položek nutných pro komunikaci protokolem TCP jako například sekvenční číslo. Jak již bylo řečeno, tyto pakety nenesou v datové části žádné informace, protože slouží k inicializaci komunikace.

2.3.2 Ukončení komunikace

Pro ukončení komunikace slouží v hlavičce TCP flag nazvaný FIN. Pokud chce kterákoliv strana ukončit spojení, pošle druhé komunikující straně paket s nastaveným flagem FIN. Druhá strana pouze odpoví paketem ACK a spojení se korektně ukončí.

2.4 Protokol UDP

Protokol UDP funguje stejně jako TCP na transportní vrstvě. Podrobné informace a jeho specifikace jsou popsány v RFC 768. Primárně je určen pro nespolehlivé doručování dat. To znamená že odesílací strana pošle paket a dále se nezajímá co se s ním děje. Pokud bychom chtěli použít UDP pro spolehlivé doručování, museli bychom toto implementovat na aplikační vrstvě, UDP pro to prostředky nemá. Hlavička UDP obsahuje pouze nezbytné položky a vypadá následovně:[6]

Bits	**0-15**	**16-31**
0	Source Port	Destination Port
32	Lenght	Checksum
64	Payload	

Tabulka 2.3: Hlavička protokolu UDP

Položky v hlavičce UDP mají stejný význam jako v případě TCP. UDP sice slouží pro negarantovaný přenos, ale i přes to musí existovat způsob, jak zjistit, zda paket nebyl při přenosu porušen a v případě, že ano tak se podle toho zachovat např. zahozením. Proto hlavička obsahuje položky checksum a lenght.

2.5 Protokoly aplikační vrstvy

Protokoly na této vrstvě jsou již vlastním prostředkem, pomocí kterého mezi sebou komunikují konkrétní aplikace. Vzhledem k tomu, že detekce těchto protokolů je jádrem celé práce, bude jim věnována celá kapitola.

2.6 Terminologie P2P a BitTorrentu

BitTorrent je open source protokol pro P2P sdílení souborů. Byl vytvořen Bramem Cohenem v roce 2002. Nyní je spravován a vyvíjen firmou BitTorrent Inc.[5] Základní myšlenkou BitTorrent protokolu je absence centrálního datového serveru. Veškerá data se sdílí mezi uživateli navzájem a také jsou uložena přímo u uživatelů. Centrální server sice existuje, ale slouží pouze pro navazování spojení a pro výměnu informací o aktuálně nabízených a stahovaných souborech.

Torrent

Malý soubor obsahující metadata souborů, která jsou sdílena. V metadatech jsou obsaženy informace jako názvy souborů, jejich velikosti a kontrolní součty.

Tracker

Centrální server udržující informace o klientech sdílejících nebo stahujících data. Tracker uchovává také informace o tom, které bloky dat má který uživatel k dispozici. Trackerem poskytované informace o dostupných blocích a adresách peerů se aktualizují periodicky (nejčastěji v intervalu 1h), nebo na vyžádání (kliknutí na položku v menu softwarového klienta).

Seeder

Klient připojený do sítě, který má k dispozici kompletní požadovaná data. Od těchto uživatelů mohou ostatní, na základě znalosti hashe, žádat o stažení. Po navázání komunikace mezi klientem požadujícím data a seederem dojde k výměně informací o datech která jsou k dispozici.

Leecher

Klient požadující data. Jedná se o klienta, který nemá kompletní data torrentu. Na základě informací získaných z trackeru se připojuje k seederům a od nich stahuje bloky dat které mu chybí. Leecher se připojuje i k ostatním leecherům a s nimi si vyměňuje další bloky, které sám nemá.

Swarm

Skupina klientů, kteří jsou seedery nebo leechery konkrétního torrentu.

Peer

Souhrnné označení pro seed a leech.

DHT

Distributed hash table. Technologie umožňující šíření informací o protějšcích bez použití trackeru. Tabulka obsahuje data o dostupných blocích a peerech tak, jak by to dělal tracker. Při použití této technologie je u každého peera jen část celého seznamu peerů, proto distribuovaná hash tabulka.

Trackerů existují v zásadě 2 druhy. Privátní a veřejné. U veřejných stačí uživateli navštívit stránku, na které vyhledá a stáhne požadovaný torrent soubor jako anonymní uživatel. U privátních trackerů má server ještě další funkci a to statistickou.

Uživatel, který požaduje dat,a se nejprve přihlásí do webové aplikace obsahující torrenty. Následně vyhledá požadovaný obsah, a stáhne torrent soubor. Při stahování (výměně dat mezi uživateli) se ovšem na trackeu logují data o množství stažených a odeslaných dat jednotlivých uživatelů. To je pak využíváno pro vedení statistik a pro řízení přístupu např. k nejnovějším torrentům podle toho, který uživatel jak přispívá k chodu sítě tím, že odesílá větší množství dat než stáhne. Poměr stažených a odeslaných dat se nazývá ratio.

2.7 Princip P2P

Princip P2P je založen na decentralizované síti. Stahovaný/sdílený obsah klientská aplikace rozdělí na bloky pevné velikosti (obvykle 1-4 MB). Každý takový blok má vlastní hash. Bloky jsou pak sdíleny mezi jednotlivými klienty, bez nutnosti stahovat data z centrálního serveru.[2]

V dnešních dobách je P2P sdílení využíváno převážně pro nelegální šíření audiovizuálních děl, aplikací a také masově rozšířených bezplatných aplikací, které by jinak stahoval velký počet zájemců z centrálního ftp serveru. P2P umožňuje takový obsah distribuovat s téměř nulovými náklady, na rozdíl od centrálního ftp serveru, který musel mít velký výkon a šířku přenosového pásma.

Ke sdílení informací o protějšcích používají P2P sítě 2 základní mechanismy. Prvním je uložení adres a informací o sdílených blocích na serveru (trackeru). Každý stahující/sdílející klient se připojí k trackeru a podle hash_id konkrétního torentu požádá o seznam peerů. Druhým mechanismem je uložení informací o protějšcích pomocí DHT.

DHT plní funkci trackeru. Databáze obsahující seznam připojených uzlů a dostupných bloků je však rozdistribuována mezi protějšky. Po stažení torrent souboru, který obsahuje seznam peerů, se klient začne k těmto protějškům připojovat a pomocí DHT sdílet seznam dostupných bloků. U Privátních trackerů bývá tento mechanismus obvykle zakázán, protože neexistuje způsob jak takto přenesená data a připojené klienty logovat a počítat množství přenesených dat k výpočtu ratia.

2.7.1 Vytvoření torrentu

Pro vytvoření torrent souboru je nutný softwarový klient (uTorrent, Ktorrent, Vuze, Azureus, atp.). V grafickém rozhraní klienta vybereme v menu volbu „Nový torrent“. Vybereme soubory, které budou jeho součástí a způsob šíření (tracker nebo DHT). V případě zvolení trackeru, musíme zadat url announce a zda chceme i v tomto případě povolit DHT. Při volbě DHT si klient vyžádá zadání adresy alespoň jednoho peera. V případě vytváření nového torrentu to bude adresa klienta, který bude data sdílet jako první. Adresa musí být veřejná aby bylo možné připojení dalších peerů. Tuto adresu získají klienti právě z torrent souboru. Dalším krokem je nahrání torrent souboru na nějaký web server, kde si jej budou moci stáhnout další uživatelé. Pokud byl zvolen způsob šíření peerů přes tracker, tak se soubor nejčastěji nahrává na web, který je přiřazen přímo konkrétnímu trackeru.

2.7.2 Stažení torrentu

Pro stažení torrentu jej stačí vyhledat na webové stránce trackeru nebo vyhledávače torrentů a otevřít v torrent klientu. Po otevření se klient připojí buď k trackeru a stáhne seznam peerů nebo se pokusí z DHT získat seznam peerů z adresy uvedené v torrent souboru. Po získání adres peerů se k nim začne připojovat a vyměňovat si jednotlivé bloky. Takto navázaných spojení mohou být desítky až stovky, což značným způsobem zatěžuje síť.

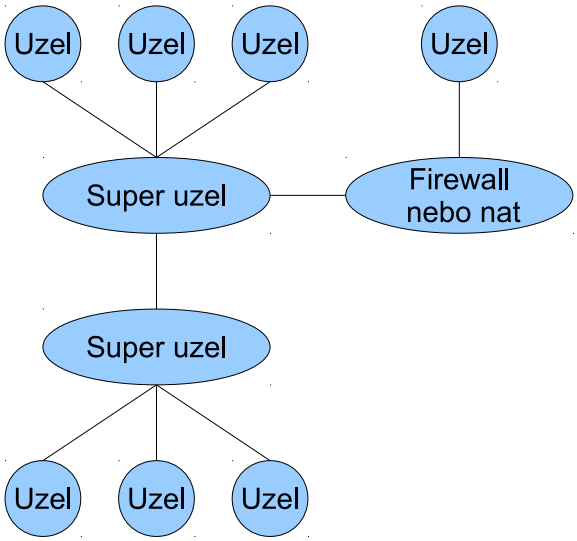
2.7.3 Active/pasive mode

Komunikující peeři mohou být v jednom ze 2 módů. Buď v aktivním, kdy je klient dostupný zvenčí a může přijímat požadavky na příchozí připojení (má veřejnou IP adresu a povolen port na kterém naslouchá klient ve firewallu), nebo v pasivním módu ve kterém je schopný spojení pouze navazovat. Aktivní klienti mohou stahovat a sdílet s kýmkoliv, protože pasivní klienti se k nim sami připojují. Pasivní klienti mohou stahovat a sdílet pouze s aktivními. Pasivní klient nemůže sdílet data s pasivním, protože nemá jak navázat spojení.

2.8 Skype

Skype je aplikace pro internetovou telefonii. Umožňuje přenos jak hlasu, tak videa a textových zpráv mezi dvěma a více koncovými uzly. Skype umožňuje uskutečňovat i hovory do veřejné telefonní sítě díky protokolu SkypeOut a přijímat hovory z veřejné telefonní sítě díky SkypeIn.[3]

Protokol, který Skype využívá je proprietární a proto je komplikovaná nejen detekce ale i analýza toho, jak Skype vlastně pracuje. Skype ná nespornou výhodu v tom, že díky principu, na kterém funguje nemá problémy s naty a firewally. Architektura sítě je následující:



Obrázek 2.1: Architektura sítě Skype

Uzel

Koncový uživatel, může být za natem nebo firewallem.

Super uzel

V podstatě se také jedná o běžného uživatele, který si nainstaluje klientský software stejně jako jakýkoliv jiný koncový uzel. Rozdíl je v tom, že Skype po spuštění zkontroluje, zda se uživatel nenachází za firewallem nebo natem, zda má dostatečně rychlé připojení a dostatek systémových prostředků. Pokud ano, určí ho jako super uzel a jsou přes něj směrovány hovory uživatelů, které by jinak nebylo možné přímo adresovat. S touto skutečností uživatelé souhlasí při instalaci Skype, protože toto je uvedeno v licenčních podmínkách.

Princip super uzlů se podobá P2P sdílení, protože Skype nemá žádný centrální server, přes který by byly směrovány všechny hovory. Jediný centrální server využívaný Skype slouží pro přihlašování uživatelů do sítě na základě Skype jména a hesla.

Každý klient si spravuje vlastní tabulku super uzlů. První záznamy jsou do tabulky zapsány již při instalaci (jsou přidány do instalátoru). Kdyby byla tabulka prázdná, musel by první záznamy Skype získat jinak. Pokud by totiž došlo k tomu, že nebude dostupný ani jeden super uzel, aplikace se sice přihlásí na login server ale nebude schopná uskutečnit žádné spojení. Napadením mechanismu super uzlů a výměny informací o super uzlech by bylo teoreticky možné, vyřadit celou síť z provozu.

Skype není jeden samostatný protokol ale využívá několik následujících mechanismů:

- SkypeOut – volání Skype → veřejná telefonní síť
- SkypeIn – volání veřejná telefonní síť → Skype
- Signalizace – komunikace mezi uzly a superuzly
- FileTransfer – pro přenos souborů
- IM – pro zasílání zpráv mezi uživateli

Skype veškerou komunikaci šifruje mechanismem end-to-end za pomocí AES. Toto je velice důležité, protože pakety prochází nejen internetem ale i přes super uzly, kterými jsou běžní uživatelé. Bez šifrování by nebylo možné uskutečňovat hovory takto nedůvěryhodnou infrastrukturou.

Kapitola 3

Současné technologie

3.1 LibPCAP

Knihovna funkcí, poskytujících rozhraní pro nízkoúrovňové zachytávání paketů ze síťové karty, čtení paketů ze souboru a jejich zápis do souboru. První verze knihovny byly vyvinuty v Berklyho laboratořích týmem, který vytvořil i Tcpdump. V současné době se týmy spojily a LibPCAP vyvíjí tým odpovědný za Tcpdump.

Knihovna je poměrně rozšířená a oblíbená pro jednoduché rozhraní a také proto, že se na knihovnu vztahuje licence open source. Je nedílnou součástí aplikací jako Wireshark, Tcpdump nebo Microsoft Network monitor 3.x. V současnosti existuje port pro operační systémy Windows, distribuovaný pod stejnou licencí jako LibPCAP. Verze pro Windows je nazvána WinPcap.[7]

Pro použití stačí knihovnu nainstalovat do systému. Některé aplikace pro Windows, mají přibalen instalátor WinPCap. V unixových systémech bývá LibPCAP jako závislý balík. Pokud chceme vyvíjet aplikace, schopné pracovat s LibPCAP, musíme nainstalovat i balík LibPCAP-dev, který obsahuje hlavičkové soubory s rozhraním knihovny.[8]

3.2 Tcpdump

Terminálová aplikace, schopná vypisovat pakety zachycené pomocí LibPCAP. Tcpdump má desítky různých parametrů, kterými lze ovlivnit jeho chování a formát výpisu. Všechny je možné nalézt v oficiální dokumentaci na http://www.tcpdump.org

Tcpdump není pouhé rozhraní napsané pro LibPCAP, dokáže toho mnohem více. Je schopný komunikaci filtrovat podle adresy nebo hostname kterékoliv z komunikujících stran. Dále umožňuje komunikaci filtrovat podle portu služeb na l4 vrstvě nebo podle protokolu l3 vrstvy. Toto jsou velice užitečné funkce, pokud nám jde o zachycení konkrétních paketů a ne o monitorování kompletního síťového provozu. Tcpdump bude v práci použit jako zdroj dat.[9]

3.3 L7-filter

Aplikační paketový klasifikátor, určený pro unixové operační systémy a to především pro linux. Na linuxu závisí hlavně proto, že pracuje jako nadstavba iptables. L7-filter existuje ve 2 variantách kernel a userspace. L7-filter kernel.[4]

Kernel varianta vyžaduje pro instalaci a běh následující:

- Jádro 2.4 nebo 2.6 vč. Zdrojových kódů
- iptables včetně zdrojových kódů

userspace varianta vyžaduje následující:

- nainstalované iptables

Kernel varianta není v současné době dále vyvíjena. Userspace varianta se sice dále vyvíjí ale v současné době ji nelze zkompilovat pod novějšími jádry. Staršího data (r. 2009) jsou i definice protokolů distribuované s L7-filterem.

Detekce protokolů probíhá na základě porovnání předpřipravených pravidel ve formě regulárních výrazů, uložených v souborech s příponou .pat. Výrazy jsou vždy definovány v oddělených souborech a každý soubor má jméno podle názvu protokolu. Vzorový výraz ze souboru smtp.pat:

$$\hat{}220[\backslash x09 - \backslash x0d\ -\ \tilde{}] *\ (e?smtp|simple\ mail)$$

L7-filter nekontroluje z důvodu efektivity všechny pakety. Kontroluje pouze několik prvních a když se detekce nezdaří, označí protokol jako unknown. V definicích lze často nalézt pravidla reprezentující první paket v komunikaci.

Chceme-li detekovat nový protokol, stačí vytvořit soubor s názvem protokolu a příponou .pat a do něj vložit výraz popisující protokol. Pokud nový protokol nelze detekovat na základě regulárního výrazu nebo řetězce v paketu, L7-filter si s ním neporadí.

3.4 Wireshark

Pokročilý analyzátor síťového provozu, vybavený grafickým uživatelským rozhraním a bohatou knihovnou podporovaných protokolů. Pakety zachytává pomocí knihovny LibPCAP popsané výše. Dokáže jednotlivé pakety nejen zachytit a zobrazit ale u jemu známých protokolů, dokáže dekódovat jednotlivé položky v hlavičkách.

Konkrétní protokol aplikační vrstvy určuje dle portu na l4 vrstvě. Pokud administrátor serveru port změní, wireshark protokol určí špatně, selže i dekódování a ve výpisu se zobrazí protokol s l4 vrstvy s nedekódovaným payloadem. Tato vlastnost je zřejmě dána zaměřením wiresharku na analýzu provozu, vizualizaci paketů a dekódování provozu.

Wireshark není primárně určen pro klasifikaci provozu. Wireshark se stal za dobu své existence neocenitelným nástrojem nejen aplikačních programátorů pro hledání chyb ve svých aplikacích ale i síťových administrátorů pro analýzu problémů se sítí.

Při psaní této práce byl použit k prostudování obsahu paketů jednotlivých flow a pro hloubkovou analýzu chování BitTorrent protokolu.

Kapitola 4

Principy detekce aplikací

4.1 Protokoly aplikační vrstvy

Vzhledem k tomu, že nikde není řečeno jak by měl protokol vypadat a vývojáři si jej mohou implementovat podle svého uvážení, není jednoduché na první pohled určit o jaký protokol se jedná. Pro tyto účely je vhodné protokoly rozdělit do tříd, podle základních vlastností a tím usnadnit návrh detekce.

4.2 Rozdělení protokolů

Pro účely detekce budou v této práci protokoly rozděleny podle několika kritérií.
Podle typu obsahu:

- Binární – Obsahem paketu jsou binární data. Paket může nést prakticky cokoliv.
- Textové – Obsahem paketu jsou tisknutelné znaky. Do této třídy budou patřit i protokoly, které nejsou ryze textové ale i ty, které obsahují většinový podíl čitelného textu.

Podle struktury:

- S hlavičkou – Každý paket začíná jednotnou hlavičkou.
- Bez hlavičky – Paket začíná přímo daty nebo je hlavička nejednotná.

Podle způsobu komunikace:

- Jednokanálové – Protokol používá pro komunikaci pouze jedno spojení.
- Vícekanálové – Protokol používá více spojení (obvykle 2).

Mechanismus vícekanálových protokolů používá např. protokol FTP, kde klient naváže řídící spojení na portu 21. Pokud se začnou přenášet data, musí být sestaveno další spojení na jiném portu a tímto nově vytvořeným kanálem jsou přeneseny žádané soubory. První kanál je po celou dobu používán na řízení přenosu a zasílání stavových zpráv. Toto rozdělení není jednoznačné, vlastnosti se mezi sebou často prolínají a u některých protokolů není na první pohled jasné, do které třídy patří.

4.3 Metody detekce

Vzhledem ke klasifikaci, kterou jsme provedli v předchozí kapitole, můžeme navrhnout metody pro jednotlivé typy protokolů a tím je detekovat co nejrychleji, nejefektivněji a s co možná nejmenšími nároky na zdroje (čas CPU a paměť). Rozdělení podle typu obsahu (binární, textové), je z pohledu detekce nepodstatné, proto se navrhované metody vztahují pouze ke klasifikaci podle struktury a podle způsobu komunikace. Pro vlastní detekci používá většina detektorů pravidla. Formát pravidel je dán použitým detektorem a může vypadat následovně:

- Konstantní řetězec vyskytující se ve všech paketech
  hlavička, patička, náhodný jev při šifrování
- Konstantní řetězec vyskytující se v jednom z paketů
  příkaz používaný u FTP, SMTP, řetězec při navazování spojení
- Regulární výraz nebo jiný prostředek pro popis obecného formátu řetězce
  regulární výrazy používá právě L7-filter

Pravidla se v zásadě nemusejí týkat všech paketů ale mohou se vztahovat např. k 1. paketu v komunikaci. Takový způsob sice vyžaduje zachycení třeba i jediného paketu ale vzhledem k tomu, že klasifikátory síťového provozu jsou určeny k monitorování síťového provozu, lze předpokládat nepřetržitou činnost a tím minimalizaci problémů při neodchycení důležitého paketu detektorem.

4.4 Protokoly s hlavičkou

Pro detekci protokolů s hlavičkou můžeme využít právě hlavičky. Metoda je založena na obsahu paketů, které budou začínat podobným, ne-li naprosto stejným řetězcem. Některé protokoly se dokonce do hlavičky podepisují aby bylo možné komunikující aplikací určit, zda se opravdu jedná o protokol, kterému ona sama rozumí nebo jen kvůli rozlišení verzí (např. smtp/esmtp).

Pokud se nám podaří určit, že se jedná o hlavičku a navíc nalezneme „podpis“ aplikace nebo jméno a verzi protokolu je detekce snadná, velice rychlá a efektivní. Často stačí přečíst prvních pár bajtů na začátku protokolu (např. http).

Automatické určení názvu protokolu je možné jen v případě, že v paketu je textově jeho název. Pokusíme-li se vyhledat ve všech paketech podřetězec, který se nachází v seznamu známých protokolů (dostupný na http://www.iana.org/assignments/port-numbers), můžeme přesně určit o jaký protokol se jedná. Toto je jediný případ, kdy lze automaticky určit i název protokolu. Další možností by bylo inteligentní prohledávání webu a pokus o nalezení definice protokolu. Tato technika je v současnosti nepoužitelná, protože neexistuje žádná centrální databáze nebo přesný formát, jak by měla tato data vypadat.

4.5 Protokoly bez hlavičky

Detekce těchto protokolů je mnohem komplikovanější právě kvůli absenci hlavičky. Jako nejlepší způsob se jeví nalezení podobností v paketech, charakteristické pro konkrétní protokol. Takovou vlastností můžou být délky paketů, konstantní řetězec uvnitř paketu nebo užití zápatí.

Hlavním problémem detekce takovýchto protokolů bývá efektivita. Pravidla. popisující protokol. jsou často velmi komplikovaná a při použití komplikovaných regulárních výrazů na jeden paket, který nutně nemusí být zasílaný jako první, může být detekce velice pomalá a extrémně náročná na procesorový čas.

Při detekci budou do kategorie protokolů bez hlavičky patřit nejen protokoly které nemají hlavičku, ale i ty co ji mají ale z nějakého důvodu nelze přesně určit. zda se jedná o hlavičku nebo datovou část paketu (např vlivem šifrování, nebo různých mechanismů komunikace, způsobujících nejednotnost hlavičky).

4.6 Vícekanálové protokoly

Na vícekanálové protokoly můžeme pohlížet těmito způsoby:

- Každý kanál brát jako samostatný protokol
- Všechny kanály brát jako jeden protokol

Výše zmíněné pohledy se projeví pouze na výstupu detektoru. Pro typického zástupce vícekanálového protokolu FTP, by mohl detektor označit řídící spojení jako FTP a datové jako FTP-DATA nebo oba pouze jako FTP. První přístup má velkou výhodu v možnosti využití řídících spojení k detekci spojení datových. Každé datové spojení, ať už jedno nebo více, musí být navázáno a k tomuto účelu slouží právě řídící spojení. Metodu zachycení řídícího spojení a získání informací o datových nelze použít při automatické analýze, protože nelze jednoduše automatizovat analýzu a z paketů určit, které informace popisují datová spojení.

4.7 BitTorrent

Cílem této práce je detekce dynamických síťových aplikací. Reprezentanty těchto aplikací jsou především P2P protokoly a to pro způsob jakým fungují. Aplikace podporující bittorrent používají 2 datové kanály:

- Pro komunikaci s trackerem
- Pro přenos dat a komunikaci s ostatními klienty

Kanál mezi klientem a trackerem není řešen jako součást BitTorrent protokolu ale využívá protokol http. Příklad komunikace mezi trackerem a klientem:
$GET\ /announce?parametry\ HTTP/1.1$
$Host:\ torrent.ubuntu.com:6969$
$Connection:\ Keep-Alive$
$User-Agent:\ KTorrent/4.0.3$

Odpověd:
$HTTP/1.0\ 200\ OK$
$Content-Length:\ 363$
$Content-Type:\ text/plain$
$Pragma:\ no-cache$
$d8:completei4950e10:incompletei480e8:intervali1800e5:peers300:peerlist$

Požadavek GET zasílá BitTorrent klient s těmito parametry:

- peer_id – Jednoznačná identifikace klienta
- port – Číslo portu na kterém klient naslouchá
- uploaded – Odeslaná data od poslední aktualizace
- downloaded – Stažená data od poslední aktualizace
- left – Objem dat, který zbývá ke stažení
- event – Událost, může obsahovat started, stoped, atd.
- info_hash – Hash torrentu

Každý klient připojením do sítě začne fungovat jako komunikační uzel. Aby bylo možné takovou síť provozovat, musí o sobě klienti navzájem vědět a znát adresy a porty, na kterých naslouchají další uzly.

Po stažení torrent souboru a jeho otevření v klientské aplikaci se klient připojí k trackeru od kterého tato data získá a zároveň trackeru sdělí na kterém portu naslouchá. Po předání těchto informací již není, pro sdílení dat, účast trackeru nutná.

Současné metody detekce jsou založeny na identifikaci tzv. BitTorrent handshake. BitTorrent handshake je paket, který posílá jeden klient druhému ještě před započetím vlastního přenosu dat. Vzor pro detekci používaný L7-filterem:

$$\hat{}(\backslash x13bittorrent\ protocol|azver\backslash x01\$|get\ /scrape\backslash ?info\_hash=$$
$$get\ /announce\backslash ?info\_hash=|get\ /client/bitcomet/|GET\ /data\backslash ?fid=)$$
$$|d1:ad2:id20:|\backslash x08'7P\backslash)[RP]$$

Pro bezproblémovou detekci stačí hledat řetězec „$\backslash x13bittorrent\ protocol$“ kde „$\backslash x13$“ reprezentuje délku řetězce „bittorrent protocol“. Tato metoda je dostatečně rychlá a dokáže detekovat všechna navazovaná spojení, která nejsou šifrována.

4.7.1 Šifrování

Chce-li klient šifrovat spojení z důvodu zabránění detekce, může šifrovat komunikaci s trackerem nebo vlastní přenosy dat. Šifrování komunikace s trackerem nelze explicitně vynutit. Šifrování musí podporovat tracker a to tak že http server (rozhraní trackeru) naslouchá na portu 443 a podporuje zabezpečenou verzi http protokolu pojmenovanou https. Šifrování přenosů dat mezi uzly lze povolit v klientské aplikaci. Při zapnutém šifrování je možné povolit i nešifrovaná spojení.

4.7.2 Detekce šifrovaných spojení

Jsou-li šifrována pouze datová spojení, lze jednoduchým způsobem detekovat veškeré klienty. Odchycením a analýzou http požadavku na registraci k trackeru klient prozradí port, na kterém naslouchá. Adresu klienta pak lze získat z hlavičky protokolu TCP. Taková informace stačí pro detekci všech příchozích připojení. Pokud ji uložíme, nemusíme pro nová připojení kontrolovat obsah paketů a detekce se mnohonásobně urychlí.

Pro řízení provozu, na základě použitého protokolu, je pro ISP takové chování neocenitelné. Při využití šifrovaných spojení i pro komunikaci s trackerem detekce selže. Způsobem

jak detekovat kompletně šifrovaná spojení (vč. komunikace s trackerem), může být sledování chování koncových uzlů v síti. Ani nejlepší šifrovací algoritmus totiž nedokáže zamaskovat princip, na kterém BitTorrent a další jemu podobné P2P sítě pracují. V síti lze rozlišit 3 druhy koncových uzlů:

- Klient
- Server
- P2P uzel

Klient

Klienty lze identifikovat jako body v síti, které občas sestaví spojení na různých portech s vysokým číslem. Operační systém volí porty pro odchozí spojení náhodně ale pouze z rozsahu portů vyšších než 1024. Porty s číslem nižším než 1024, jsou vyhrazeny konkrétním službám a neslouží pro odchozí spojení. Klienti ale, na rozdíl od serverů, nebudou naslouchat na portech nižších než 1024 nebo naslouchat budou ale pouze v rámci vnitřní sítě.

Server

Servery lze identifikovat podle otevřených portů s číslem nižším než 1024. Takto otevřený port bude dostupný i z vnější sítě. Server navazuje odchozí spojení jen velice zřídka nebo vůbec.

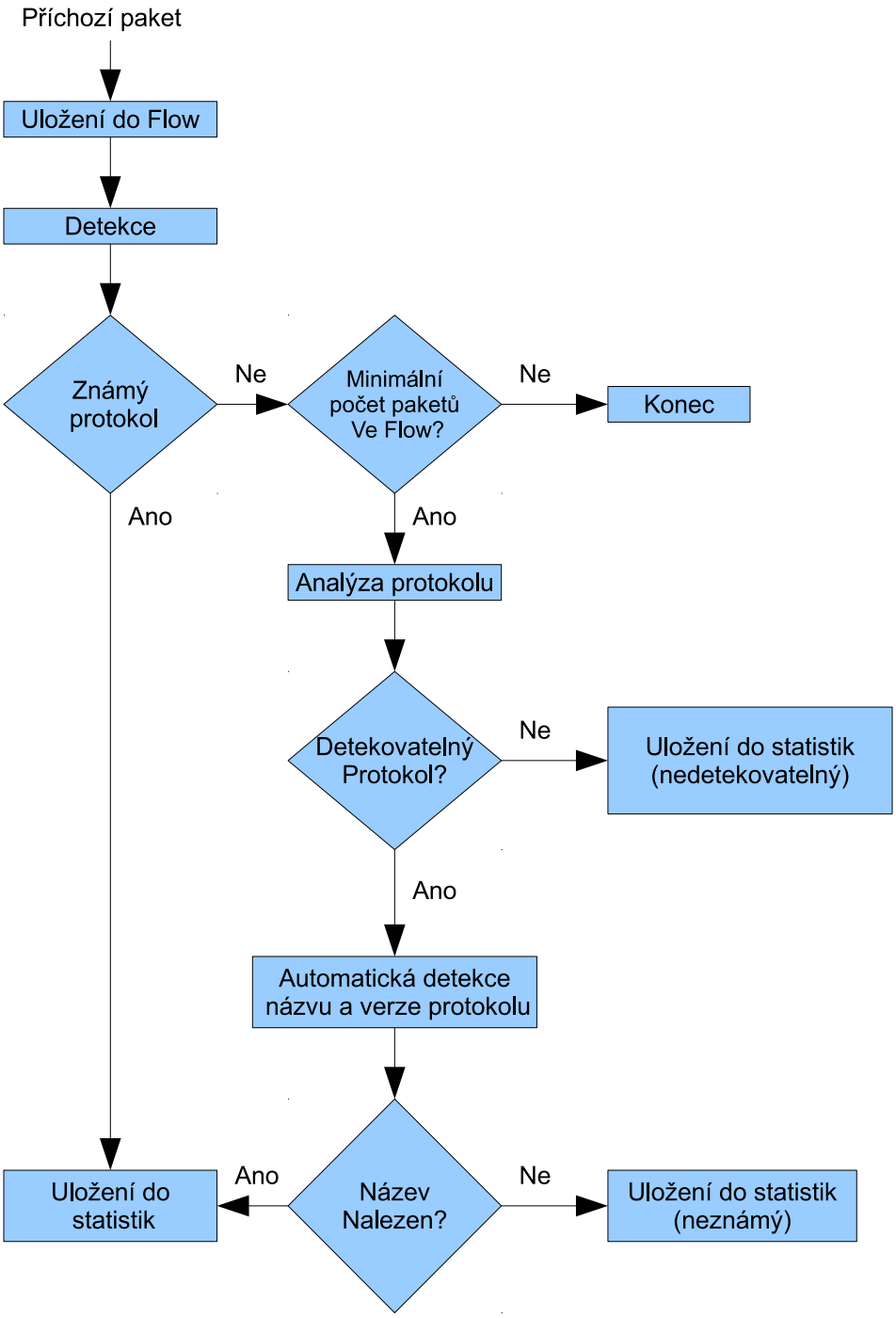
P2P Uzel

P2P uzel se bude oproti klientům a serverů chovat naprosto odlišně. Bude naslouchat na jednom portu (pro příchozí připojení od ostatních uzlů). A sám bude navazovat velké množství odchozích spojení. P2P komunikace je specifická také v délce jednotlivých spojení, která mohou být velice krátká. Často také dochází k situacím, kdy se připojí jeden uzel k druhému a nejsou přenesena žádná data, protože klient přijímající spojení nemá volný slot pro další uzel. Takto navázaná spojení obvykle nějakou dobu trvají a po určité době se ukončí.

Výše popsaný přístup má zásadní nevýhodu, nedokáže rozlišit jednotlivá spojení navázaná pomocí BitTorrent protokolu od běžných šifrovaných spojení. Tato technika tedy umožňuje pouze detekci koncových uzlů P2P sítě ale nedokáže určit, které relace jsou skutečně P2P provozem.

4.8 Automatická analýza protokolu

Automatickou analýzou můžeme dosáhnout detekce nových pro detektor neznámých protokolů, ne vždy je ovšem možné správně určit jméno a verzi protokolu. Příchozí pakety jsou předávány detektoru. Pokud detektor není schopný určit o jaký protokol se jedná, předá data k analýze. Analyzátor se pokusí nalézt v paketech vzor, použitelný jako pravidlo pro detektor. Nenalezne-li ho, označí protokol jako nedetekovatelný, flow je zapsána do statistik a dále ignorována. Když dokáže analyzátor najít použitelný vzor, pokusí se určit i jméno protokolu. Úspěch vede k zapsání do statistik pod skutečným jménem protokolu, neúspěch k zapsání jako detekovatelný ale neznámý. Algoritmus počítá i s možností nedetekovatelných protokolů. Protokol sice detekovatelný být může, ale automatická analýza nemusí nalézt způsob účinné detekce.



Obrázek 4.1: Algoritmus automatické analýzy protokolu

Úspěšnost principu automatické analýzy závisí na kvalitě analyzátoru. Pokud analyzátor nedokáže najít použitelný vzor pro detektor, bude selhávat i detekce a přínos je nulový. V případě špatně implementovaného analyzátoru, může být proces analýzy a detekce velice pomalý a neefektivní.

Kapitola 5

Implementace prototypu

5.1 Návrh systému pro detekci aplikací

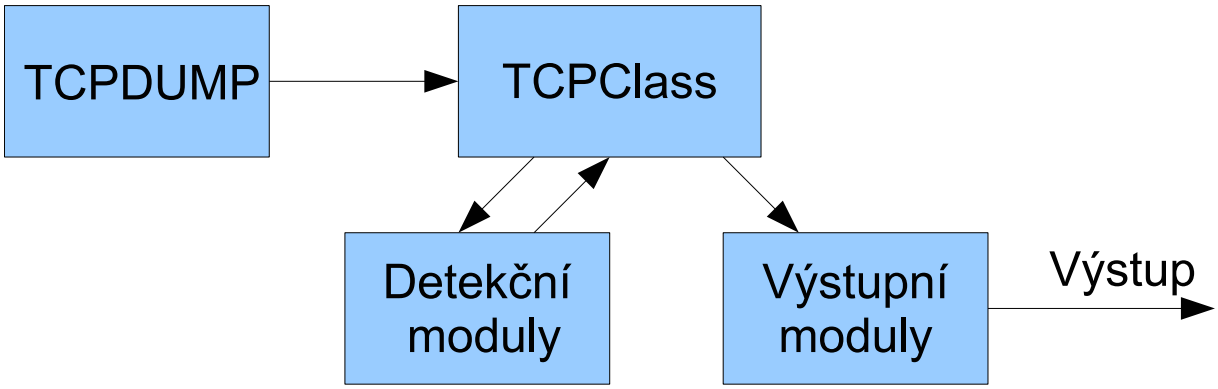
Pro implementaci systému (dále jen TcpClass) byl zvolen programovací jazyk C++ a to především pro jeho rychlost, pružnost a možnost efektivně nakládat s pamětí. Primární cílovou platformou je Linux ale vzhledem k tomu, že jedinou závislostí je Tcpdump, neměl by být problém TcpClass přenést na libovolnou jinou platfomu.

TcpClass byl navržen jako terminálová aplikace, schopná zpracovávat výstup s Tcpdumpu. Parsování Tcpdumpu a ukládání paketů do připravených struktur je jediná vlastnost samotné aplikace. Metody detekce a výstupy jsou implementovány jako moduly (sdílené, dynamicky načítané knihovny) a to především pro snadnou rozšiřitelnost o libovolnou novou metodu detekce nebo výstupu. Základní detekční moduly, implementované jako součást této práce, jsou popsány v kapitolách 5.4.1 a 5.4.2. Moduly pro výstup popisují kapitoly 5.4.3 a 5.4.4.

TcpClass je určen k nasazení u ISP pro detekci používaných protokolů a následné řízení kvality služeb tak, aby jeden druh provozu nezahlcoval aktivní síťové prvky a neblokoval tak běžný provoz ostatních účastníků sítě. Za ideálních podmínek (dostatečné množství připojených peerů) může aktivní bittorent klient navázat desítky až stovky krátkou dobu trvajících spojení. Výměna dat pak probíhá velmi rychle. Velký počet spojení na druhou stranu zaplňuje překladové tabulky na směrovačích a zahlcuje dostupné přenosové pásmo. Dalším možným využitím je přehled administrátorů o dění na síti nebo vedení statistik využití přenosového pásma. Na základě těchnto údajů může poskytovatel připojení např. změnit tarif tak, aby více vyhovoval požadavkům zákazníka. Jako nejvhodnější umístění serveru s TcpClass se jeví hraniční aktivní síťové prvky, přes které prochází veškerý provoz z vnitřní sítě do internetu.

5.2 Architektura

Techniky pro analýzu provozu, detekci jednotlivých protokolů a výstupy jsou implementovány jako moduly (knihovny s předdefinovaným rozhraním). Jednotlivé moduly lze aktivovat pouhým nakopírováním knihovny do adresáře /modules/detect/ (pro moduly na detekci a analýzu) nebo /modules/output/ (výstupní moduly). Tím zajistíme rozšiřitelnost o libovolnou detekční a analytickou metodu nebo libovolný formát výstupu. TcpClass očekává na svém vstupu výstup Tcpdumpu, spuštěného s těmito parametry: Tcpdump -i any -x -vvv -nn -S tcp



Obrázek 5.1: Architektura TcpClass

Vysvětlení parametrů

-i any: Parametr není nutné uvádět, nastavuje název síťového zařízení, které Tcpdump použije pro sběr dat. Bez uvedení toho parametru bude použito síťové zařízení, nastavené v operačním systému jako výchozí.

-x: Parametr nezbytný pro funkčnost detektoru. Způsobuje že Tcpdump nebude vypisovat pouze hlavičky protokolů na jednotlivých vrstvách ale vypíše i obsah paketů jako 16ková čísla.

-vvv: Parametr zapíná verbose mode. Bez zapnutého verbose módu nebude Tcpdump vypisovat textově hlavičky jiných vrstev než l3. Pro nás jsou důležitá i data hlaviček l4 vrstvy a proto musí být verbose mód zapnut.

-nn: Vypíná překlad IP ardes na doménová jména. Překlad doménových jmen zbytečně spomaluje činnost Tcpdumpu, což je nežádoucí. Zapnutý překlad způsobuje nejednotnost ve výstupech a následně zbytečné komplikace při parsování hlaviček.

-S tcp: Parametr není nutné uvádět. TcpClass sám pozná, zda se jedná o paket aplikační vrstvy zabalený do protokolu tcp. Uvedením tohoto parametru zamezíme pouze tomu aby se aplikace zdržovala parsováním hlaviček l4 protokolů, které nakonec stejně zahodí.

Příklad výstupu za použití parametrů -i any -x -vvv -nn -S tcp
12:11:34.549650 IP (tos 0x0, ttl 122, id 20654, offset 0, flags[.], proto TCP(6), length 40)
178.40.104.135.55620 >192.168.2.4.6881: Flags [.], cksum 0x9262 (correct),
seq 1762530056,ack 2598696091, win 16473, length 0
0x0000: 4500 0028 50ae 4000 7a06 d2c5 b228 6887
0x0010: c0a8 0204 d944 1ae1 690e 1308 9ae4 f49b
0x0020: 5010 4059 9262 0000 0000 0000 0000

Tcpdump vypisuje hexadecimálně všechna data vč. Hlaviček IP a TCP. Parser implementovaný v TcpClass je automaticky ořezává, protože nepatří do aplikační vrstvy.

5.3 Moduly

Architektura využívající moduly ve formě dynamických knihoven byla zvolena z důvodu jednoduché rozšiřitelnosti o nové metody detekcí. Pro přidání nové techniky, stačí dodržet rozhraní modulů a není nutné jakkoliv zasahovat do kódu aplikace.
Součástí implementace TcpClass jsou tyto moduly:

Pro detekci:	SubString
	Dynamic
Pro Výstup:	Terminal
	Http

5.4 Rozhraní modulů

Každý modul, který chceme použít s TcpClass musí implementovat jednoduché rozhraní. Část rozhraní je společná a část se mírně liší. Společnou částí jsou bezparametrové a beztypové funkce Init a Destroy. Funkce slouží pro inicializaci knihovny a ukončení její činnosti. Měly by obsahovat především kód pro inicializaci globálních proměnných, pro alokaci a uvolnění dynamicky alokované paměti. Poslední částí je funkce Process, lišící se podle typu modulu (detekční a výstupní).

TcpClass při inicializaci projde adresáře určené pro uložení modulů. Pokud ve složce nalezne modul dynamicky jej načte a zavolá inicializační funkci. Naopak při ukončování dojde k volání funkce Destroy. Ve funkci destroy by po sobě měl každý modul uvolnit veškeré alokované zdroje.

- Rozhraní funkce process detekčních modulů: int process(flow*, stat*)
- Rozhraní funkce process výstupních modulů: int process(stat*)

Pro každý příchozí paket je vytvořena instance třídy Packet, vypočítán hash z hlavičky TCP protokolu a paket je podle hashe zařazen do flow. Flow, do které byl jako poslední přidán paket, je pak postupně předávána funkcím process všech načtených detekčních modulů. Pokud byl modul schopný detekovat protokol obsažený ve flow, naplní daty strukturu, předanou jako druhý parametr.

V případě že modul není schopen protokol detekovat, neměl by měnit atributy struktury stat, protože jakmile kterýkoliv modul vyplní třídu stat, TcpClass přestane flow předávat funkci process. Proto je nutné aby detekční modul vyplňoval strukturu pouze v případě, že dokázal protokol úspěšně detekovat. Předávání flow po úspěšné detekci by bylo neefektivní a zbytečné.

Po úspěšné detekci, je struktura stat postupně předána všem výstupním modulům, voláním funkce process. Funkce process výstupních modulů nějak neovlivňuje strukturu stat, pouze vypisuje statistiky shromážděné ve struktuře. Výstupní moduly jsou z důvodu efektivity volány poprvé po úspěšné detekci aby bylo na výstupu okamžitě jasné o jaký protokol se jedná.

Struktura stat obsahuje i statistické údaje jako počet paketů ve flow, minimální a maximální délka paketu. Proto jsou výstupní moduly pro úspěšně detekované flow volány periodicky po každém 32. příchozím paketu. Hodnota byla zvolena experimentálně a je optimální pro zamezení zahlcení výstupu ale přesto byl výstup dostatečně dynamický.

Poslední výpis je proveden při zanikání flow. Flow zanikne vždy, po uplynutí předdefinované doby. Timeout byl v TcpClass nastaven na 60 sekund po doručení posledního paketu.

Hodnota byla opět zvolena experimentálně. Jejím nastavením na příliš vysokou hodnotu by docházelo ke zbytečnému zahlcování operační paměti a tím i zpomalení TcpClass a v kritickém případě i k vyčerpání veškeré dostupné paměti. Nastavením na velmi nízkou hodnotu se zmenší paměťové nároky TcpClass. V tomto případě může docházet k předčasnému odstranění flow, do kterých mohou přijít nové pakety. Dojde-li k odstranění flow, musí být vytvořena znovu a opakovaně musí být provedena i detekce. Tím se zvyšují nároky na procesorový čas.

5.4.1 Modul SubString

Vlastnosti modulu:

- Analýza závislostí mezi pakety
- Tvorba detekčních pravidel
- Automatické určení názvu protokolu

Závislosti mezi pakety modul vyhledává naplněním datové struktury kombinacemi podřetězců z každého paketu a následným porovnáním takto vytvořených struktur. Dynamické vytváření regulárních výrazů se ukázalo jako značně neefektivní a extrémně náročné.

Datová struktura pro uložení podřetězců, byla implementována jako pole indexů hodnot kterých může nabývat 1 bajt. Na vytvořených indexech, jsou uloženy vektory ukazatelů, na všechny podřetězce které začínají na hodnotu indexu. Tímto způsobem se značně urychlí porovnávání a je to nejjednodušší způsob vytvoření všech možných podřetězců z řetězce. Paměťová náročnost je minimální, protože obsahem vektoru jsou pouze ukazatele na místo, kde podřetězec začíná. Celý řetězec se pak nachází v jedné jediné kopii. Detekční pravidla jsou vytvořena právě na základě shod v paktech.

Aby nedocházelo k chybám a vytváření pravidel, fungujících pouze pro jedinou relaci protokolu (např. řetězec „img=“ u http protokolu), je v modulu použit ještě jeden zdroj dat a to čtečka bannerů služeb a technika pro získání chybové zprávy služby. Modul se, po přijetí nové flow pro detekci, pokusí připojit na cílovou adresu. Když se podaří připojit, počká na uvítací zprávu (banner služby). Následně zašle nesmyslný řetězec, na který naslouchající služba odpoví chybovou zprávou. Z takto získaných dat vytvoří strukturu, jako z jakéhokoliv jiného paketu. Díky tomu má modul jistotu, že získal skutečnou hlavičku protokolu a ne např. html dokument, popisující hlavičky protokolů.

K automatickému určení názvu protokolu používá modul techniky popsané v kapitole 4.8. V případě selhání výše popsaných technik, pro získání detekčního pravidla a názvu protokolu, je detekce tímto modulem degradována na detekci dle čísla portu. Zdaří-li se detekce, modul do výstupní struktury zapíše i způsob, jakým se mu podařilo protokol detekovat.

Shrnutí vlastností

SubString dokáže analyzovat závislosti mezi pakety a na jejich základě určit konstantní řetězec, vyskytující se v rámci komunikace. Získaný řetězec je pak dále využíván pro detekci. Pro textové protokoly dokáže automaticky určit i název daného protokolu. U binárních protokolů je schopen určit detekční řetězce, ale nedokáže určit jejich název. Není-li modul schopen protokol detekovat a zároveň je protokol provozován na portu nižším než 1024, degraduje detekci na detekci podle čísla portu.

Možné komentáře výstupu:

- string in packet content (protocol have header)
  např. HTTP, SMTP, POP3, SSH, řídící spojení FTP
- Protocol dont have header or crypted. Detected by port number
  např. HTTPs, POP3s

5.4.2 Modul Dynamic

Modul určený pro rychlou a efektivní detekci BitTorrent protokolu. K detekci používá kombinaci několika technik aby byl schopen správně označit všechna BitTorrent spojení. Pro základní detekci využívá výše popsaný princip vyhledání posloupnosti „bittorrent protocol“ (viz. 4.7). Pro detekci šifrovaných spojení využívá techniku odchycení registrace k trackeru (viz. 4.7). Pro každé detekované připojení si modul uloží adresu a port na kterém klient naslouchal a každé další navazované spojení (v otevřené nebo i šifrované podobě) pro takového klienta je pak detekováno velice rychle.

Shrnutí vlastností

Dynamic dokáže detekovat libovolnou nešifrovanou probíhající BitTorrent komunikaci. Pro šifrovaná spojení umí detekovat spojení, pro která se mu podaří odchytit registraci klienta k trackeru. Kompletné šifrovaná spojení detekovat neumí.
Možné komentáře výstupu:

- Found handshake
- Known client (handshake)
- Known client (Registered to tracker)

5.4.3 Modul Terminal

Výchozí výstupní modul. Vypisuje formátovaný výstup na standardní výstup a stejná data zapisuje i do logovacího souboru.
Příklad výstupu:

Source : 192.168.2.104 : 50493 *Destination* : 147.229.176.14 : 22 *Protocol* : *SSH*
Max packet len : 928 *Packet count* : 32
Module : *SubString*
Comment : *SSH in packet content* (*protocol have header*)

Source: IP adresa a port uvedené v prvním paketu komunikace jako zdrojové. V případě že TcpClass nezachytí první paket, určí zdroj jako adresu s vyšším číslem portu. Zdrojový port je pro účely detekce nepodstatný. Určí ho náhodně operační systém a má nulovou vypovídací hodnotu.

Destination: IP adresa a port uvedené v prvním paketu jako cílové, nebo adresa s nižším číslem portu.

Protocol: Název použitého protokolu.

Max packet len: Délka datové části největšího paketu v rámci flow bez hlaviček. Je-li tato hodnota nulová, signalizuje buď neúspěšně navázané spojení nebo spojení bylo navázáno, ale neproběhla žádná komunikace.

Packet count: Celkový počet paketů přenesených v rámci flow. Hodnota reprezentuje aktuální počet, a nemusí být konečná (viz. 5.4).

Module: „podpis“ detekčního modulu, který protokol detekoval.

Comment: Komentář detekčního modulu. Položka nejčastěji obsahuje metodu kterou se podařilo modulu detekovat protokol nebo přímo detekční pravidlo.

5.4.4 Modul Http

Rozšiřující výstupní modul. Při inicializaci vytvoří jednoduchý webový server, naslouchající na portu 2011. Každé volání funkce process, zapíše statistiku do souboru. Při příchodu http požadavku modul zpracuje logovací soubor a odešle výsledek formátovaný do přehledné html tabulky. Významy položek a perioda výpisu jsou shodné s modulem terminal.

5.5 Srovnání s L7-filterem

Porovnání základních vlastností L7-filteru s TcpClass

Vlastnost	**TcpClass**	**L7-Filter**
Automatická analýza protokolu	ANO	NE
Detekce pomocí regulárních výrazů	NE	ANO
Detekce P2P sítí	ANO	ANO
Detekce šiforvaných P2P sítí	**ANO**	NE
Snadná přídání protokolu	ANO	ANO
Snadné přidání nové metody detekce	ANO	NE

L7-filter nedokáže automaticky přidávat do své databáze nové vzory. Jedinou možností, je analýza protokolu jinou aplikací nebo manuálně a následné přidání vzoru. Na rozdíl od L7-filteru si TcpClass vytváří vzory sám, na základě podobnosti v paketech. TcpClass na druhou stranu nedokáže vytvářet složitější vzory. Je-li součástí vzoru, potřebného pro detekci, dynamický řetězec vyžadující vzor typu „NazevProtokoku.*Eof“ TcpClass jej neodhalí. TcpClass pak takový protokol označí jako undetected. Tuto nepříjemnou vlastnost lze vyřešit implementací modulu schopného využívat k detekci vzory s L7-filteru.

Detekci P2P sítí implementuje L7-filter pouze pomocí vzoru, vyskytujícího se v handshake paketu (viz. 4.7). Použijeme-li L7-filter nebo jakýkoliv jiný systém popsaný v kapitole 3 pro detekci šifrovaného provozu, všechny selžou. Buď klasifikují šifrovaný P2P jako „Undetectable“ nebo „Unknown“. Na rozdíl od nich TcpClass dokáže, za určitých podmínek, takový provoz odhalit a správně oklasifikovat jako BitTorrent.

Možnosti přidávání nových metod detekce jsou pro TcpClass popsány v kapitole 5.3. L7-filter nic takového neumožňuje. Přidání nové metody detekce do L7-filteru, vyžaduje radikální zásah do kódu aplikace.

5.6 Možnosti rozšíření

TcpClass byl od počátku navržen tak, aby jej šlo snadno rozšiřovat. Využívá architekturu dynamických modulů. To z něj dělá jednoduchý framework, schopný zpracovat vstupní data a předat je ke zpracování modulům. Pro aktivaci modulu stačí, knihovnu nakopírovat do správného adresáře.

Implementace libovolné nové techniky detekce pro protokoly, u kterých běžné metody selhávají je proto snadná. Úprava TcpClass by byla nutná, jen v případě nutnosti rozšíření o detekci protokolů, přenášených pomocí UDP na l4 vrstvě nebo pro rozšíření o IPv6 na l3 vrstvě. Tyto funkce lze implementovat pouze úpravou parseru výstupu Tcpdumpu. Dalším značným vylepšením, by byla implementace techniky popsané v kapitole 4.7, pro detekci spojení s šifrovaným provozem mezi uzly a trackerem.

Kapitola 6

Závěr

V práci byly probrány základní principy počítačových sítí, nutné k pochopení technik používaných při klasifikaci síťového provozu. Dále byly probrány možnosti detekce libovolného protokolu aplikační vrstvy, možnosti a techniky automatické analýzy protokolů. Také byly dopodrobna probrány možnosti detekce BitTorrent protokolu, na základě obsahu paketů a detekce, na základě registrace k trackeru, jako obrana proti šifrování datového provozu. Následující část práce byla věnována současným aplikacím, umožňujícím klasifikaci síťového provozu. V poslední části byl probrán návrh a implementace aplikace pro detekci dynamických síťových aplikací.

Aplikace byla testována na linuxových distribucích Ubuntu a Debian. TcpClass ke své činnosti vyžaduje Tcpdump, proto by měl fungovat všude, kde je Tcpdump k dispozici.

Testováním bylo zjištěno, že při velkém vytížení sítě a velkému počtu dlouho trvajících spojení značně narůstá paměťová náročnost. To je způsobeno mechanismem odstraňování starých relací. Paměťovou náročnost značně ovlivňuje hodnota časové konstanty pro odstranění starých relací z paměti.

Součástí práce je implementace modulu pro detekci obecných protokolů a automatické vyhledávání vzorů, vhodných pro detekci na základě podobnosti mezi pakety. Tuto funkcionalitu neumožňuje žádná ze srovnávaných aplikací pro klasifikaci síťového provozu.

Další modul je nejvýznamnějším rozšířením. Je schopný detekovat šifrované datové přenosy na základě registrace uzlu k trackeru. Po zachycení registrace a identifikaci klienta a portu na kterém naslouchá, je i detekce každého dalšího spojení jen otázkou porovnání adresy a portu, což značně urychluje činnost detektoru.

Výstup aplikace je realizován implementací dvou modulů. První vypisuje data na standardní výstup a jeho úpravou je možné dosáhnout formátu pro zpracování skripty. Druhý modul funguje jako jednoduchý webový server pro vzdálený přehled o stavu sítě.

Implementovaný prototyp lze snadno rozšiřovat o nové detekční techniky, díky architektuře využívající dynamických knihoven a tím jí umožnit detekovat protokoly, vyznačující se např. neobvyklým chováním vhodným pro návrh nové metody detekce. Touto architekturou poskytuje jednoduchý framework, realizující parsování výstupu Tcpdumpu a uložení do jednoduše zpracovatelé datové struktury.

Vzhledem k faktu masového využívání P2P sítí k šíření nelegálního obsahu a nově vznikajících zákonů pro postihování takového chování, lze předpokládat dále se rozvíjející metody maskování a šifrování P2P provozu na straně uživatelů a snahu vylepšovat schopnosti detektorů na straně poskytovatelů internetového připojení.

Literatura

[1] Kolektiv autorů: The OSI model [online]. http://www2.themanualpage.org/networks/networks_osi.php3, 2003-02-19 [cit. 2011-01-23].

[2] Kolektiv autorů: Bittorrent Protocol Specification v1.0 [online]. http://wiki.theory.org/index.php/BitTorrentSpecification, 2006-12-12 [cit. 2011-04-02].

[3] Kolektiv autorů: Skype [online]. http://www.protocolinfo.org/wiki/Skype, 2008-08-13 [cit. 2011-04-10].

[4] Kolektiv autorů: Application Layer Packet Classifier for Linux [online]. http://l7-filter.clearfoundation.com/docs/start, 2010-07-22 [cit. 2011-04-18].

[5] Kolektiv autorů: BitTorrent User Manual [online]. http://www.bittorrent.com/help/manual/, 2010-08-12 [cit. 2011-04-06].

[6] L. Dostálek and A. Kabelová: *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press, 2008, iSBN 978-80-251-2236-5.

[7] The WinPcap Team: LibPCAP Man Page [online]. http://www.winpcap.org/docs/docs_412/html/main.html, 2009 [cit. 2011-04-22].

[8] V. Jacobson and L. Craig and S. McCanne: LibPCAP Man Page [online]. http://www.tcpdump.org/pcap3_man.html, 2003-11-21 [cit. 2011-04-22].

[9] V. Jacobson and L. Craig and S. McCanne: Tcpdump Man Page [online]. http://www.tcpdump.org/tcpdump_man.html, 2009-03-05 [cit. 2011-04-22].