

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCE P2P SÍTÍ

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

JINDŘICH VRBA, DiS.

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCE P2P SÍTÍ

P2P NETWORKS DETECTION

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

JINDŘICH VRBA, DiS.

VEDOUcí PRÁCE

SUPERVISOR

Ing. JIŘÍ TOBOLA

BRNO 2009

Abstrakt

Tato práce se zabývá detekcí peer-to-peer sítí. Popisuje možné způsoby identifikace na různých vrstvách ISO/OSI. Samotná praktická část se zabývá detekcí protokolů na aplikační vrstvě pomocí známých vzorů. Součástí praktické části je i prezentace výsledků detekce pomocí grafů na webu. Cílová platforma je operační systém GNU/Linux.

Abstract

This thesis deals with peer-to-peer network detection. It describes possible techniques of identification on various ISO/OSI Layers. The goal of the practical part is to examine detection on the L7 layer by means of string patterns. A presentation of the results with graphs on web pages is also included. The application is intended for the GNU/Linux operating system.

Klíčová slova

P2P, peer-to-peer, netfilter, iptables, L7-filter, detekce, RRD, GNU/Linux

Keywords

P2P, peer-to-peer, netfilter, iptables, L7-filter, detection, RRD, GNU/Linux

Citace

Jindřich Vrba: Detekce P2P sítí, bakalářská práce, Brno, FIT VUT v Brně, 2009

Detekce P2P sítí

Prohlášení

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

.....
Jindřich Vrba
20. května 2009

Poděkování

Děkuji za trpělivost, nápady a náměty vedoucímu práce Ing. Jiřímu Tobolovi.

© Jindřich Vrba, 2009.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1	Úvod	2
2	Teoretický rozbor	3
2.1	<i>P2P</i> sítě	3
2.2	<i>P2P</i> protokoly	6
2.2.1	BitTorrent	6
2.2.2	Direct Connect	7
2.2.3	FastTrack	7
2.2.4	Freenet	8
2.2.5	Gnutella	8
2.2.6	Skype	9
3	Systém pro detekci <i>P2P</i> sítí	12
3.1	Návrh	12
3.2	Implementace	13
3.2.1	Spouštěcí skript	13
3.2.2	Sběr statistik	14
3.2.3	Grafy	15
3.2.4	Statistiky za časové období	15
3.2.5	Přehled statistik	16
3.3	Použité nástroje	16
3.3.1	ipt_ACCOUNT	16
3.3.2	IPP2P	16
3.3.3	L7-filter	17
3.3.4	RRD tool	18
4	Výsledky a testy	19
4.1	<i>P2P</i> protokoly	19
4.1.1	BitTorrent	19
4.1.2	Direct Connect	20
4.1.3	Freenet	21
4.1.4	Skype	22
4.2	Propustnost	23
4.3	Chyby detekce	24
5	Závěr	25

Kapitola 1

Úvod

P2P sítě, díky jejich agresivitě (malým paketům a velkému množství spojení) dokáží velmi rychle plně vytížit linku. To může způsobit nedostupnost běžných služeb (web, elektronická pošta), případně zvýšení odezvy nebo ztrátovost.

Tento problém je velmi výrazný především u bezdrátových sítí, kde díky malým paketům může být linka plně vytížená i přes malou přenosovou rychlost. I pokud je takováto linka rozdělena mezi více uživatelů, kde každý má nastavenou maximální rychlost, tak aby mohli všichni uživatelé komunikovat zároveň, může se stát, že se ke komunikaci dostane právě jen *P2P* aplikace.

Provádět detekci *P2P* protokolů je vhodné z mnoha důvodů. Statistiky nám dávají přehled o využití sítě, odhalí problematické části a pomohou při klasifikaci provozu či nastavení *QoS*.

Cílem této práce je prozkoumat situaci na poli *P2P* sítí, navrhnout systém pro automatickou detekci a implementovat prototyp aplikace, která dokáže *P2P* sítě na základě obsahu paketů detekovat. Výsledky této aplikace by měly sloužit poskytovatelům internetu, ale i organizacím, které přistupují k internetu a potřebují mít přehled o využití sítě. Grafickým výstupem praktické části budou především grafy, ze kterých bude patrné využití *P2P* protokolů v dané síti v čase.

Následující text je členěn do několika kapitol, jejichž téma je následovné. V kapitole 2 seznámím čtenáře s teoretickými informacemi, které se týkají *P2P* sítí a popíši i některé *P2P* protokoly důkladněji. V kapitole 3 popíši praktickou část práce, tedy jak jsem postupoval při návrhu, jaké další aplikace jsem k řešení použil a v neposlední řadě také jaký je účel každé aplikace či skriptu, které jsem vytvořil. V kapitole 4 se budu věnovat testování detekce různých *P2P* sítí, porovnávat efektivitu a kvalitu detekce. V závěrečné kapitole shrnu dosažené výsledky a zamyslím se nad dalším vývojem situace v oblasti *P2P* aplikací.

Kapitola 2

Teoretický rozbor

2.1 P2P síť

Oproti běžným klient – server sítím (viz. obrázek 2.1), kde se k jednomu centrálnímu serveru připojuje mnoho klientů, v P2P síti neznáme výraz server ani klient. Místo toho je zde *peer*¹ (dále budu používat český výraz uzel), který zabezpečuje jak služby klientské tak serverové (viz. obrázek 2.2).

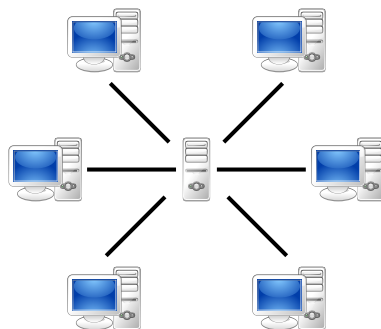
V případě architektury klient – server jsou data umístěna na serveru a klienti je z něj stahují. Nevýhoda tohoto způsobu je, že se server v případě přetížení stává úzkým hrdlem (anglicky *bottleneck*). P2P model tímto netrpí, klienti stahují různě mezi sebou (z jednoho nebo více zároveň). Mohou se tedy chovat zároveň jako klient i server.

Typicky se při potřebě něco stáhnout provádí tyto fáze [11]:

Signalizace: Připojování, vyhledávání a *keep-alive* zprávy.

Stahování: Kontaktování uzlů a stažení požadovaných dat přímo od nich.

Při vyhledávání dat uzel zjišťuje, na kterých ostatních uzlech jsou dostupná požadovaná data. V případě mnoha protokolů tato komunikace neprobíhá přímo mezi ostatními uzly, ale s nějakým centrálním serverem, který dané informace udržuje.



Obrázek 2.1: klient – server síť (převzato z [20])

První P2P aplikace využívaly pro signalizaci *TTL controlled flooding*² šíření – například

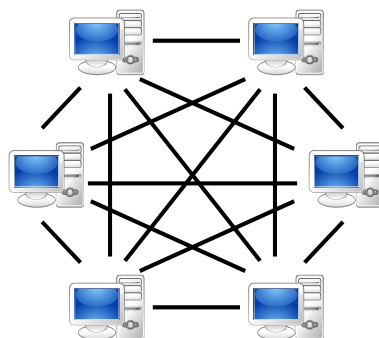
¹Peer (česky uzel) má význam vrstevník nebo rovný – od toho peer-to-peer znamená rovný s rovným.

²TTL controlled flooding funguje takto: Uzel který se dotazuje, pošle dotaz všem sousedním uzlům. Sousední uzly pokud na dotaz dokáží odpovědět, tak odpoví, jinak posílají dotaz dále a zvýší počet kroků. Pokud se počet kroků vyšplhá až na hodnotu TTL, přeposílání se zastaví [4]. Viz. obrázek 2.3 na straně 9.

starší verze aplikace Gnutella. To mělo za následek objemné toky dat a proto novější verze Gnutelly a novější protokoly posílají dotazy cíleněji a jsou tak mnohem méně náročné na využití přenosového pásma. [12]

Oproti stažení 5 minut hudby ve formátu mp3 o velikosti 5 MB, nebo hodinu a půl dlouhého videa o velikosti 700 MB je signalizace v řádu několik stovek bytů naprosto zanedbatelná. Pro detekci a statistiky nás tedy bude zajímat především fáze stažení, kde probíhá největší přenos dat.

Pokud bychom ale chtěli zamezit použití dané *P2P* sítě, zakázat signalizaci může být nejjednodušší způsob, jak toho dosáhnout. Řešit to takovým způsobem ale silně nedoporučuji, protože považuji především toto jednání za čin vedoucí uživatele a vývojáře ke snaze vyhnout se detekci. Mnohem lepší je cesta spolupráce s uživateli, definování snesitelných pravidel a omezení na rozumnou rychlost, při které už nejsou ostatní uživatelé omezováni.



Obrázek 2.2: *P2P* síť (převzato z [20])

V běžné klient – server síti, když přibývá klientů využívajících služeb nějakého serveru, je server postupně více a více zatěžován. V *P2P* sítích, kde se každý chová jako klient i server, když přibývá uživatelů, každý poskytuje zdroje a tak se teoreticky zvyšuje šířka pásma, datový prostor a výpočetní výkon. Čím více má daná *P2P* síť uživatelů, tím je teoreticky odolnější vůči výpadku. V případě čisté *P2P* sítě dokonce neexistuje *Single Point of Failure*³

Použití distribuovaného systému jako je *P2P* má svoje počátky už v podobě komunikace serverů v případě *usenet* diskuzních skupin nebo systému *SMTP* [20]. U obojího tento způsob fungoval právě jen pro komunikaci serverů, samotní uživatelé využívali systém klient – server. To je ale obdoba dnešních *P2P* sítí, kde většina neběží čistě *P2P* způsobem, ale pro některé části využívá také klient – server model, typicky pro vyhledávání dat. Například pro získání souboru *torrent* je potřeba se připojit například na web nebo FTP server a tam daný soubor nalézt.

Některé *P2P* sítě ale fungují na čistě *P2P* bázi, například *Freenet* [9]. Uzly jsou si zde tedy naprosto rovné, žádný centrální server zde neexistuje. Dalším případem by mohla být *Gnutella*, ale je to sporné, neboť využívá directory server pro informování o *IP* adresách ostatních uzlů [9].

Některé *P2P* sítě využívají výkonnější uzly jako *superuzly*, na něž se běžné uzly připojují do hvězdice (například Skype nebo FastTrack). Tím už odchází od standardního „rovný s rovným“.

P2P architektura představuje jeden z klíčových technických konceptů internetu, byl

³Single Point of Failure je taková část systému, která, když přestane fungovat, zapříčiní nefunkčnost celého systému.

popsán v prvním RFC⁴: „RFC 1 – Host Software“ z roku 1969. Dnešní *P2P* sítě se používají na celou řadu účelů. Od nejznámějšího sdílení souborů (BitTorrent) a telefonie (Skype), až po streaming audia a videa, nebo třeba pro diskuzní fóra. *P2P* způsob se využívá také v jiných oblastech, než datových přenosech, například princip člověk člověku u free softwaru a licencí *GPL* nebo *Creative Commons*. [9]

Poslední dobou se pracuje na šifrování komunikace *P2P* sítí. Díky tomu lze dosáhnout několika efektů [20]:

- Znemožnění detekce *P2P* sítí na straně poskytovatelů internetu.
- Ochrana proti odposlechnutí.
- Vyhnutí se cenzuře a detekci nezákonné výměny dat.
- Ochrana proti *Man in the Middle* útoku.
- Anonymita.

P2P sítě je možné detekovat na více vrstvách. Na *síťové* vrstvě lze detekovat podle známých *IP* adres, což lze jen u hybridních sítí fungujících z části na principu klient–server, například pro Direct Connect existuje seznam známých *hubů*, kam se lze připojit. Takto nám samozřejmě uniknou málo známé nebo neveřejné *huby*. Na *transportní* vrstvě je možné detekovat podle čísel používaných portů. Nevýhoda je, že mnoho *P2P* sítí využívá náhodné porty a nebo se snaží běžet na jiných, než standardních. Na nejvyšší *aplikační* vrstvě je možné *P2P* sítě detekovat pomocí hledání známé sekvence znaků v datech.

Když přestala být účinná detekce pomocí portů, která se běžně používá na detekci klient–server protokolů, objevily se snahy detekovat *P2P* aplikace pomocí L7 signatur v IDS⁵ [2]. Dekódování protokolu pomocí IDS je ale výpočetně velmi náročné a vyžaduje velké množství operační paměti. Proto se detekce pomocí IDS nepoužívá na detekci v reálném čase. Pro detekci v reálném čase je možné využít například rozšíření systému netfilter – *L7-filter* nebo *IPP2P*. Právě na detekci těmito nástroji se zaměřím v praktické části.

P2P protokoly se průběžně vyvíjí, často neexistuje oficiální dokumentace, nebo je zastaralá a tak je nutné pro alespoň částečně úspěšnou detekci zjistit chování protokolů pomocí zpětného inženýrství nebo odchyťáváním paketů (např. pomocí *tcpdump* nebo *wireshark*). Pro mnoho protokolů existuje více druhů klientských i serverových aplikací a každá se může chovat trochu rozdílně, což detekci dále ztěžuje.

Michal Špondr ve svém semestrálním projektu [22] uvádí, že dalším problémem *P2P* sítí je kromě datového toku i jejich dlouhotrvající přenos. Proto je vhodné přidělit *P2P* sítím nízkou prioritu. Ovšem je potřeba odlišit *P2P* protokoly pro stahování dat a *P2P* protokoly pro *IP* telefonii, například *Skype*.

Dalším problémem je legálnost použití *P2P* sítí. Ne, že by samotné jejich použití bylo nelegální, ale velmi často se využívají na výměnu programů, filmů a hudby chráněné licencemi, které takovéto zacházení zakazují. Díky tomu jsou také kontrolovány společnostmi chránícími autorská práva. Zdroj [17] informuje o právním pojetí v České republice, například že hudbu a filmy je zde nelegální nabízet, ale software i stahovat. To se týká samozřejmě jen dat pod licencí zakazující takové zacházení.

⁴RFC (Request for Comments - česky požadavek na komentář) jsou dokumenty definující chování a principy služeb a protokolů. Tyto dokumenty lze požadovat za standardy internetu.

⁵IDS (Intrusion Detection System) je nástroj určený pro detekci útoků na koncový systém.

2.2 P2P protokoly

Historický přehled popisovaných P2P protokolů [20]:

červenec 1999	publikován Freenet protokol
listopad 1999	vydána první klientská aplikace pro Direct Connect
březen 2000	první verze protokolu Gnutella
březen 2001	představení FastTrack protokolu
duben 2001	návrh BitTorrent protokolu
srpen 2003	uvolněna betaverze aplikace Skype

2.2.1 BitTorrent

Protokol BitTorrent byl navržen v roce 2001. Jeho primární určení je distribuce velkých objemů dat. Díky svému principu dokáže ušetřit šířku pásma a i hardwarové prostředky, z tohoto důvodu je používán komerčními společnostmi, nebo pro šíření Linux distribucí. Jako zdroj dat funguje nejen původní zdroj (anglicky *initial seeder*), ale kdokoli kdo už nějaká data získal. Na původním zdroji je tak možné omezit rychlost nabízení na nižší úroveň, aby největší výměna dat probíhala mezi samotnými uživateli. Původní zdroj je ale vhodné nechat zapnutý, aby byla jistota, že nabízená data budou kompletní. BitTorrent je jeden z nejvíce používaných protokolů na výměnu dat, podle některých propočtů tvoří 35 % provozu internetu⁶. Základní pojmy používané u tohoto protokolu:

peer	uživatelský uzel
seeder	uzel nabízející kompletní data
initial seeder	seeder, který s nabízením dat začal
swarm	skupina uzlů
tracker	uchovává seznam uživatelů podílejících se na výměně dat

Pokud si chce uživatel stáhnout nějaká data pomocí BitTorrentu, stáhne si nejprve soubor s příponou **torrent** (běžně z webu). Tento soubor obsahuje metadata (informace o sdílených souborech a trackeru), pomocí nichž BitTorrent klient stahuje data. Uživatelský uzel naváže spojení s více dalšími uzly pomocí *TCP* a od každého stahuje nějakou část dat. Data jsou rozdělena na stejně velké části, typicky po velikostech 256 kB, 512 kB nebo 1 MB, ke každé části se vytváří kontrolní součet pomocí SHA1 algoritmu [14]. Rychlost stahování typicky postupně narůstá (sami nemáme moc dat k nabídnutí) a na závěr se snižuje (nemáme už takový výběr z rozsahu dat).

BitTorrent běžně stahuje metodou „prvně nejvzácnější“ (anglicky *rarest first* - více se lze dočíst v [6]). Díky tomu se zvyšuje redundance, a tak výpadek seederu nemusí být fatální (pokud všichni účastníci dohromady dokáží poskládat kompletní data). Pokud ale většina účastníků brzy po stažení dat ukončuje komunikaci (nestávají se seedery), torrent postupně „umírá“. Proto je BitTorrent vhodný především na aktuálně velmi žádaná data.

BitTorrent je velmi oblíbený protokol, podle některých zdrojů, se přes něj sdílí okolo 1.1 PB⁷, tudíž jeho detekce je dobře otestována. Je snaha se detekci vyhnout a proto se začíná komunikace šifrovat, za tímto účelem se využívá *Message Stream Encryption*⁸.

⁶<http://www.dslreports.com/shownews/56403>

⁷<http://isohunt.com/forum/viewtopic.php?t=145853>

⁸Message Stream Encryption je protokol vytvořený za účelem znemožnění klasifikace síťového provozu – snaží se, aby hlavičky a volitelně i data vypadaly náhodně. [15]

2.2.2 Direct Connect

Direct Connect patří mezi centralizované *P2P* sítě. Vznikl v listopadu 1999, jeho autorem je Jonathan Hess. Direct Connect je textový protokol, kde jsou příkazy posílány jako čistý nekryptovaný text. K protokolu neexistuje oficiální specifikace, proto všechny klientské aplikace kromě původního Neo-modus vznikly pomocí zpětného inženýrství (anglicky. *reverse engineering*). To může mít za následek niance mezi implementacemi. [22, 7]

Klienti se připojují na *huby*, to jsou centrální servery poskytující informace o ostatních klientech a slouží k vyhledávání dat a chatu. Samotné stahování funguje skutečným peer-to-peer způsobem, kdy mezi sebou klienti komunikují pomocí *TCP* protokolu. Na vyhledávání se standardně využívá protokol *UDP*, ostatní komunikace mezi *hubem* a klientem probíhá pomocí *TCP*.

Peer-to-peer komunikace mezi klienty probíhá na bázi *slotů* – počet otevřených slotů znamená možný počet spojení.

Klient se standardně připojuje na *hub* na *TCP* port 411, po navázání komunikace se jako první ohlásí *hub*. Seznam příkazů protokolu je možné nalézt na <http://www.teamfair.info/wiki>. Oddělovače protokolu jsou \$, | a mezera.

Direct Connect je možné detekovat na mnoha vrstvách. Na síťové vrstvě můžeme využít fakt, že existuje seznam veřejných DC *hubů*⁹. Na transportní vrstvě můžeme detekovat použití Direct Connect pomocí detekce komunikace s *hubem* – na *TCP* portu 411. Další možnost je detekce na aplikační vrstvě, která bude testována dále.

2.2.3 FastTrack

FastTrack vznikl v roce 2001, jeho autory jsou Niklas Zennström, Janus Friis a tým programátorů Jaana Tallinna – stejná skupina, která později vytvořila Skype. FastTrack je distribuovaná síť, výkonné počítače s dobrým připojením jsou automaticky povýšeny na *superuzly*, což jsou vlastně lokální vyhledávací uzly. Běžní klienti se připojují na superuzly, aby nahráli informace o souborech, které sdílí a za účelem vyhledávání.

První klientské aplikace byly proprietární software, používaný protokol nebyl zveřejněn, ale reverzním inženýrstvím byl zjištěn způsob komunikace uzel – superuzel. V roce 2003 to byla nejpopulárnější *P2P* síť s celkem 2,4 milionů uživatelů [19]. Používala se především pro výměnu audia ve formátu mp3. Nejznámější aplikace využívající protokol FastTrack je Kazaa.

Na kontrolu stažených dat používá FastTrack hašovací algoritmus UUHash, ten vyniká svou rychlostí¹⁰, bohužel je ale schopen přehlédnout i masivní poškození dat. To bylo v historii použito na cílené šíření poškozených souborů a tím poškozování této sítě. [19]

Kazaa verze 1.5 a vyšší může použít šifrovanou zprávu pro poslání odpovědi na dotaz [11], ale inicializační data šifrovacího algoritmu jsou posílána v čisté podobě. Šifrovaná i nešifrovaná zpráva obsahuje položku *X-Kazaa-SupernodeIP*, která specifikuje *IP* adresu a port superuzlu. To lze použít k detekci. Posílaná data mají tento tvar (převzato z [11]):

⁹Seznam veřejných *hubů* například na http://dc.torrentblog.net/dc_hublisty.php.

¹⁰Rychlost hašovacího algoritmu UUHash spočívá v hašování jen určitých částí souboru.

```
Požadavek na stažení:  
GET /.files HTTP/1.1\r\n  
Host: IP adresa/port\r\n  
UserAgent: KazaaClient\r\n  
X-Kazaa-Username: \r\n  
X-Kazaa-Network: KaZaA\r\n  
X-Kazaa-IP: \r\n  
X-Kazaa-SupernodeIP: \r\n
```

```
Odpověď:  
HTTP/1.1 200 OK\r\n  
Content-Length: \r\n  
Server: KazaaClient\r\n  
X-Kazaa-Username: \r\n  
X-Kazaa-Network: \r\n  
X-Kazaa-IP: \r\n  
X-Kazaa-SupernodeIP: \r\n  
Content-Type: \r\n
```

2.2.4 Freenet

Základní kameny projektu Freenet položil Ian Clarke v červenci 1999. Freenet je decentralizovaná anonymní síť. Samotná Freenet aplikace je free software a je napsána v jazyku Java. Hlavní cíl projektu je možnost publikovat a získávat informace bez cenzury. Pro zajištění tohoto cíle jsou síťová data šifrována a prochází přes více uzlů. Tímto způsobem je velmi obtížné zjistit, kdo data požaduje a jaký je jejich obsah.

Každý uživatel sítě poskytuje určitou šířku pásma a diskový prostor. Na jaká data bude diskový prostor využit, nemůže uživatel ovlivnit a ani nemůže zjistit, jaká data jsou na jeho disku uložena, neboť jsou šifrována. Data se na disk ukládají nebo se z něj mažou podle jejich žádanosti v síti. Tyto pravidla přispívají k anonymitě sítě – je velmi těžké dopátrat, odkud data pocházejí.

Freenet neslouží jen ke sdílení dat, ale jak píše autoři [21], je to anonymní internet v internetu. Pomocí této sítě je možné publikovat webové stránky, komunikovat, sdílet data i posílat elektronickou poštu.

Data se šifrují pomocí veřejných a privátních klíčů, na výměnu klíčů se používá algoritmus JFK¹¹. Kromě šifrování využívá protokol Freenet ještě například náhodná zarovnání dat. Jeho síťová komunikace je proto velmi těžko detekovatelná. Autoři považují Freenet za určitým způsobem detekovatelný pomocí sledování charakteristik toků, neboť vysílá mnoho malých *UDP* paketů pro udržení rychlosti vyhledávání a až vyžádaná data se přenáší po větších paketech. [21]

2.2.5 Gnutella

Gnutella je plně distribuovaný protokol. Uzly jsou si rovné, v případě tohoto protokolu se pro ně používá název *servent*. Uzel se do sítě připojí tak, že naváže spojení s nějakým serventem v síti Gnutella. Po připojení do sítě komunikuje s ostatními serventy (vyhledávání a signalizace) pomocí Gnutella protokolu (převzato z [11]): Připojení do sítě: `GNUTELLA CONNECT/<verze protokolu>\n\n`, odpověď: `GNUTELLA OK\n\n`. Pro představu o struktuře Gnutella sítě slouží obrázek 2.3.

Samotné stahování dat probíhá mezi serventem požadujícím data a serventem tato data poskytujícím pomocí protokolu podobném HTTP (převzato z [11]):

¹¹JFK (Just Fast Keyring) je varianta protokolu Diffie-Hellman pro výměnu tajných klíčů po nezabezpečeném médiu.

```

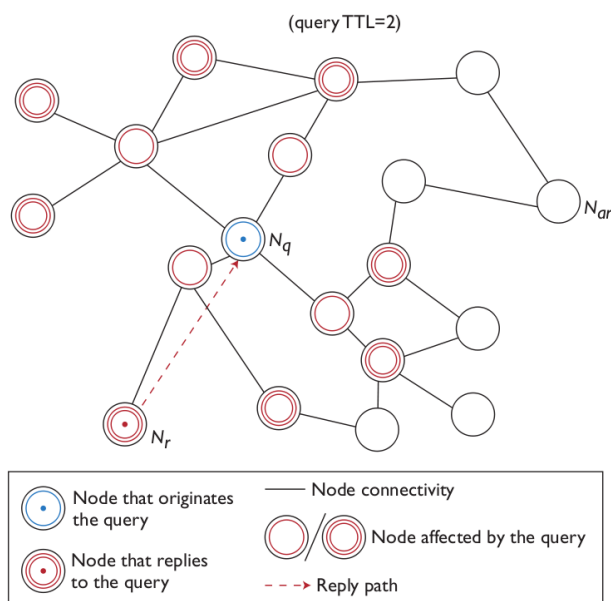
Požadavek na stažení:
GET /get/<Index souboru>/<Jméno souboru>
/HTTP/1.0 \r\n
Connection: Keep-Alive\r\n
Range: byte=0-\r\n
User-Agent: <Jméno>\r\n
\r\n

```

```

Odpověď:
HTTP 200 OK\r\n
Server: <Jméno>\r\n
Content-type: \r\n
Content-length: \r\n
\r\n

```



Obrázek 2.3: Příklad vyhledávání v *P2P* síti založené na *TTL controlled flooding*. Uzel N_q hledá data, která vlastní uzel N_r . Červené kruhy představují počet kroků. (převzato z [4])

2.2.6 Skype

Skype byl vyvinut v roce 2003 stejnými vývojáři jako Kazaa. Slouží pro komunikaci – VoIP¹², posílání textových zpráv a později i videokonference. Dokáže také přenášet soubory, poskytuje hlasovou schránku, volání do běžné telefonní sítě (*SkypeOut*) a umožňuje také volání z běžné telefonní sítě do sítě Skype (*SkypeIn*). Pro telefonování přes Skype je potřeba relativně malá přenosová rychlost, podle měření popsanych v literatuře [1] je potřeba 32 kb/s pro rozumnou kvalitu hlasu, pro hlasové konference více, v závislosti na počtu účastníků.

Skype síť je distribuovaná (viz. obrázek 2.4), kromě běžných uzlů existují i superuzly. Kandidát na superuzel je uzel, který má dostatečně výkonný procesor, dostatek operační paměti a rychlé připojení k internetu. V nastavení nelze vynutit, aby se aplikace nestala superuzlem.

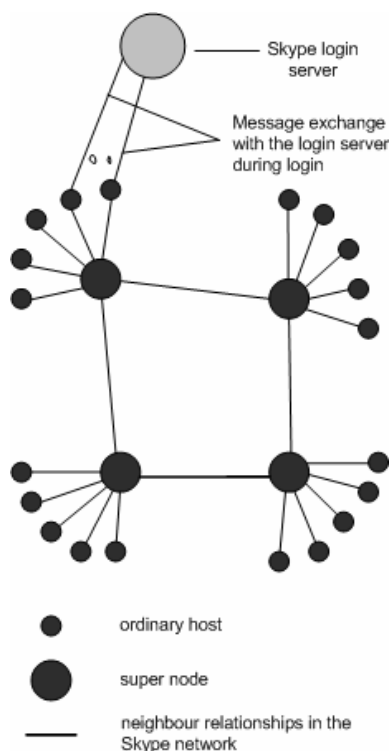
Všechna komunikace v síti Skype je silně šifrována [10], na výměnu klíčů je použito *RSA*, na samotné šifrování provozu pak *AES*.

Skype poslouchá na *TCP* i *UDP* portu, jeho číslo je generováno náhodně při instalaci aplikace. Pokud nejsou obsazeny porty 80 a 443, poslouchá i na nich. Na signalizaci používá

¹²VoIP (Voice over IP) je technologie pro přenos digitalizovaného hlasu přes počítačovou síť.

TCP, na samotnou komunikaci upřednostňuje *UDP*, ale může využít i *TCP*, pokud je *UDP* zakázáno.

Běžný uzel se při přihlášení musí připojit k nějakému superuzlu a autentizovat se ke Skype *login serveru*. Seznam několika superuzlů a portů, na kterých komunikují, má běžný uzel v tabulce superuzlů (*host cache*). Tato tabulka je pro funkci Skype klíčová, aby bylo možné připojit se po instalaci, má zde Skype aplikace už několik superuzlů zapsáno. Po instalaci si Skype aplikace postupně plní tabulku superuzlů, které jsou umístěny poblíž, nebo jsou pro něj jiným způsobem vhodné. Jelikož Skype síť je velmi dynamická a superuzly vznikají a zanikají, může mít klientská aplikace v seznamu až 200 superuzlů a jejich portů. [1]



Obrázek 2.4: Struktura sítě Skype (převzato z [1]).

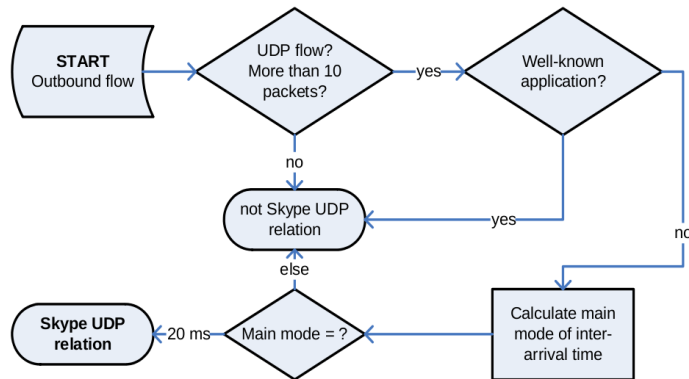
Získání informací o uživateli a jejich vyhledávání je řešeno decentralizovaně. Skype tvrdí, že aplikace má implementovanou technologii která dokáže vyhledat jakéhokoliv uživatele, který se do Skype sítě připojil během posledních 72 hodin.

Přihlášení do sítě probíhá následovně: Nejprve se pokusí navázat spojení se superuzlem na *UDP* portu a *IP* adrese z tabulky superuzlů. Pokud se nezdaří, snaží se použít *TCP* na stejném portu. V případě neúspěchu zkusí *TCP* na portu 80 a poté *TCP* na portu 443. V případě neúspěchu opakuje celý proces 4×. Toto je zjednodušená verze, kdy má aplikace v tabulce superuzlů jen jeden záznam. V reálném případě, kdy je zde uvedeno více superuzlů, se snaží připojit pomocí *UDP* na kterýkoliv z nich a až poté se pokusí o spojení pomocí *TCP*. Po úspěšném přihlášení se autentizuje pomocí jména a hesla na *login serveru*¹³. [1]

Skype po přihlášení do sítě udržuje komunikaci mimo superuzel i s uživateli v seznamu

¹³Login server je jediný centrální server ve síti Skype. Jsou na něm uložena uživatelská jména a hesla, pomocí kterých se zde uživatelé Skype sítě autentizují.

přátel. Děje se tak pomocí protokolu *UDP*. Účelem je monitorovat spojení mezi stranami. Sleduje se, jaký je stav uživatele na druhé straně, zda je přihlášen a dostupný. Zároveň se takto logicky zjistí, zda je vůbec možné komunikovat s protistranou pomocí *UDP*. Pokud je zjištěno, že je zakázána, použije se pro případné volání hned protokol *TCP*. Tyto zprávy lze tedy považovat za *UDP* ping mezi klienty a dopřednou přípravu na volání. Průměrně se protistrany kontaktují po dvaceti sekundách. Při samotném volání se využije stejný komunikační kanál, jen se zvýší intenzita posílání paketů a jejich velikost. Při více takovýchto spojení lze spolehlivě určit komunikační port Skype klienta, což je možné použít jako část detekce Skype protokolu (viz. obrázek 2.5). [10]



Obrázek 2.5: Detekce *UDP* relací mezi Skype uzly (převzato z [10]).

V literatuře [10] popisují detekci *P2P* sítě Skype na síťové a transportní vrstvě, pomocí sledování charakteristik datových toků a paketů. Také sledují velikosti paketů i datových toků. Dále informují, že při přenosu hlasu se přenáší *UDP* pakety o velikosti 70 B až 420 B. Data se přenáší i v případě ticha, pakety jsou pak velké méně než 60 B.

Kapitola 3

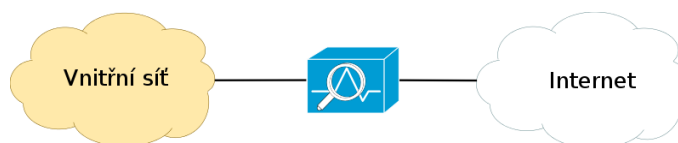
System pro detekci *P2P* sítí

3.1 Návrh

Pro implementaci tohoto projektu jsem si vybral operační systém GNU/Linux – především kvůli jeho svobodě, rozšíření a síle. Konkrétně distribuci Debian, to ale neznamená, že by nebylo možné použít jinou Linux distribuci.

Pro detekci budu používat upravené Linux jádro – distribuční jádro Debianu se všemi aktualizacemi, rozšířené o *L7-filter*, *ACCOUNT* a *IPP2P*. Statistiky budu sbírat pomocí *L7-filteru* a zaznamenávat v operační paměti pomocí *ACCOUNT*. Dále je zapisovat pomocí bash skriptu do *RRD* databáze¹. Statistiky budou dostupné na webu jako grafy, generované v PHP pomocí *rrdtool*. Pro přehled nejvíce komunikujících *IP* adres budu zaznamenávat statistiky pomocí bash skriptu do MySQL databáze, z které budou rychle přístupné. Počítání těchto statistik a zapisování do databáze se bude provádět každý den v noci, zobrazovat se budou opět na webu.

Jak píše výše, data jsem se rozhodl ukládat do RRD databáze. Toto řešení má spoustu výhod – tyto databáze postupem času nenarůstají a *rrdtool* z nich dokáže relativně jednoduchým způsobem generovat grafy. Pro každou *IP* adresu a protokol vytvářím vlastní databázi. RRDtool samozřejmě umožňuje zaznamenávat data pro všechny protokoly do jediné (obdobu tabulek) a na první pohled to vypadá, že by to bylo i nejlepší řešení. Ale každý uživatel internetu má jiné chování a používá jiné *P2P* sítě. Proto je zbytečné uchovávat statistiky pro všechny *P2P* protokoly pro danou *IP* adresu. Navíc, pokud bude každý protokol ve vlastní RRD databázi, lze jednoduše přidávat, ale i odebírat další filtry.



Obrázek 3.1: Detektor přímo na cestě do internetu.

Detektor je možné umístit buď jako bridge nebo router přímo mezi vnitřní počítačovou sítí a internet (viz. obrázek 3.1). To je vhodné, pokud už v této pozici je nějaký počítač se systémem GNU/Linux a tudíž se zbytečně nepřidává do cesty další server a nezvyšuje tak zbytečně odezvu.

¹RRD databázi mám na mysli soubor s příponou *rrd*, kde se uchovávají data o dané *IP* adrese a protokolu.

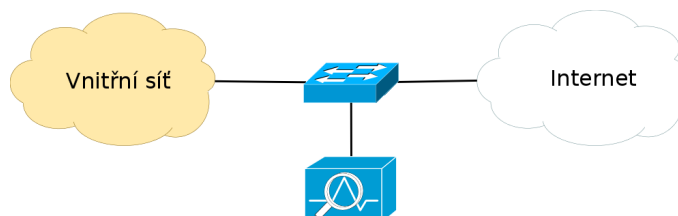
Pokud se takováto metoda nehodí, je možné klonovat síťový provoz a posílat jej na detektor stojící osamocně mimo cestu do internetu (viz. obrázek 3.2). Klonovat je možné na switchi i routeru. Při použití *hubu* není potřeba nic klonovat, neboť *hub* ze svého principu posílá příchozí byty na všechna rozhraní. Tato metoda má v případě klonování rámců na switchi malý háček, totiž klonované rámce přicházející na detektor mají MAC adresu jinou než jakou má detektor. Mohlo by se zdát, že přepnutí do promiskuitního režimu síťové karty to vyřeší. Skutečně pomocí aplikací jako `tcpdump` uvidíme že k nám tato data přichází, ale `iptables` je budou ignorovat. Řešení spočívá v přepnutí se do režimu bridge, pak už `iptables` bude příchozí komunikaci kontrolovat. Nastavení serveru do módu bridge vypadá v systému GNU/Linux takto:

```
brctl addbr br0 (1)
```

```
brctl addif br0 eth0 (2)
```

```
ifconfig br0 up (3)
```

Na řádce 1 vytvoříme „bridge“ rozhraní. Na řádce 2 přiřadíme síťové karty tomuto rozhraní (logicky jsou potřeba 2 síťové karty, ale v našem případě data jen přijímáme a pak zahazujeme, a tak bude stačit jedna). Na řádce 3 aktivujeme „bridge“ rozhraní.



Obrázek 3.2: Detektor mimo cestu do internetu, síťová data jsou klonována.

3.2 Implementace

3.2.1 Spouštěcí skript

Pro počáteční spuštění detekce využívám standardní unixový způsob inicializačních skriptů umístěných v `/etc/init.d/`, konkrétně `/etc/init.d/account-p2p`. Spouštěcí skript vymaže používanou tabulku v `iptables` a naplní ji pravidly pro detekci *P2P* protokolů. Například spuštění pravidla pro BitTorrent vypadá takto:

```
$iptables -A FORWARD -m layer7 --l7proto bittorrent -j ACCOUNT \
--addr $net --tname bittorrent
```

Ačkoliv je tento prototyp navržen na detekci *P2P* protokolů, je velice jednoduše rozšiřitelný o jakoukoliv detekci podporovanou systémem `netfilter`. To může být například běžná detekce podle portů a *IP* adres. Ale také podle typu *ICMP* zprávy, délky paketu, *TCP* příznaků, *TTL* nebo třeba i podle státu, ze kterého byl paket poslán. Například pro detekci požadavků na *TCP* spojení stačí přidat následující řádek. Sbíráni statistik pro tuto detekci a vypsání na webu je řešeno automaticky.

```
$iptables -A FORWARD -p tcp --tcp-flags SYN,RST,ACK,FIN SYN -j ACCOUNT \
--addr $net --tname syn
```

3.2.2 Sběr statistik

Statistiky jsou sbírány pomocí iptables a cíle *ipt_ACCOUNT*. Pomocí *iptaccount* získávám statistiky ve tvaru:

```
10.0.1.200;7;659;137;157368
```

Je možné nechat si je vypsat i v uživatelsky přívětivějším formátu, který už nejspíš nepotřebuje komentář:

```
IP: 10.0.1.200 SRC packets: 7 bytes: 659 DST packets: 137 bytes: 157368
```

Statistiky jsou sbírány po 2 minutách a zaznamenávány do RRD databáze. Zaznamenává se velikost v obou směrech procházejících dat. Statistiky se uchovávají pro maximální dobu jednoho roku. Uchovávat po celou tuto dobu statistiky po 2 minutách by zabralo mnoho diskového prostoru (v našem případě s 32 bit hodnotami přes 2 MB na každou databázi). Ani pro generování grafů a sledování statistik není nutné uchovávat všechna data, proto jsou s přesností 2 minuty uchovávány jen pro 1 den. Týden staré statistiky jsou uchovávány s přesností 14 minut, měsíční s přesností 60 minut a pro roční dostačuje 24 hodin. Takto má výsledná databáze zhruba 44 kB, téměř 50 krát menší velikost.

Ukázka vytvoření RRD databáze s výše popsány vlastnostmi:

```
rrdtool create "rrd/$proto-$ip.rrd" --step 120 \  
  DS:down:GAUGE:160:0:U DS:up:GAUGE:160:0:U \  
  RRA:AVERAGE:0.5:1:720 RRA:AVERAGE:0.5:7:720 RRA:AVERAGE:0.5:30:744 \  
  RRA:AVERAGE:0.5:365:365;
```

Zapisování hodnot vypadá zhruba takto:

```
rrdtool update "rrd/$proto-$ip.rrd" -t down:up N:$B_dst:$B_src
```

Při spouštění je kontrolováno, zda aplikaci spustit uživatel *p2p-detector*. Je to z důvodu bezpečnosti, aby jej nemohl spouštět *root*, který má zbytečně velká oprávnění. Jediná aplikace, kterou je nutné spustit s oprávněními uživatele *root* je *iptaccount*. To se vyřeší relativně jednoduše, nastaví se *setuid* bit, který zařídí, aby měl běžící program oprávnění svého vlastníka, nikoliv uživatele který jej spustil. Aby *iptaccount* nemohl spouštět kdokoliv, je potřeba povolit spouštění jen pro skupinu do které patří uživatel *p2p-detector* a nastavit vlastnictví souboru touto skupinou:

```
chgrp p2p-detector /usr/local/bin/iptaccount  
chmod 4750 /usr/local/bin/iptaccount
```

Sbírání statistik je spouštěno každé 2 minuty pomocí plánovače *cron*. Aplikace kontroluje, zda neběží minulé spuštění. Pokud ano, informuje o této situaci a ukončí se. Záznam v */etc/crontab* může vypadat takto:

```
*/2 * * * * p2p-detector /opt/p2p-detector/account
```

Pro testování protokolů jsem potřeboval statistiky přesněji rozdělené do kratších časových úseků. Na tento účel jsem používal skript *account-fast*. Ten je variantou *account* sbírající statistiky řádově po jednotkách sekund. Většinou jsou používal rozmezí 2 s. Tento skript nebyl pouštěn plánovačem *cron*, ale řešil plánování ve vlastní režii.

3.2.3 Grafy

Grafy se generují pomocí rrdtool z RRD databází. Slouží k tomu 3 propojené PHP skripty. Skript `stat.php` představuje rozhraní pro uživatele, zkoumá, zda jsou prohlížečem posílány parametry dávající smysl (*IP* adresa a protokol) a zda existují nějaké statistiky pro danou *IP* adresu a protokol. Místo uvedení jména protokolu je možné uvést klíčové slovo `full`, poté se vygenerují statistiky pro všechny použité protokoly a danou *IP* adresu do jednoho grafu.

Samotné generování grafů nechává na skriptu `graf.php`. Ten přijímá parametry *typ*, *IP adresa* a *protokol*. Jako *typ* může být zadán `denni`, `tydenni`, `mesicni` nebo `rocni`.

Dále definuje způsob generování grafů pro rrdtool – velikost a rozsah grafu, formát obrázku, popisky grafu, mřížkování a hustota popisu os *x* a *y*, způsob vykreslování statistik a barvy pro jednotlivé protokoly. V závislosti, jestli se mají generovat statistiky pro jeden konkrétní protokol nebo pro všechny protokoly pro danou *IP* adresu, spojuje statistiky z více RRD databází a provádí nad nimi výpočty.

Pokud je povolen JavaScript, velikosti grafů určuje použitelná plocha prohlížeče. O to se stará skript `graf_js.php`. Velikost grafů je pak volena tak, aby se na téměř celou plochu zobrazil právě jeden graf a tak bylo možné vidět statistiky co nej přesněji. Pokud JavaScript nefunguje, nebo je zakázán, jsou použity výchozí hodnoty, které jsou ideální pro rozlišení plochy 1024×768 .

Pro generování grafů, které byly použity v této bakalářské práci (například obrázek 4.4), jsem vytvořil skript `graf_fast.php`. Ten generuje grafy pevně daných rozměrů a vypisuje jen souhrnné statistiky. Jeho hlavní rozdíl oproti `graf.php` je možnost zadání časového rozsahu grafu (parametr `delka`) a posunu (parametr `do`). To nejlépe předvedu následující ukázkou odkazu na statistiky (tento odkaz posloužil k vytvoření právě grafu na obrázku 4.4):

```
https://localhost/ipac/graf-fast.php?ip=0.0.0.0&proto=skypetoskype\  
&delka=10min&do=now-23min
```

3.2.4 Statistika za časové období

Pro rychlý přehled, kdo v síti nejvíce využívá danou P2P síť je vhodné mít připravené statistiky dopředu. Na zaznamenávání se nabízí ukládat data do databáze. V tomto praktickém řešení se už využívají RRD databáze, ale ty nejsou pro tento účel ideální. V RRD jsou totiž statistiky ukládané po 2 minutách (to není úplně přesné, detailněji tento problém popisují v kapitole Sběr statistik na straně 14) a pro získání statistik by se při každém přístupu musely hodnoty počítat.

Aby nebylo nutné počítat hodnoty při každém přístupu, budou připraveny v MySQL databázi, ze které se přečtou velmi rychle. Do MySQL databáze se budou statistiky zapisovat pravidelně v noci, kdy je server minimálně zatížen. Pravidelné zapisování bude opět řízeno plánovačem `cron`. Záznam v `/etc/crontab` může vypadat následovně:

```
23 5 * * * p2p-detector nice -n 19 /opt/datstat/datstat
```

Příkaz `nice -n 19` sníží prioritu procesu na nejnižší možnou úroveň. To může být vhodné provést, pokud nám na počítači v tuto dobu běží další a důležitější aplikace. Na vygenerování statistik pravděpodobně příliš nespícháme a tak můžeme upřednostnit ostatní procesy.

3.2.5 Přehled statistik

Pro zobrazení statistik za časové období, popsaných v předchozí kapitole slouží skript `index.php` – tedy úvodní stránka webu. Tato webová stránka slouží ke dvěma účelům. Prvním účelem je zobrazit v tabulce statistiky pro každou *IP* adresu, protokol a směr takovým způsobem, aby bylo možné přímo na webu jednoduše řadit statistiky podle hodnot.

Druhým účelem je sloužit jako úvodní stránka, tedy odkazovat na následující grafy. Políčka s hodnotami v této tabulce slouží jako odkaz na graf k dané *IP* adrese a protokolu. Ve sloupci *IP* adresa, je pak souhrnný přehled k dané *IP* adrese označené v kapitole Grafy jako full.

3.3 Použité nástroje

3.3.1 ipt_ACCOUNT

Nástroj `ipt_ACCOUNT` je určen pro účtování procházejících dat v systému GNU/Linux, je licencován pod GPL verze 2. Především kvůli rychlosti je napsán jako jaderný modul. Pro zaznamenání dat se používá ve spojení s netfilter/iptables systémem, je schopen účtování sítě o maximální velikosti 24 bitů. Účtování pro síť velikosti 8 bitů zabírá v operační paměti 4 kB, pro větší síť se prostor v paměti alokuje jen v případě potřeby. [5]

`ACCOUNT` zaznamenává statistiky pro každou *IP* adresu zvlášť. Například pro detekci všech procházejících dat v dané síti spustíme:

```
$iptables -A PREROUTING -j ACCOUNT --addr $net --tname all
```

Pro získání statistik se používá knihovna `libipt_ACCOUNT`, je možné si vypsat statistiky o velikosti prošlých dat nebo počtu paketů (výpis zkrácen):

```
# iptaccount -l all
Showing table: all
Run #0 - 4622 items found
IP: 192.168.76.41 SRC packets: 66 bytes: 30697 DST packets: 92 bytes: 8006
```

3.3.2 IPP2P

`IPP2P` je netfilter/iptables rozšíření licencované pod svobodnou licencí GPL. Prohledává data L7 protokolu a hledá části, podle kterých by rozpoznal *P2P*, rozhoduje se navíc podle informací z transportní vrstvy (délky paketů). Podle vzorů nedokáže označit všechny pakety patřící k danému *P2P* protokolu, proto je nutné označovat celé spojení. Bohužel tento software se nyní aktivně nevyvíjí, poslední verze je z 27. 9. 2006.

Ukázka detekce Direct Connect s použitím značení spojení pomocí `CONNMARK`²:

```
$iptables -A FORWARD -j CONNMARK --restore-mark (1)
$IPTABLES -A FORWARD -m mark --mark 2 -j ACCOUNT --tname directconnect (2)
$IPTABLES -A FORWARD -m mark --mark 2 -j ACCEPT (3)
$IPTABLES -A FORWARD -m IPP2P --dc -j MARK --set-mark 2 (4)
$IPTABLES -A FORWARD -j CONNMARK --save-mark (5)
$IPTABLES -A FORWARD -m mark --mark 2 -j ACCOUNT --tname directconnect (6)
```

²`CONNMARK` je netfilter modul pro označování spojení.

Řádek číslo 1 načte označení spojení, v případě, že dané spojení už bylo označeno dříve. Na řádku 2 zaznamenáváme statistiky pro Direct Connect spojení a na řádku 3 říkáme, že aktuální paket už není potřeba dále kontrolovat. Na řádku 4 detekujeme nová Direct Connect spojení. Na řádku 5 ukládáme značení pro nová spojení. Na řádku 6 zaznamenáváme statistiky pro nová spojení Direct Connect.

3.3.3 L7-filter

L7-filter je klasifikátor pro linuxový projekt netfilter licencovaný pod svobodnou licenci GPL verze 2. Síťové toky klasifikuje na základě dat aplikačního protokolu. L7-filter začal vznikat roku 2003. Hlavním impulzem bylo, že do té doby byly klasifikátory komerční, velmi drahé a velmi pomalu se přizpůsobovaly nově vznikajícím protokolům nebo změnám existujících.

L7-filter byl původně patch pro QoS³ systém obsažený v Linuxovém jádře. To se neosvědčilo a proto se vývojáři rozhodli upravit jej pro netfilter. Díky použití netfilteru lze L7-filter použít nejen na detekci, ale i shapování provozu nebo účtování. Dnešní stabilní verze funguje právě na tomto systému. Pracuje se také na verzi, která detekuje data v uživatelském prostoru, využívá k tomu opět netfilter a nechává si posílat data do uživatelského prostoru pomocí *QUEUE* [18]. Tato verze má ale momentálně problémy s novějšími jádry, s kterými ji nelze zkompileovat. Stabilní verze má naproti tomu problém s během na SMP⁴ systémech.

L7-filter dokáže pracovat nad *TCP* a *UDP* protokoly transportní vrstvy a *IP* a *ICMP* protokoly síťové vrstvy (nepracuje s *IPv6* ani *ICMPv6*) [18]. Zkoumá data z více paketů (používá netfilter *connection tracking*), standardně nahlíží do prvních deseti paketů nebo 2 kB, záleží co nastane dříve [18]. Po prozkoumání tohoto množství dat je celé spojení klasifikováno podle protokolu, ke kterému patří, nebo označeno jako neidentifikovatelné a už se dále nezkoumá. L7-filter rozlišuje neidentifikovatelné a nové spojení a je možné je i detekovat pomocí *unknown* respektive *unset*.

L7-filter nezkoumá všechna data patřící k danému spojení, to by bylo výkonnostně velmi náročné a také zbytečné. Zajímavá data, podle kterých by bylo možné identifikovat protokol jsou na začátku spojení, poté už dochází k přenosu souborů nebo například hlasu. Snažit se zde detekovat protokol by bylo mrhání výpočetním výkonem. Přestože L7-filter zkoumá jen začátek toku, je mnohem náročnější na výkon, než například detekce podle čísla portu, proto autoři upozorňují [18], že je vhodné jej využít jen v těchto případech:

- Detekce protokolů které využívají náhodná čísla portů (např. *P2P* protokoly).
- Detekce protokolů využívající nestandardní port (např. *HTTP* na portu 1111).
- K rozlišení protokolů využívajících stejný port (např. *P2P* síť využívající port 80).

Zjednodušeně je možné říci, že L7-filter je *CONNMARK* a *string*⁵ modul dohromady. Vzory vyhledává přes více paketů a není je nutné zadávat ručně, neboť jsou připraveny v konfiguračních souborech.

³QoS (*Quality of Service*) – schopnost nastavit různé priority různým síťovým tokům a tak umožnit službám náchylným na zpoždění mít rychlou odezvu. Více je možné se dočíst například v [3].

⁴SMP (*Symmetric MultiProcessing*) je označení pro využití více procesorů na jedné základní desce, které spolu sdílí paměť [13].

⁵*String* je netfilter modul pro vyhledávání textových nebo hexadecimálních vzorů v datech.

L7-filter je jednoduše rozšiřitelný, pro přidání dalšího protokolu stačí přidat do adresáře `/etc/l7-protocols/protocols` soubor se vzorem, ten může vypadat například jako v tabulce 3.1.

```
# Freenet - Anonymous information retrieval - http://freenetproject.org
# Pattern attributes: poor veryfast fast
# Protocol groups: p2p document_retrieval open_source
# Wiki: http://www.protocolinfo.org/wiki/Freenet
# Copyright (C) 2008 Matthew Strait, Ethan Sommer; See ../LICENSE

freenet
# Freenet is intentionally hard to identify...
# This is empirical, only tested on one computer.
^\x01[\x08\x09][\x03\x04]
```

Tabulka 3.1: Konfigurační soubor `freenet.pat` s informacemi a vzorem pro freenet protokol.

3.3.4 RRD tool

RRDtool je zkratka pro Round Robin Database tool [8], je to GPL nástroj vyvinutý Tobiasem Oetikerem a dalšími. Jeho primární určení je monitorování (datových toků, teploty a jiných číselných dat zaznamenávaných po stejných časových rozmezích). Round robin je technologie, kdy se pracuje stále se stejným množstvím dat a ukazatelem na naposled vloženou položku.

RRDtool jako celek není jen běžná databáze. Stejně jako běžná databáze uchovává data, ale navíc vykresluje grafy, zajišťuje, aby se data ukládala po konstantní době (pokud nejsou vložena vloží se hodnota *unknown*). Nemusí ani ukládat přímo zadávaná data, ale třeba rozdíl vůči minule ukládané hodnotě. Stejně tak při vykreslování grafů může nad daty provádět různé výpočty a kombinovat data z více zdrojů a vykreslovat až výsledné hodnoty.

Kapitola 4

Výsledky a testy

Při vyhodnocování výsledků detekce síťového provozu je potřeba ověřit a zaměřit se na následující [11]:

Přesnost Použitá technika by měla mít malý počet *false positives* (chybné označení za *P2P* pro jiný protokol) a zároveň nízký počet *false negatives* (neoznačení patřičného protokolu).

Propustnost Technika by měla být schopná zpracovat velké síťové toky s dobrou přesností a měla by být co nejméně výpočetně náročná.

V této kapitole postupně předvedu jakým způsobem jsem testoval uváděné protokoly. Poté se zaměřím na propustnost sítě při použití detekce na aplikační vrstvě a shrnu výše avizované *false positives* a *false negatives*.

4.1 *P2P* protokoly

P2P protokoly jsem testoval pomocí routeru, který prováděl NAT a jedním počítačem za tímto routerem (znázorněno na obrázku 3.1 na straně 12). Na počítači jsem ve výchozím stavu neměl kromě základních síťových služeb (DNS, DHCP) spuštěnou žádnou síťovou aplikaci. Na routeru jsem měl aktivovány všechny testované filtry a procházející data jsem měřil pomocí skriptu `account-fast`. Výsledky jsem zobrazoval v grafu generovaném pomocí `graf-fast.php` – porovnával jsem celkový tok vůči toku testovaného protokolu. Po ukončení jednotlivých testů jsem kontroloval, které další protokoly byly naměřeny a tak zjišťovat *false positives*.

Pro každý protokol jsem také testoval, jak úspěšná je detekce po již zahájené komunikaci. To jsem prováděl jednoduše tak, že jsem začal s detekcí až po spuštění dané aplikace. Některé další rozšířené testy jsem volil podle poznatků z literatury, blíže je popisuji u daných protokolů.

4.1.1 BitTorrent

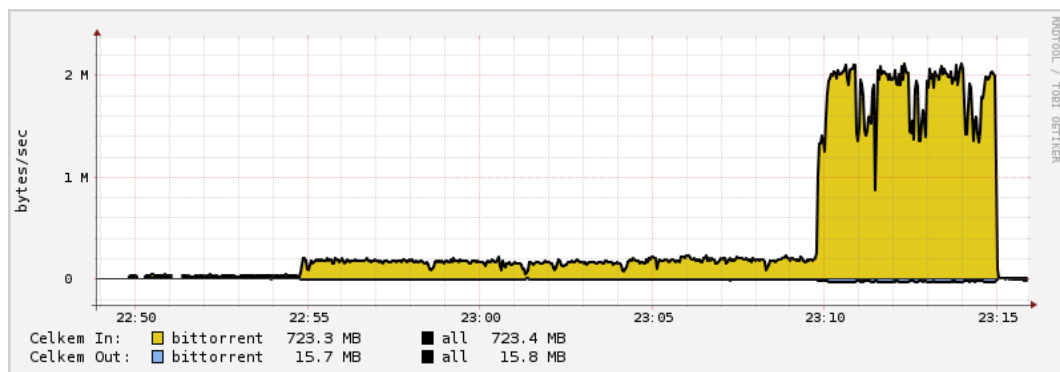
V případě protokolu BitTorrent se uzly pro výměnu souborů připojují k sobě navzájem. Pro detekci BitTorrentu se tedy omezíme jen na výměnu dat, neboť samotné vyhledávání se provádí pomocí webu. Po navázání spojení pokračuje komunikace klientů tokem paketů s obsahem začínajícím číslem 19 (0x13) a pokračující textem „BitTorrent protocol“ [11], tomu také odpovídá L7 vzor:

```
^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=)
```

Výše uvedený vzor není kompletní, je to zjednodušená rychlejší verze vzoru (jeho rychlost je uvedena v závorce v tabulce 4.1), která neodchytne některé části komunikace v případě BitComet aplikace. Celý L7 vzor má tento tvar:

```
^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=get \
/announce\?info_hash=|get /client/bitcomet/|GET /data\?fid=)\
|d1:ad2:id20:|\x08'7P\)[RP]
```

Během testů jsem prověřoval originální BitTorrent verze 3.4.2 a KTorrent verze 3.1.4, oboje se stejnými výsledky – detekce byla velmi přesná. Zajímavý je průběh stahování (obrázek 4.1), kdy jsem ze začátku přijímal data velmi nízkou rychlostí, později jsem se začal podílet na nabízení dat a díky tomu vzrostla stahovací rychlost. Na závěr jsem měl největší rychlost nabízení a stahování. To odpovídá principům popsaným v literatuře. Dále literatura popisuje postupné snižování rychlosti stahování v závěru, kvůli menšímu množství uzlů vlastníci tato data. To se mi v tomto případě nepotvrdilo, může to být zaviněno tím, že jsem stahoval oblíbenou distribuce Debian, na jejíž distribuci se podílelo velké množství uzlů.



Obrázek 4.1: Detekce BitTorrent protokolu.

Detekce pomocí IPP2P doznala stejně kvalitních výsledků jako v případě L7-filteru. Detekovat protokol po začátku stahování nepředstavovalo ani jednomu nástroji problém. Při použití šifrování dat, nedokázal ani L7 filter ani IPP2P detekovat ani jediný byte.

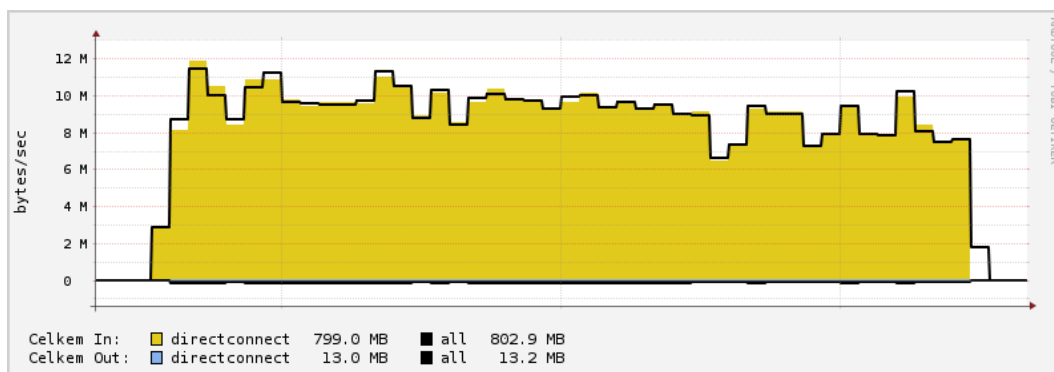
4.1.2 Direct Connect

Pro detekci Direct Connect používá L7-filter vzor `^(\$mynick |\$lock |\$key)`. Rozpoznávání protokolu je díky jednoduchosti protokolu velmi přesné. Spojení detekuje například na těchto datech:

```
@$MyNick KuzmaKuzmic|$Lock EXTENDEDPROTOCOLABCABCABCABCABC \
Pk=DCPLUSPLUS0.699ABCABC|$Supports MiniSlots XmlBZList ADCGet \
TTHL TTHF ZLIG|$Direction Upload 17704|$Key A 00 0 0 0 0 0|@.
```

Testoval jsem klientskou aplikaci LinuxDC++ verze 1.0.2 a připojoval jsem se na VerliHub 0.9.8c. Detekce byla velmi přesná (viz. obrázek 4.2). Nástroje L7-filter i IPP2P dokázaly

detekovat jen samotné stahování, vyhledávání a komunikaci s *hubem* detekováno nebylo. To ale není problém detekovat na transportní vrstvě – používá standardně *TCP* port 411. Ani při použití nestandardního portu by nedetekování nečinilo problém, neboť se zde přenáší jen malé množství dat. Ani L7-filteru ani IPP2P nečinilo problém, pokud začalo stahování až po začátku detekce.



Obrázek 4.2: Detekce Direct Connect protokolu.

4.1.3 Freenet

L7-filter používá pro detekci vzor `^x01[\x08\x09][\x03\x04]`, sám autor vzoru ale píše, že tento vzor fungoval částečně v jednom případě a bude těžko fungovat znovu. Proto mě nepřekvapilo, že detekce byla neúspěšná. Data protokolu Freenet jsou šifrována a protokol je navržen za účelem, aby nebyl detekovatelný. Na ukázkou zachycená data obsažená v jednom z UDP paketů (sloupce: offset, data hexadecimálně, data textově – tečky představují netisknutelné znaky).

```

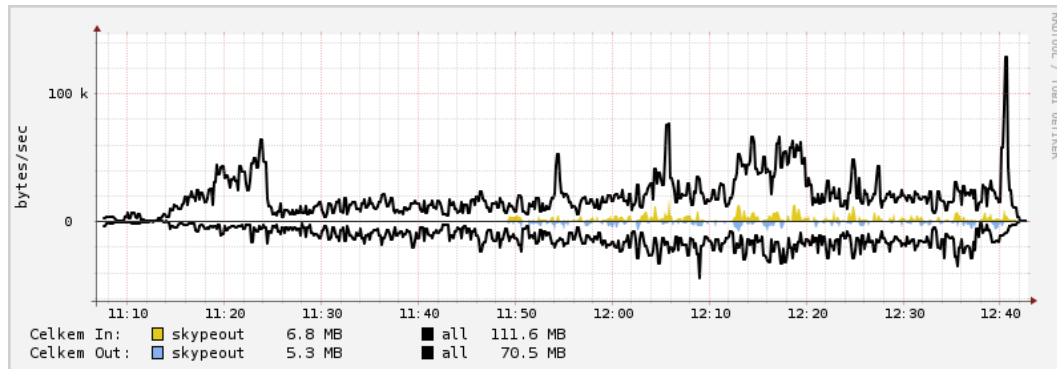
0000  9f 78 5e f2 40 7b 3c 8b df 5d 00 76 f5 08 50 c2  .x^.@{<...}.v..P.
0010  8f dd 5b 3d 5b 0b ad a6 f0 b2 1a 68 c8 8c c8 86  ..[=[.....h....
0020  be 47 3f 28 6a ba a7 54 50 41 97 4b 56 53 d5 9e  .G?(j..TPA.KVS..
0030  8a 78 99 69 bc 6f d2 3b 5d 95 eb 90 b2 40 d2 74  .x.i.o.;]....@.t
0040  5e 90 1b 07 f4 ad ee 2d d9 e6  ^.....-...

```

Samotný Freenet tedy detekován nebyl, ale díky šifrování se objevilo mnoho náhodných dat a to ověřilo kvalitu jiných vzorů. SkypeOut označil přibližně 6,6 % (viz. obrázek 4.3) těchto dat. Jedná samozřejmě o *false positive*.

Poskytovatelům internetu ale nahrává, že Freenet není momentálně určen větší mase lidí, ale spíše jen zapáleným jedincům, kteří jsou si vědomi jeho výhod. Běžný uživatel, který má zájem stahovat data, narazí při použití Freenetu v porovnání s jinými *P2P* sítěmi (jako BitTorrent nebo Direct Connect) na několik nevýhod. Například pro úspěšné stahování dat je nutné mít Freenet zapnutý delší dobu dopředu, aby mohl navázat spojení s dalšími klienty.

Při mém testování Freenet zhruba po hodině navázal spojení s necelými dvaceti dalšími uzly. Stahování bylo i tak nesmírně pomalé. Z grafu je vidět, že v případě stahování je téměř stejná výměna dat jako bez. Při rychlosti připojení do internetu 100 Mbit jsem v případě nečinnosti i stahování dosahoval rychlostí v řádu několik desítek kB/s.



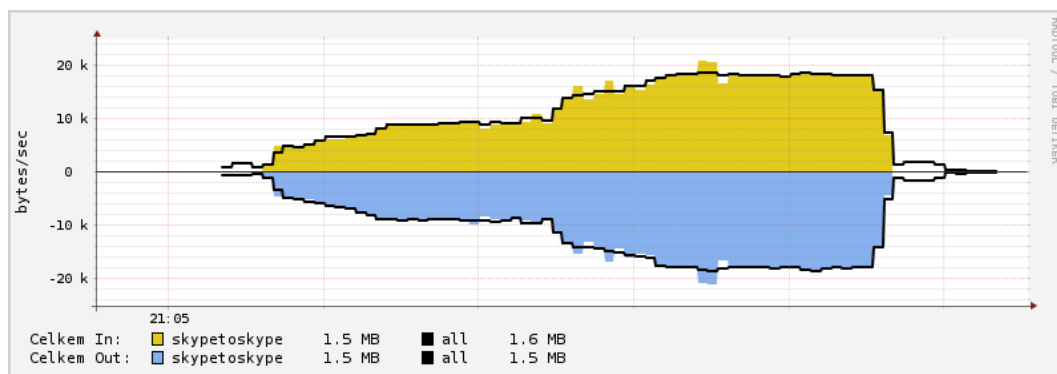
Obrázek 4.3: Detekce Freenet – false positive SkypeOut.

4.1.4 Skype

Testoval jsem Skype verze 2.0.0.72 pro operační systému GNU/Linux. Na druhých stránkách byl taktéž použit operační systém GNU/Linux. Pro systém Windows ve stejnou dobu existovala verze 4.0, tu jsem netestoval. *UDP* pakety v síti Skype lze detekovat pomocí pochybného L7 vzoru: `^..\x02.....`

To znamená že datová část paketu musí být dlouhá alespoň 16 B a na třetí pozici v datech aplikačního protokolu musí být daná hodnota. *TCP* pakety jsou pomocí L7 v podstatě nedetekovatelné. [16]

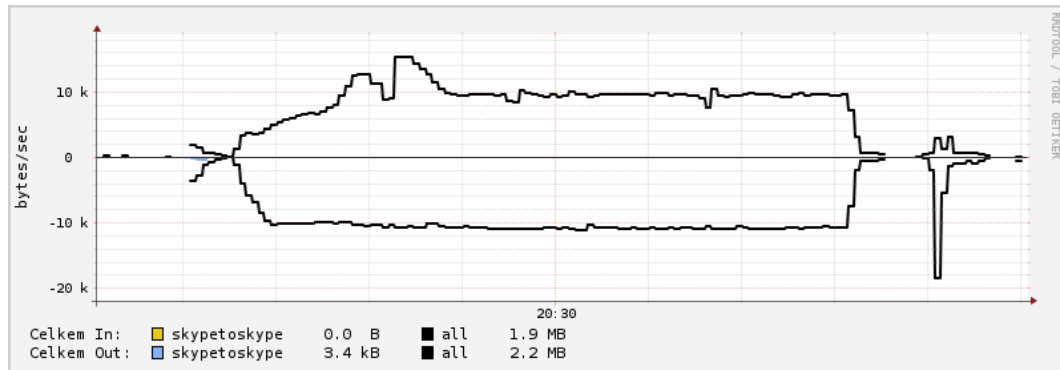
Nejprve jsem otestoval volání a konferenci po protokolu *UDP*. Z obrázku 4.4 je možné vyčíst, že část na začátku a konci nebyla detekována – to je signalizace v síti Skype, která probíhá přes *TCP*. Samotné volání (první část, tok zhruba 9 kB obousměrně) a telefonická konference (mezi třemi účastníky – druhá část, tok zhruba 18 kB) bylo detekováno s velkou přesností.



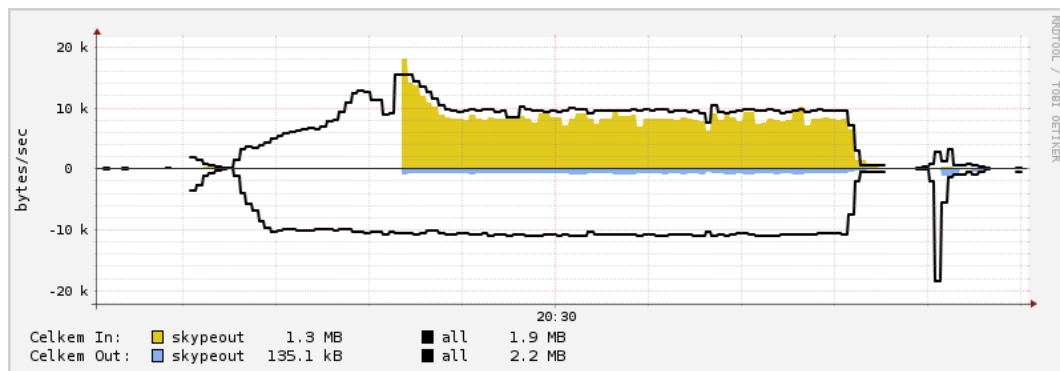
Obrázek 4.4: Volání v Skype síti a telefonická konference po *UDP*.

V dalším testu jsem kompletně zakázal komunikaci pomocí *UDP* a vyzkoušel, jak je na tom detekce při použití protokolu *TCP*. Z obrázku 4.5 je zřejmé, že filter tuto komunikaci detekovat nedokázal. Při kontrole *false positives* jsem ale odhalil detekci pomocí filtru SkypeOut, ta dokázala detekovat část downloadu a malou část uploadu. Jak je vidět z obrázku 4.6, detekce začala až v průběhu posílání. Je možné, že detekce Skype po protokolu

TCP tedy alespoň částečně možná bude.



Obrázek 4.5: Volání ve Skype síti a telefonická konference po TCP.



Obrázek 4.6: Volání ve Skype síti a telefonická konference po TCP – false positive SkypeOut.

V následujícím testu jsem zkoušel testovat detekci posílání souborů v síti skype. Některým uživatelům se soubory posílaly pomocí protokolu *UDP*, potom byla detekce velmi přesná. Některým se posílaly pomocí *TCP* (i přesto, že volání probíhalo přes *UDP*) a detekce byla téměř nulová. *SkypeToSkype* detekoval okolo 2% downloadu a *SkypeOut* zhruba 1% uploadu.

Při testování *SkypeOut* jsem dosáhl zhruba totožných výsledků, jako v případě *SkypeToSkype*. *UDP* komunikace byla detekována na téměř 100%, *TCP* vůbec.

4.2 Propustnost

Propustnost detektoru s aktivními filtry jsem testoval pomocí generátoru paketů a také v reálném prostředí u poskytovatele internetu.

U poskytovatele jsem použil techniku klonování dat na switchi, jak je zobrazeno na obrázku 3.2 na straně 13. Jako detektor sloužil vysloužilý notebook s procesorem Intel Core Duo U2500 o frekvenci 1,2 GHz a 1 GB operační paměti. V době měření bylo u tohoto poskytovatele aktivně využíváno okolo 1550 uživatelských přípojek s unikátní *IP* adresou. Tok dat představoval přibližně 12 MB/s a 15 200 paketů za sekundu dohromady oběma směry.

V módu bridge bez jakýchkoliv použitých filtrů bylo v těchto podmínkách zatížení procesoru přibližně 8%. Po zapnutí počítání celkových procházejících dat pro každou *IP* adresu pomocí *ACCOUNT* jsem neměřil žádné navýšení zátěže. Následně jsem testoval náročnost některých protokolů, vynechal jsem takové, kde bylo zřejmé, že by je server výkonnostně nezvládl. Kompletní přehled výsledků najdete v tabulce 4.1.

Protokol	Rychlost L7-filter	Úspěšnost		Zatížení	
		L7-filter	IPP2P	L7-filter	IPP2P
BitTorrent	61,1 s (0,04 s)	99 %	99 %	není reálné (17 %)	3 %
Direct Connect	0,03 s	99 %	99 %	16 %	2 %
FastTrack	73,5 s	—	—	není reálné	2 %
Gnutella	0,2 s	—	—	22 %	3 %
Freenet	0,02 s	0 %	—	17 %	—
SkypeToSkype	0,04 s	UDP 99 %	—	18 %	—
SkypeOut	9,82 s	UDP 99 %	—	98 %	—

Tabulka 4.1: Testování L7-filteru a IPP2P. Ve sloupci rychlost jsou časy detekce pro 100 000 vzorků daného protokolu. Úspěšnost představuje procento detekovaného provozu daného protokolu, které jsem naměřil a popsal v textu. Zatížení představuje nárůst zatížení testovacího počítače při zapnutí detekci daného protokolu – měřeno v reálném prostředí poskytovatele internetu na počítači s procesorem Intel 1,2 GHz. Zatížení pro L7-filter představuje nárůst oproti použití úplně bez L7-filteru. Při použití více L7-filter vzorů je celkový nárůst zatížení nižší než součet. Například při použití filtrů pro BitTorrent (rychlejší verze), Direct Connect, Freenet, Gnutella a SkypeToSkype je součet nárůstů 90 %, přitom skutečný nárůst byl jen přibližně 45 %. Všechny uvedené hodnoty jsou přibližné.

4.3 Chyby detekce

False positive je termín používaný pro případ detekce protokolu, který použit nebyl. Takové chování jsem naměřil především u filtru SkypeOut, který například při testování Freenet detekoval přibližně 6,6 % této komunikace. Z velmi velké části označil také volání Skype – Skype v případě protokolu TCP. To ale nepovažuji za velký problém, neboť detekce SkypeToSkype a SkypeOut může být prováděna zároveň, neboť účel klasifikování tohoto provozu bude velmi pravděpodobně stejný.

Zamezit *false positive* by bylo možné pomocí zlepšení vzorů. Tedy hledat souvislosti mezi chybně detekovaným protokolem a protokolem který měl být detekován a řešit společné znaky. To ale může mít za následek složitější vzor, což je pak výpočetně náročnější na detekci. Případně můžeme ze vzoru odstranit sporné části, díky čemuž ale mohou vzniknout *false negatives*.

False negative je označení pro takovou neúspěšnou detekci, kdy protokol, který měl být detekován, detekován nebyl. Z testovaných protokolů můžu jmenovat především Freenet. Ten byl za tímto účelem také navržen. Dále byl těžko detekovatelný hovor Skype – Skype při použití protokolu *TCP* (z přibližně 35 % detekováno filtrem SkypeOut) a hovor SkypeOut (nedetekován).

Pro řešení *false negatives* se opět můžeme vydat cestou sledování komunikace a opravení vzorů. Při opravování vzorů si musíme naopak dát pozor na vznik *false positives*. Někdy může být také lepší méně složitý vzor, který sice některou komunikaci nedetekuje, ale zase může být podstatě rychlejší (viz. například BitTorrent).

Kapitola 5

Závěr

V této práci byl navržen a implementován systém pro detekci *P2P* aplikací, byly využity standardní prostředky Linux jádra obohacené o patche pro detekci vzorů na aplikační vrstvě. Získané výsledky byly přehledně vizualizovány pomocí webového rozhraní a grafů. Činnost navrženého systému byla otestována jak paketovým generátorem, tak v reálném provozu u poskytovatele internetu. Bylo změřeno zatížení serveru při použití jednotlivých filtrů. Výsledná aplikace byla nasazena na serveru poskytovatele, díky ní mají technici přehled o využití sítě a dokáží odhalit možné zdroje problémů.

Navrženou práci lze dále rozšířit o klasifikaci a shaping síťového provozu. Omezení *P2P* sítí na vhodnou úroveň, nebo řazení do méně prioritních tříd dokáže zlepšit výkonnost sítě, což přináší užitek samotným uživatelům dané sítě. Přestože aplikace byla vyvíjena za účelem detekce *P2P* provozu, lze ji velmi jednoduše rozšířit o jakoukoliv detekci, kterou umožňuje systém *netfilter*.

Měřením jsem zjistil, že detekce na aplikační vrstvě pomocí vzorů je v některých případech opravdu náročná. Pro detekci *P2P* na 100 Mbit lince už lze použít jen velmi rychlé vzory L7-filtru. Kombinovaná detekce na aplikační a transportní vrstvě v případě nástroje *IPP2P* byla výpočetně mnohem méně náročná. Díky uváděným hodnotám si mohou zájemci o detekci udělat velmi rychle obrázek, co je v jejich síti možné detekovat a na co jim výkon serverů už nestačí.

Detekce *P2P* sítí ze strany poskytovatelů internetu a snaha o jejich maskování ze strany uživatelů je věčný koloběh. Když se podíváme zpět, první *P2P* měly relativně jednoduchý protokol a byly jednoduše detekovatelné. Jak se objem jejich provozu zvětšoval, poskytovatelé začali omezovat a někteří dokonce zakazovat takovýto provoz. To pomohlo k zániku daných *P2P* sítí, nebo změně jejich protokolu. Postupně se zlepšovaly maskovací metody a poskytovatelé měli obtížnější práci.

Je pravděpodobné, že do budoucna bude stále větší snaha vyhnout se detekci. Stále více *P2P* aplikací bude šifrováno a tak detekce na aplikační vrstvě bude stejně neúčinná jako dnes běžně používaná detekce na síťové a transportní vrstvě. Pro detekci bude pravděpodobně nutné zkoumat charakteristiku datových toků, časování, velikosti paketů a objemy přenášených dat. Například Michal Špondr popisuje ve svém semestrálním projektu [22] detekci *P2P* provozu založenou na modelu chování, autoři Marcell Perényi a Sándor Molnár podobným způsobem detekují síťový provoz Skype [10].

Ideální situace by bylo najít kompromis, kdy uživatelé využívají *P2P* sítě tak, aby neomezovali ostatní uživatele a poskytovatelé jim toto umožní. Zde jde ale o pochopení problému a ochotu jej sám řešit, což z globálního hlediska bude těžko proveditelné, navíc když na první pohled by měl takto uživatel omezovat sám sebe.

Literatura

- [1] Baset, S. A., Schulzrinne, H.: *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol* [online]. <http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0412017>, 2004-12-05 [cit. 2009-05-08].
- [2] Bin, L., ZhiTang, L., Hao, T.: *A methodology for P2P traffic measurement using application signature work-in-progress*. Brussels, Belgium: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007, ISBN 978-1-59593-757-5.
- [3] Campbell, A., Coulson, G., Hutchison, D.: A quality of service architecture. In *ACM SIGCOMM Computer Communication Review*, New York, USA: Association for Computing Machinery, 1994, ISSN 0146-4833, s. 6–27, volume 24, issue 2.
- [4] Doval D., O’Mahony, D.: Overlay networks: A scalable alternative for P2P. In *IEEE Internet Computing*, IEEE Computer Society, 2003, ISSN 1089-7801, s. 79–82, volume 7, issue 4.
- [5] Intra2net AG 2009: *ipt_ACCOUNT* [online]. http://www.intra2net.com/en/developer/ipt_ACCOUNT/, 2009 [cit. 2009-04-24].
- [6] Legout, A., Urvoy-Keller, G., Michiardi, P.: *Rarest First and Choke Algorithms Are Enough* [online]. <http://www.imconf.net/imc-2006/papers/p20-legout.pdf>, 2006-10-27 [cit. 2009-05-07].
- [7] Newitz, A.: *Sharing the Data* [online]. <http://www.metroactive.com/papers/metro/07.12.01/work-0128.html>, 2001-12-07 [cit. 2009-04-22].
- [8] Oetiker, T.: *RRDtool* [online]. <http://oss.oetiker.ch/rrdtool/>, 2009-04-29 [cit. 2009-04-30].
- [9] P2P World: *What is Peer-to-Peer (P2P) Networking?* [online]. <http://www.solyrich.com/how-p2p-works.asp>, 2006 [cit. 2009-05-04].
- [10] Perényi, M., Molnár, S.: *Enhanced Skype Traffic Identification*. Brussels, Belgium: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007, ISBN 978-963-9799-00-4.
- [11] Sen, S., Spatscheck, O., Wang, D.: *Accurate, scalable in-network identification of p2p traffic using application signatures*. New York, USA: Association for Computing Machinery, 2004, ISBN 1-58113-844-X, s. 512–521.

- [12] Sen, S., Wang, J.: Analyzing peer-to-peer traffic across large networks. In *IEEE/ACM Transactions on Networking (TON)*, Piscataway, USA: IEEE Press, 2004, ISSN 1063-6692, s. 219–232, volume 12, issue 2.
- [13] Strnad, P.: *SMP - Symmetric Multi Processing* [online]. <http://www.volny.cz/drd/smp.html>, prosinec 1998 [cit. 2009-05-05].
- [14] více autorů: *Bittorrent Protocol Specification v1.0* [online]. <http://wiki.theory.org/index.php/BitTorrentSpecification>, 2006-09-12 [cit. 2009-04-18].
- [15] více autorů: *Message Stream Encryption* [online]. http://www.azureuswiki.com/index.php/Message_Stream_Encryption, 2007-12-01 [cit. 2009-05-13].
- [16] více autorů: *Skype* [online]. <http://www.protocolinfo.org/wiki/Skype>, 2008-08-13 [cit. 2009-04-26].
- [17] více autorů: *Seminář „Ochrana autorských práv“* [online]. <http://www.kn.vutbr.cz/az-prednaska/2008/>, 2008-12-03 [cit. 2009-04-18].
- [18] více autorů: *Application Layer Packet Classifier for Linux* [online]. <http://17-filter.sourceforge.net>, 2009-01-07 [cit. 2009-04-18].
- [19] více autorů: *FastTrack* [online]. <http://en.wikipedia.org/wiki/FastTrack>, 2009-04-26 [cit. 2009-04-28].
- [20] více autorů: *Peer-to-peer* [online]. <http://en.wikipedia.org/wiki/Peer-to-peer>, 2009-04-30 [cit. 2009-05-02].
- [21] více autorů: *The Free Network Project* [online]. <http://freenetproject.org>, 2009-05-07 [cit. 2009-05-16].
- [22] Špondr, M.: *Detekce provozu v P2P sítích založená na modelu chování*. Semestrální projekt, Brno, 2008.