VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE IDENTIFIKACE PROTOKOLŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PETR KOBIERSKÝ
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE IDENTIFIKACE PROTOKOLŮ

HARDWARE ACCELERATION OF PROTOCOL IDENTIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR KOBIERSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2008

Abstrakt

S dynamickým rozvojem sítí dochází také k bouřlivému rozvoji síťových aplikací a služeb. Pro zajištění jejich kvality je nutné v síťových tocích identifikovat aplikační protokoly a podle jejich typu jednotlivé toky omezovat nebo blokovat. Tato práce popisuje metody identifikace síťových protokolů, diskutuje jejich přesnost a analyzuje možnosti použití těchto metod na vysokorychlostních sítích. Na základě provedené studie byl vytvořen a analyzován model identifikace aplikačních protokolů. Výsledky analýzy byly použity k návrhu hardwarové architektury akcelerující výpočetně náročné části algoritmu identifikace aplikačních protokolů. Navržené řešení je schopno pracovat na 10 Gb/s sítích a exportovat informace o aplikačních protokolech ve formě NetFlow protokolu.

Klíčová slova

Identifikace protokolů, monitorování sítí, vyhledávání vzorů, NetFlow, FPGA

Abstract

Dynamic growth of computer networks encourages rapid development of network applications and services. To provide sufficient network service quality, it is important to limit some network flows based on their application protocol type. This thesis deals with the methods of network protocol identification and discusses their accuracy and suitability for multi-gigabit networks. Based on the analysis, a protocol identification model was created and evaluated. The model was used for the design of hardware architecture accelerating computationally intensive operations of protocol identification. The proposed solution is able to work on 10 Gb/s networks and export protocol information using NetFlow protocol.

Keywords

Protocol identification, network monitoring, pattern matching, NetFlow, FPGA

Citace

Petr Kobierský: Hardwarová akcelerace
identifikace protokolů, diplomová práce, Brno, FIT VUT v Brně, 2008

Hardwarová akcelerace identifikace protokolů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kořenka. Další informace mi poskytli kolegové z projektu Liberouter. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Kobierský
14. května 2008

Poděkování

Především bych rád poděkoval vedoucímu své diplomové práce panu Ing. Janu Kořenkovi za odborné vedení a čas věnovaný konzultacím této práce. Také bych rád poděkoval kolegům z projektu Liberouter za poskytnutí informací a zajištění technické podpory při návrhu a implementaci

Obsah

Kapitola 1

Úvod

Internet se od svého zrodu v roce 1987 rozšířil z původních několika desítek tisíc uživatelů na aktuální více než jednu miliardu uživatelů. Z původní sítě pro akademické účely se tak Internet stal jedním z nejvyužívanějších komunikačních a informačních médií. Toto obrovské rozšíření s sebou přineslo ohromný rozvoj v oblasti síťových technologií. Stále větší požadavky na rychlost a nízkou latenci přenosu vedly k vyvinutí kvalitních multigigabitových síťových technologií, které umožňují přenášet obraz a zvuk ve vysokém rozlišení mezi různými místy naší planety.

Internet je v současné době používán velkým množstvím služeb jako je email, webové stránky, distribuované databáze, VoIP, internetové bankovnictví atd. Každá z těchto služeb klade na síť jiné požadavky a síť je musí být schopna zajistit. S rozšířením Internetu se bohužel objevily také problémy s bezpečností. Po Internetu se šíří obrovské množství virů, systémy jsou napadány hackery a napadené stanice útočníci využívají k DDOS útokům nebo rozesílání spamů.

Se spoustou těchto bezpečnostních problémů se lze pomocí specializovaných zařízení jako jsou IDS/IPS systémy vypořádat. Problém zajištění kvality služeb lze s úspěchem řešit pomocí většiny dnešních routerů. Všechny tyto systémy však pro svou správnou činnost potřebují identifikovat konkrétní typ provozu, aby v něm mohli vyhledávat určitou množinu nebezpečných vzorů nebo mohli nežádoucí aplikace omezit či blokovat.

Klasické metody identifikace provozu založené na číslech portů se v poslední době ukazují jako nedostatečné [35, 43]. Je to hlavně z důvodu zvětšujícího se množství aplikací (síťové hry, multimédia streaming, P2P sítě pro sdílení obsahu), které pro svou komunikaci využívají dynamicky zvolených portů. Některé P2P aplikace dokonce používají k zamaskování své činnosti porty, které jsou běžně využívány službami jako HTTP a FTP. Z těchto důvodů je nutné hledat nové přístupy pro zajištění kvalitního rozpoznávání aplikačních protokolů.

Jedním z přístupů jak tento problém řešit, je parsování přenášených dat na aplikační vrstvě. Tato možnost ovšem vyžaduje přístup k celým přenášeným datům (problém fragmentace na L3 a L4 vrstvě) a také přístup k specifikaci komunikačního protokolu, který v případě nestandardních protokolů nemusí být k dispozici. Proto se většina detekčních mechanizmů spoléhá na vyhledávání opakujících se vzorů v komunikaci, které mohou danou aplikaci identifikovat. V poslední době se bohužel ukazuje, že i tento přístup již v mnohých případech nedostačuje z důvodu stále častějšího šifrování provozu na aplikační vrstvě.

Jednou z možností jak klasifikovat šifrovaný provoz je zkoumat jeho statistické vlastnosti na L2-L4 vrstvě [36, 15, 39]. Tyto vlastnosti nejsou závislé na přenášeném obsahu a dokáží velmi přesně identifikovat charakter provozu (interaktivní, stahování velkého ob-

jemu dat, streaming, VoIP). Tento přístup lze rovněž použít k identifikaci jednotlivých typů komunikujících aplikací s přesností mezi 70-90 % dle použitých statistických vlastností a klasifikačních metod.

Klasifikace aplikačních protokolů je řešena v řadě volně dostupných aplikací [49, 56, 51] k omezení šířky přenosové kapacity nežádoucích aplikací. Identifikace aplikačního protokolu je ve všech těchto systémech řešena pomocí vyhledávání vzorů v přenášených datech. Tato činnost je bohužel výpočetně náročná a omezuje použití softwarových řešení na 100-300 Mb/s sítě. Propustnost klasifikačních metod založených na statistických vlastnostech je omezena zejména rychlostí sběru statistických údajů o tocích. Standardem v oblasti sběru statistických dat je technologie NetFlow, která je implementována i v několika volně dostupných nástrojích [1, 10]. Tato řešení jsou schopná pracovat na rychlostech blížících se 1 Gb/s [65].

Existují síťové aplikace, které musí činnost identifikace aplikačních protokolů řešit na vysokých rychlostech. Pro tyto účely vznikla celá řada hardwarově akcelerovaných řešení [23, 38, 5, 7], které lze použít na gigabitových sítích. Bohužel žádné z těchto zařízení nedosahuje dostatečné propustnosti pro 10 Gb/s sítě. Proto je nutné hledat nové hardwarové architektury pro identifikaci aplikačních protokolů.

Pro akceleraci kritických operací identifikace aplikačních protokolů lze s výhodou použít technologii FPGA (Field-programmable gate array). Vznikla řada prací, které s její pomocí akcelerují řadu síťových aplikací. Díky použití této technologie je možné pro tyto aplikace dosáhnout propustnosti 10 Gb/s a více. Cena vývoje nad touto platformou je však ve srovnání s klasickými ASIC technologiemi mnohonásobně nižší.

Tato práce se zabývá možnostmi hardwarové akcelerace identifikace protokolů s použitím technologie FPGA, univerzální akcelerační karty COMBO6x a platformy pro rychlý vývoj síťových aplikací NetCope. V první kapitole jsou shrnuty přístupy pro detekci protokolů na základě obsahu a jsou zde popsány metody pro rychlé vyhledávání vzorů pomocí FPGA. Druhá kapitola pojednává o problematice detekce protokolů pomocí statistických vlastností síťových toků. Ve třetí kapitole je shrnut aktuální stav v oblasti detekce aplikačních protokolů včetně nynějších problémů. Ve čtvrté kapitole je popsán model identifikace aplikačních protokolů a výsledky analýzy výpočetní náročnosti jednotlivých operací na obecných procesorech. V kapitole 5 je prezentována architektura sondy FlowMon, která byla rozšířena o identifikaci protokolů. V závěru jsou shrnuty výsledky a naznačeny další směry práce.

Kapitola 2

Rozpoznávání aplikačních protokolů na základě obsahu

2.1 L7-filter

L7-filter [49] je jedním z nejznámějších volně dostupných programů, který řeší klasifikaci provozu. Aplikace je primárně určena pro použití s QoS systémy např. tc [53]. Pro blokování nechtěných protokolů (NetFilter [54]) ji autoři spíše nedoporučují z důvodu stále relativně velkého množství false positives, jež by mohly přinést pro síť výrazné problémy. K této aplikaci existuje také celá řada webových konfiguračních rozhraní, která usnadňují její nasazení.

Aplikační protokoly jsou detekovány na základě vyhledávání regulárních výrazů v datech aplikační vrstvy. K aplikaci je poskytována otevřená databáze regulárních výrazů identifikující jednotlivé protokoly. V současné době databáze obsahuje okolo 100 různých protokolů zahrnující jak standardizované protokoly (http, ftp, ssl, sip, rdp...), tak řadu P2P aplikací (Ares, eDonkey, Kazaa, Gnutella...). K některým aplikačním protokolům existuje i více regulárních výrazů, z kterých může uživatel vybírat. Výrazy se liší především v přesnosti a také ve výpočetní náročnosti, která ovlivňuje už tak malou propustnost tohoto řešení. V následující tabulce 2.1 jsou zobrazeny regulární výrazy pro některé aplikační protokoly.

Aplikační protokol	Regulární výraz
ssh	ˆssh-[12]\.[0-9]
ftp	ˆ220[\x09-\x0d -]*ftp
pop3	ˆ(\+ok |-err)
sip	ˆ(invite|register|cancel) sip[\x09-\x0d -∼]*sip/[0-2]\.[0-9]

Tabulka 2.1: Vybrané regulární výrazy z databáze L7 filtru

L7 filter je implementován buď jako součást jádra operačního systému nebo ve formě uživatelské aplikace. Verze se liší zejména v použité knihovně pro vyhledávání regulárních výrazů. Kernel-Space varianta používá starší knihovnu (V8 regexp), která nepodporuje některé běžné syntaktické konstrukce (např. znakové třídy [[:PUNCT:]]) a není citlivá na velikost písmen. Userspace varianta nabízí větší volnost ve psaní regulárních výrazů, jelikož podporuje standardizovanou knihovnu regex.

Program za svého běhu vidí komunikaci v obou směrech a snaží se pomocí nadefinovaných regulárních výrazů najít tzv. Hello zprávy (např. 220 FTP SERVER READY, HTTP/1.1 200 OK). Tyto zprávy není nutné hledat po celou dobu komunikace, ale jen v průběhu několika prvních odeslaných paketů. Pokud se nepodaří do daného limitu nalézt některý z regulárních výrazů, tak se dále v daném spojení nic nevyhledává. Po úspěšné identifikaci provozu jsou další pakety spojení označovány uživatelem definovanou značkou, jež lze využít při psaní pravidel pro netfilter a tc.

Jednou z velkých nevýhod tohoto programu je nedostatečná propustnost, která nestačí ani pro 100Mb/s sítě. Samotní autoři programu uvádějí, že dosáhli propustnosti jen okolo 20 Mb/s, což je pro dnešní multigigabitové sítě nedostatečné. Dalším problémem tohoto řešení je paketová analýza, při které se neřeší problém fragmentace na L3 a L4 vrstvě. Vzory, které jsou rozděleny mezi více paketů tak nebudou detekovány. Velkým problémem pro některé aplikace je také nenulová míra false-positives.

2.1.1 IPP2P

Podobná volně dostupná aplikace jež lze použít stejným způsobem jako L7 filter se nazývá IPP2P [56]. Tento program se však soustředí pouze na detekci P2P aplikací. Bohužel signatury P2P aplikací jsou pevně zakódované do zdrojových souborů. To sice umožňuje větší volnost ve psaní signatur, ale přidávání signatur třetích stran již tak snadné není. Program totiž nenabízí standardizované softwarové rozhraní pro nové signatury.

2.1.2 HiPPIE - Hi-Performance Protocol Identification Engine

Tato pokročilejší volně dostupná aplikace [51] detekuje na 29 protokolů zahrnující standardizované a P2P aplikace. U této aplikace není použit princip detekce na základě regulárních výrazů, ale jsou zde implementovány pokročilé signatury umožňující detailně analyzovat protokol. Signatury mají tvar programu, který má na rozdíl od IPP2P standardizované rozhraní. Jelikož se provádí analýza protokolů, lze například odhalit, na jakém portu bude probíhat datová komunikace u FTP protokolu, což v případě použití regulárních výrazů nelze provést.

2.2 Detekce protokolů v IDS systémech

Systémy pro detekci narušení (IDS) potřebují pro správnou detekci některých útoků detailně analyzovat aplikační vrstvu. Většina IDS systémů včetně nejoblíbenějšího systému Snort [57] však stále k identifikaci aplikačního protokolu a následný výběr jednotky pro analýzu aplikačních dat využívá portů. IDS systém Bro [13, 50] je jako první rozšířen o detekci aplikačních protokolů s použitím signatur. Podle detekovaného protokolu se dynamicky určuje, která z jednotek pro analýzu aplikačních dat bude použita. Aplikační data tak nemusí být analyzována všemi jednotkami, ale stačí pouze jediná. Tím lze značně zvýšit propustnost IDS systému.

V systému Bro byly využity signatury z projektu L7-filter, které byly přepsány do formy, kterou akceptuje výkonná vyhledávací jednotka Bro. Systém Bro zavádí tzv. obousměrné signatury, které jsou velmi výhodné pro přesnější detekci klient-server komunikace. S výhodou je lze použít například pro identifikaci http odpovědi na http požadavek. Signaturu reprezentující odpověď serveru lze totiž podmínit výskytem signatury http požadavku viz obrázek 2.1.

```
signature http_server {                    # Server-side signature
  ip-proto == tcp                          # Examine TCP packets
  payload  /^HTTP\/[0-9]/                  # Look for server response
  tcp-state responder                      # Match responder-side of connection
  requires-reverse-signature http_client   # require client-side signature as well
  enable "http"                            # Enable analyzer upon match
}
signature http_server {                    # Client-side signature
  ip-proto == tcp                          # Examine TCP packets
  payload  /^[[:space:]]*GET[[:space:]]*/  # Look for requests [simplified]
  tcp-state originator                     # Match originator-side of connection
}
```

Obrázek 2.1: Obousměrná signatura pro identifikaci http provozu

Protokol aplikační vrstvy zpravidla nelze identifikovat na základě prvního paketu. Pokud je potřeba aplikační data dále analyzovat, je nutné přijatá data uchovávat alespoň do té doby, než bude identifikován protokol nebo bude dosaženo limitu vyrovnávací paměti. Vyrovnávací paměť pro každý síťový tok je v Bro nastavena na 4 Kb.

2.3 Přesnost metod

Přesnost detekce aplikačních protokolů závisí na hloubce prováděné analýzy. Existující řešení se liší v tom, zda provádějí analýzu nad toky nebo nad pakety, zda jen vyhledávají signatury v podobě regulární výrazů nebo analyzují aplikační protokol, zda zpracovávají celý tok nebo jen několik úvodních paketů. Moore [35] ve svém článku analyzoval, jaký vliv má hloubka analýzy aplikačních dat na přesnost identifikace aplikačních protokolů. Pro svou studii použil ručně klasifikovaný vzorek dat reprezentující týdenní provoz na velmi vytížené síti. Výsledky studie jsou shrnuty v následující tabulce 2.2.

Metoda identifikace aplikačního protokolu	Příklad aplikace	% Úspěšnost identifikace
Identifikace na základě portů	-	69.27 %
Jediný paket (signatura)	Červi/Viry	69.39 %
Jediný paket (analýza protokolů)	IDENT	69.62 %
První KByte (signatura)	P2P	71.48 %
První KByte (analýza protokolů)	SMTP	78.84 %
Celý tok (analýza vybraných protokolů)	FTP	98.78 %
Celý tok (analýza všech protokolů)	VNC, CVS	>99.99 %

Tabulka 2.2: Úspěšnost identifikace protokolů pro různou hloubku analýzy komunikace

Z tabulky vyplývá, že v případě použití identifikace na základě portů, existuje velké množství provozu, jež nelze tímto způsobem identifikovat. Postupným přidáváním dalších stupňů analýzy se lze dostat na 78 % správně identifikovaného provozu (analýza protokolu v prvním 1Kb dat). Dalšího výrazného zlepšení lze dosáhnout analýzou vybraných aplikačních protokolů v celém toku. Pouhou analýzou FTP provozu s extrakcí portů, na kterých probíhá ftp datová komunikace, lze přesnost zvýšit téměř na 100 procent. Samozřejmě 100 % přesnosti lze dosáhnout jen v ideálních podmínkách např. je-li k dispozici specifikace aplikačního protokolu a je-li přístup ke všem přenášeným datům. Toho lze však v reálné

síti docílit stěží. K některým protokolům totiž neexistuje specifikace a je potřeba ji odvodit reverzním inženýrstvím. Zachytit všechna data komunikace může být z důvodu směrování na Internetu nebo šifrování také problematické.

Jedním z problémů, které v současné době tíží většinu ISP jsou P2P sítě. Přesnost jejich detekce byla zkoumána v článku [43]. P2P sítě byly identifikovány na základě vyhledávání regulárních výrazů v prvních 10 paketech provozu. Pro řadu nejběžnějších P2P aplikací bylo dosaženo velmi dobré přesnosti. Míra false negatives (provoz, který nebyl správně určen jako P2P aplikace) byla velmi nízká a pohybovala se okolo 1-10 %, jak je možné vidět v tabulce 2.3.

Protokol	False Negatives
Gnutella	1.03 %
Kazza	4.7 %
DirectConnect	0 %
BitTorrent	9.89 %
eDonkey	1.60 %

Tabulka 2.3: Míra false negatives u vybraných P2P aplikací

Hloubka potřebné úrovně analýzy se protokol od protokolu liší. S rostoucí hloubkou analýzy roste přesnost, ale také rostou nároky na výslednou aplikaci a klesá propustnost. V praxi se proto často sahá ke kompromisu mezi přesností a nároky na aplikaci. U L7 filtru a IPP2P lze vidět, že byla použita jen paketová analýza s vyhledáváním vzorů, kdežto IDS systém Bro prováděl vyhledávání vzorů v tocích. Jako vhodný přístup, který nabízí velkou přesnost se jeví vyhledávání regulárních výrazů v tocích se zapojením několika jednotek pro analýzu některých vybraných protokolů (např. FTP).

2.4 Statistická klasifikace na základě aplikačních dat

Přístup ke klasifikaci síťového provozu popsaný v práci pana Haffnera [19] představuje mezistupeň mezi metodami založenými na vyhledávání vzorů v aplikačních datech a statistickými metodami pracujícími s informacemi o síťových tocích. Metoda pracuje nad aplikačními daty, ale místo vyhledávání vzorů zkoumá statistické vlastnosti dat pro jednotlivé aplikace. K dosažení vysoké přesnosti u této metody je potřeba mít k dispozici kvalitní trénovací množinu.

Z přeuspořádaného síťového toku je vybráno prvních $n = 256$ bajtů aplikačních dat, které jsou převedeny na bitové vektory v o délce $n * 256$. V podstatě se jedná o one-hot zakódování. Tyto vektory jsou spolu s ručně provedenou klasifikací použity jako trénovací množina pro klasifikátor. V práci byly použity tři lineární klasifikátory Naive Bayes, AdaBoost a Maximum Entropy z nichž nejlepších výsledků dosáhl algoritmus AdaBoost. Přesnost klasifikace u této metody je pro řadu protokolů uvedena v následující tabulce 2.4. Z tabulky lze vidět, že tato metoda dosahuje 99 % pro všechny zkoumané protokoly.

Prezentovaná metoda je velmi výhodná v případech, kdy ještě není známa signatura síťového provozu, protože může být bez jakékoliv analýzy odvozena z trénovací množiny. U šifrovaného provozu je schopna se přizpůsobit na data reprezentující výměnu klíčů. Nevýhodou této metody je nutnost mít k dispozici kvalitní trénovací množinu, kterou je

Aplikační protokol	Přesnost
ftp control	99,6 %
smtp	99,8 %
pop3	99,5 %
imap	100 %
https	99.2 %
http	99.0 %
ssh	100 %

Tabulka 2.4: Přesnost statistické klasifikace na základě aplikačních dat

potřeba pravidelně aktualizovat s příchodem nových verzí protokolů, klientských a serverových software.

2.5 Vyhledávání vzorů v FPGA

Všechny dosud představené přístupy k identifikaci protokolů používaly k detekci aplikačních protokolů vyhledávání řetězců nebo regulárních výrazů. Tato činnost je také potřebná ve všech IDS systémech, kde také byla zkoumána její výpočetní náročnost. Dle Markatos [32] tvoří vyhledávání řetězců a regulárních výrazů až 80 procent vynaloženého času procesoru. Tato skutečnost limituje propustnost softwarových IDS systémů na stovky Mb/s [42].

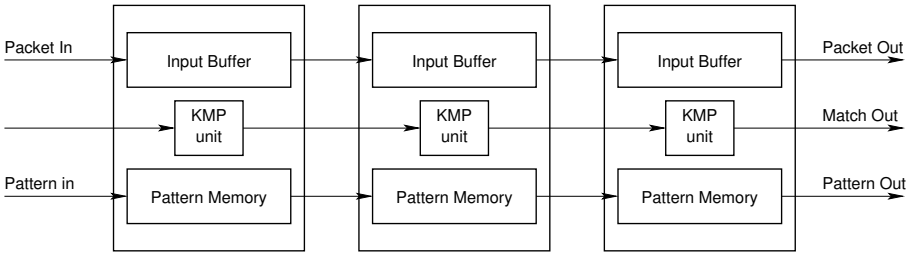
Z výše uvedených údajů vyplývá, že efektivnějším řešením úlohy vyhledávání vzorů lze dosáhnout vyšší propustnosti a výkonu celého systému. V oblasti vyhledávání vzorů byla provedena celá řada výzkumů a dnes je již známo několik desítek softwarových algoritmů zabývajících se touto problematikou. I ty nejefektivnější algoritmy pro vyhledávání vzorů však neposkytují dostatečný výkon. Hlavním důvodem je, že všechny tyto algoritmy jsou realizovány na obecném procesoru, který neposkytuje téměř žádné možnosti paralelního zpracování.

Další výzkum se tudíž soustředil na realizaci vyhledávání vzorů v hardware, který umožňuje široké možnosti paralelního zpracování. Pro tyto účely je možné použít speciální zařízení, jako je asociativní paměť nebo aplikačně specifický integrovaný obvod (ASIC). Tyto technologie sice poskytují vysoký výkon, ale za relativně vysokou cenu. Proto se většina výzkumů v posledních letech spíše zaměřila na technologii FPGA. V následující části této práce jsou shrnuty dosavadní přístupy pro rychlé vyhledávání vzorů v FPGA.

2.5.1 KMP algoritmus

Mezi velmi zajímavé přístupy pro vyhledávaní řetězců patří architektura [4] založená na bázi algoritmu KMP [27]. Tento algoritmus sice nepatří mezi nejrychlejší, ale jeho maximální složitost vyhledání je konstantní. Architektura využívá lineární pole vyhledávacích jednotek zachycené na obrázku 2.2. Každá vyhledávací jednotka se skládá ze 3 částí. Obsahuje paměť, do které je nahrán hledaný vzor ve formě předpočítané π tabulky. Dále obsahuje jednotku pro realizaci KMP algoritmu a vyrovnávací paměť, do které jsou postupně ukládána data příchozích paketů.

Nevýhodou KMP algoritmu je nutnost v některých případech provádět více porovnání pro jeden vstupní znak. Tento problém vyřešili tvůrci této architektury použitím dvou komparátorů. Díky tomu je možno zpracovávat až 2 vstupní znaky najednou a tím kompenzovat



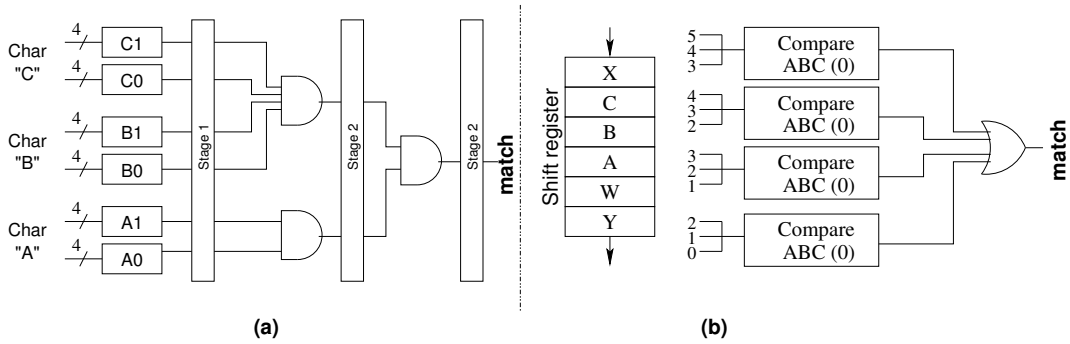
Obrázek 2.2: Architektura založená na KMP algoritmu

prodlevy, které vniknou při nutnosti provést více porovnání pro jeden vstupní znak. Tímto způsobem lze zajistit průměrné zpracování jednoho znaku v jednom taktu hodin.

Výhodou tohoto řešení je možnost aktualizace vyhledávaných vzorů bez nutnosti rekonfigurace FPGA. Přístup se vyznačuje také relativně nízkým využitím hardwarových zdrojů, protože řetězce jsou uloženy v paměti. Nevýhodou této architektury je podpora vyhledávání jen pro řetězce a nízká propustnost způsobená neschopností zpracovávat více znaků v jednom taktu hodin.

Zřetězené komparátory

Pro dosažení vyšších propustností při hledání řetězců byla navržena architektura založená na zřetězených komparátorech [46]. Základem tohoto přístupu jsou 4 bitové komparátory, hradla `AND` a registry. Vstupní znak je vždy porovnáván pomocí dvou komparátorů. Hledaný řetězec je pak složen z výstupů odpovídajících komparátorů pomocí operace `AND`. Za komparátory a členy `AND` jsou umístěny registry, které vytvářejí zřetězenou strukturu, díky níž je možné dosáhnout vysokých frekvencí. Schematicky je celá architektura znázorněna na obrázku 2.3a.



Obrázek 2.3: Architektura založená na zřetězených komparátorech

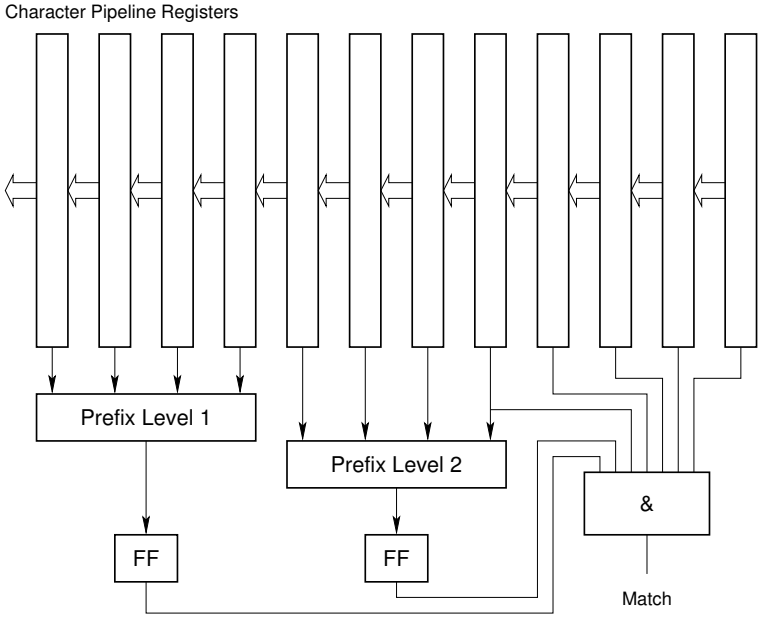
Aby architektura dosáhla vyšší propustnosti, je nutné zpracovávat více znaků v jednom hodinovém cyklu, jak je naznačeno na obrázku 2.3b. Zároveň je nutné zajistit, aby byl řetězec nalezen při libovolném posunutí, čehož lze dosáhnout použitím duplicitních komparátorů. Řetězec je nalezen, pokud je výstup alespoň jednoho komparátoru na vysoké logické úrovni.

Uvedená architektura je silně zřetězená a optimalizovaná pro použití s technologií FPGA, která v logické CLB buňce obsahuje 4-vstupé generátory logických funkcí (LUT) a registry.

Díky tomu lze s architekturou dosáhnout velmi vysokých frekvencí (až 287 MHz) a tedy i vysoké propustnosti. Bohužel díky své struktuře architektura spotřebuje velké množství zdrojů na čipu, nelze ji tedy použít pro vyhledávání velké množiny řetězců.

2.5.2 Architektura na bázi asociativní paměti

Další architektura je založena na využití FPGA jako optimalizované asociativní paměti pro vyhledávání řetězců [3, 2]. Tato relativně pokročilá architektura je zachycená na obrázku 2.4.



Obrázek 2.4: Architektura založená na bázi asociativní paměti

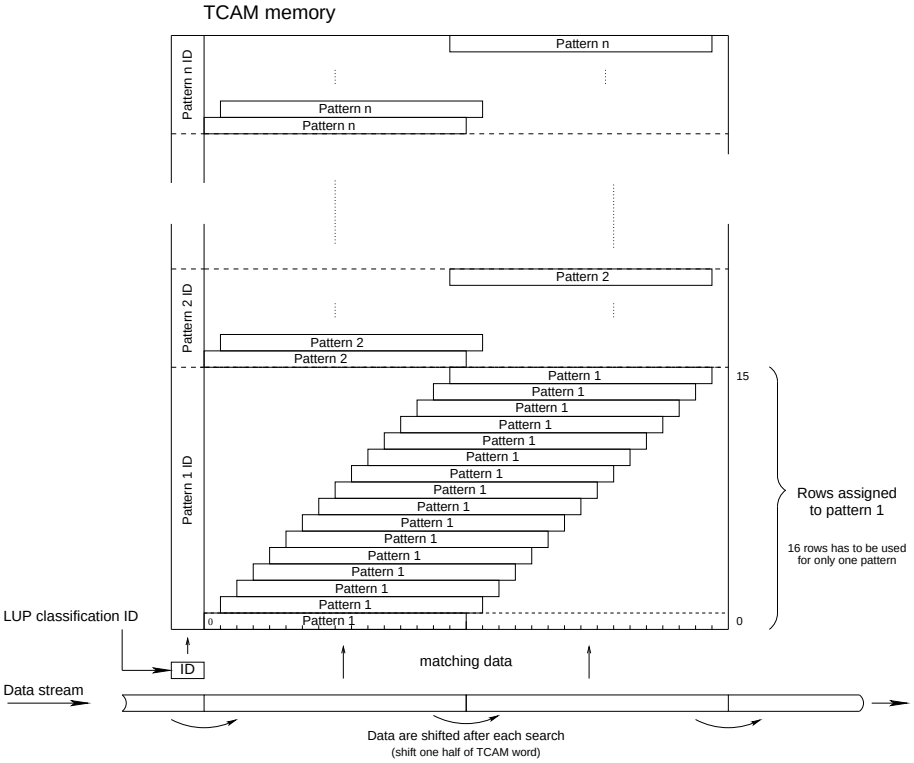
Do zřetězené struktury registrů, které zpožďují signály o jeden takt hodin, jsou přiváděny vstupní znaky převedené na kód 1 z 256. Z jednotlivých stupňů zřetězené struktury jsou přivedeny signály reprezentující jednotlivé znaky na vstup hradla `AND`. Hradlo spojuje jednotlivé znaky do podřetězců nebo celých řetězců. Díky překódování znaků jsou ušetřeny hardwarové zdroje, které by jinak byly použity pro realizaci komparátorů. Tento přístup využívá další optimalizace, z nichž nejdůležitější je možnost sdílet prefixy hledaných řetězců. Hledaná množina řetězců je rozdělena do skupin nebo do stromové struktury podle míry sdílení prefixů. Díky tomuto rozdělení je možné při sestavování řetězců pomocí logických `AND` členů maximálně sdílet hardwarové zdroje.

S výše popsanou architekturou lze díky možnosti zpracovávat více znaků najednou dosáhnout propustnosti až 10 Gb/s. Díky použitým optimalizacím umožňuje architektura vyhledávat tisíce vzorů. Nevýhodou této architektury je, že neumožňuje vyhledávat regulární výrazy.

2.5.3 Vyhledávání s použitím TCAM paměti

Pro projekt Scampi [41], který se zabývá monitorováním sítí na 1 Gb/s a 10 Gb/s, byla vyvinuta jednotka pro rychlé vyhledávání řetězců [60]. Tato architektura využívá rychlou asociativní paměť a FPGA. Samotné vyhledávání vzorů je realizováno přímo v asociativní

paměti. Rekonfigurovatelné pole je zde využito jen pro řízení TCAM paměti. Pro dosažení maximální propustnosti bylo navrženo uspořádání znázorněné na obrázku 2.5. Účelem tohoto uspořádání je možnost zpracovávat více znaků v jednom hodinovém taktu, čehož je dosaženo paralelním porovnáním řetězců pro všechny možné posunutí vůči zpracovávanému bloku dat.



Obrázek 2.5: Využití asociativní paměti pro vyhledávání vzorů

S takto navrženou architekturou bylo dosaženo propustnosti 3.2 Gb/s pro množinu 512 řetězců s maximální podporovanou délkou 15 znaků. Při použití modernějších pamětí s větší kapacitou a šířkou slova je s tímto přístupem možné dosáhnout propustnosti až 19,2 Gb/s s množinou 1024 řetězců.

Tato architektura nabízí vynikající poměr mezi propustností a velikostí podporované množiny vzorů. Mezi další výhody patří prakticky okamžitá změna množiny vyhledávaných vzorů bez nutnosti opakovat syntézu a provádět rekonfiguraci. Tato architektura rovněž neumožňuje vyhledávat regulární výrazy. Mezi její další nevýhody patří schopnost vyhledávat řetězce jen do určité délky, která je dána šířkou TCAM paměti.

2.5.4 Nedeterministické konečné automaty

Velmi zajímavé řešení pro vyhledávání vzorů v FPGA představují nedeterministické konečné automaty (NDKA). Architektury založené na tomto principu dosahují velmi dobrých výsledků s možností vyhledávat nejen řetězce, ale také regulární výrazy. Základní myšlenkou tohoto přístupu je transformace řetězce nebo regulárního výrazu na nedeterministický konečný automat, který je syntetizován do FPGA. Před podrobnějším popisem je vhodné formálně nadefinovat pojem regulární výraz a konečný automat:

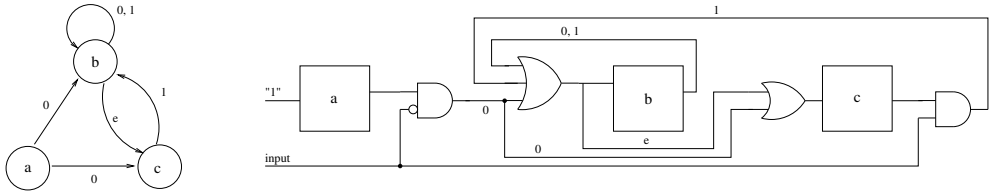
Definice: Regulární výraz nad abecedou Σ je definován:

- e a ϕ jsou regulární výrazy,
- každý symbol a, kde $a \in \Sigma$ je regulární výraz,
- jsou-li A_1, A_2 regulární výrazy na abecedou Σ, pak i $A_1 + A_2$, $A_1 \cdot A_2$ a A_1^* jsou regulární výrazy.

Definice: Nedeterministickým konečným automatem nad abecedou Σ rozumíme uspořádanou pětici $A = (Q, \Sigma, \delta, q_0, F)$, kde

- Q je konečná neprázdná množina stavů,
- Σ je neprázdná konečná množina vstupních symbolů,
- δ je zobrazení $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$, které se nazývá přechodová funkce,
- $q_0 \in Q$ je počáteční stav,
- F množina koncových stavů $F \subseteq Q$.

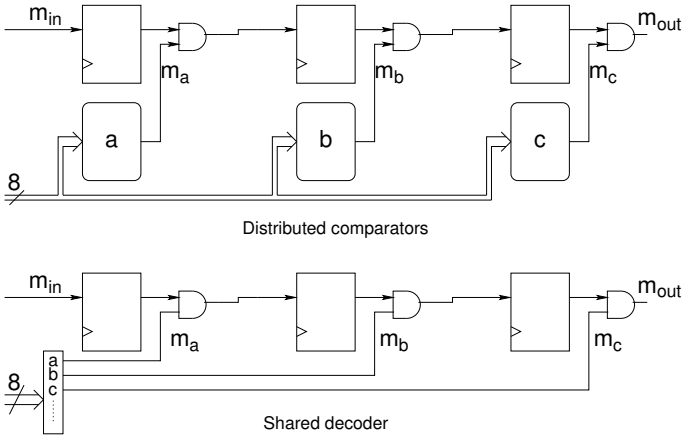
První práce [44] v této oblasti byla motivována rychlou konstrukcí NDKA a snadnou implementací v hradlovém poli. Tento přístup vychází z architektury DKA [18], kde je stav uložen v registru a přidružená kombinační logika vybírá následující stav. Přístup NDKA je ještě rozšířen o možnost realizovat e–přechody. Toto rozšíření je možné díky použití tzv. One-Hot kódovaní. To je realizováno tak, že každému stavu odpovídá jeden bit stavového registru. Tím je možné realizovat e–přechody pouhým propojením stavových registrů, jak je naznačeno na následujícím obrázku 2.9.



Obrázek 2.6: Implementace NDKA v FPGA

Uvedený přístup nelze použít pro vyhledávání velké množiny vzorů, protože je zde pro každý znak použit jeden komparátor. Bohužel díky tomu množství hardwarových zdrojů s každým dalším řetězcem narůstá. Z tohoto důvodu vylepšil Clark [8, 9] architekturu NDKA o několik optimalizací, z nichž nejvýznamnější je použití sdíleného dekodéru. Ten překóduje vstupní 8 bitové znaky na kód 1 z 256. Tento kód umožňuje snížit celkový počet komparátorů a tím i hardwarových zdrojů pro realizaci NDKA. Rozdíl mezi oběma přístupy je zachycen na obrázku 2.7.

Architektura využívající sdílený dekodér dosáhla s použitím nejnovějších FPGA Virtex II a Virtex II Pro, v závislosti na počtu paralelně zpracovávaných znaků a velikosti množiny řetězců, propustnosti až 7 Gb/s. S redukovanou množinou vzorů (250 znaků) je možno dosáhnout dokonce propustnosti 100 Gb/s.



Obrázek 2.7: Porovnání architektur s a bez použití sdíleného dekodéru

2.5.5 Deterministické konečné automaty - Bit-Split

Jedním z velmi zajímavých přístupů využívajících deterministických konečných automatů (DKA) je algoritmus Bit-Split [59]. Tento přístup se snaží efektivně mapovat DKA do specializovaného hardware s cílem maximálně ušetřit paměťové prostředky. Při transformaci vyhledávaných vzorů na DKA se totiž často využívá Aho-Corrasickův algoritmus, který přidává velké množství tzv. Fail přechodů. Pro realizaci takového automatu v paměti je nutné si uchovávat ke každému stavu 256 ukazatelů na následující stav.

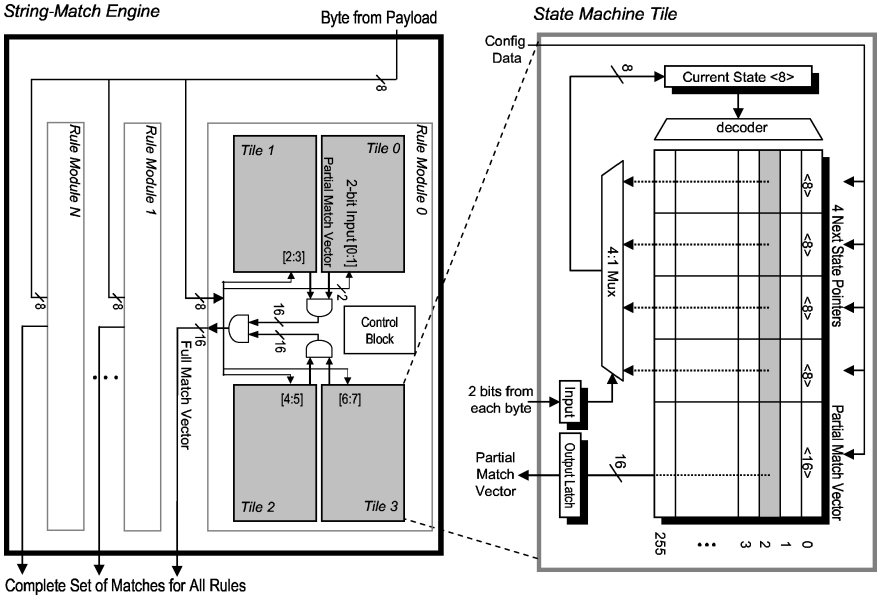
Technika Bit-Split dokáže počet těchto ukazatelů snížit minimálně na dva, a to tím způsobem, že pro každý bit vstupu konstruuje zvláštní automat. Tímto způsobem lze vytvořit 8 automatů, u kterých je potřeba ke každému stavu uchovávat pouze dva ukazatele, což značně šetří paměťové zdroje. K rozhodnutí zda byl konkrétní řetězec nalezen je potřeba, aby všechny automaty byly ve stavu, jež reprezentují přijetí řetězce.

Pro optimální využití hardwarových zdrojů se ukázalo efektivnější vytvořit čtyři automaty zpracovávající 2 bity vstupu. Výsledná architektura vyhledávací jednotky je znázorněna na následujícím obrázku 2.8. Vyhledávací jednotka se skládá z řady modulů. Každý modul obsahuje čtyři automaty přijímající dvou bitový vstup. Jeden modul může obsahovat automat o 256 stavech, který může vyhledávat až 16 řetězců najednou.

Technika Bit-Split dokáže až devět krát snížit celkové nároky na paměťové zdroje. Mezi výhody patří možnost modifikace vyhledávané množiny bez nutnosti rekonfigurace. Propustnost uvedeného řešení je omezena jen propustností paměti. Například pro technologii FPGA Virtex5 lze s integrovanými BlockRam paměťmi dosáhnout propustnosti okolo 4 Gb/s. Propustnost lze dále zvyšovat distribuováním zátěže mezi více paralelně umístěných vyhledávacích jednotek. Nevýhodou tohoto přístupu je omezené použití s technologií FPGA. FPGA čipy totiž obsahují omezené množství paměti, proto lze tímto přístupem vyhledávat jen omezené množství vzorů. Na FPGA čip Virtex4 FX100 lze do 376 BlockRam pamětí umístit až 1316 vzorů [25].

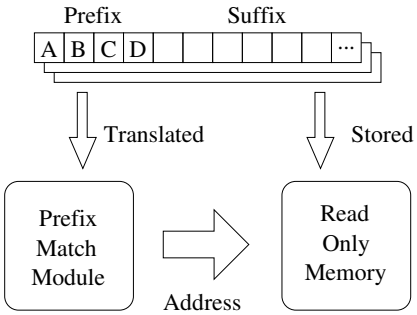
2.5.6 Speciální architektury

Mezi další architektury pro vyhledávání vzorů patří návrh využívající porovnání prefixů a následného dohledání zbytku řetězce v ROM paměti [6]. Tento přístup využívá rozdělení



Obrázek 2.8: Architektura Bit-Split

řetězců do několika jednotek, a to tak, aby nemohl nastat případ, že vyhledávaný prefix je součástí jiného hledaného prefixu. Pro prefixy je vygenerován vyhledávací modul na podobném principu jako u zřetězených komparátorů. Pokud je v řetězci nalezen odpovídající prefix, vyhledávací modul vygeneruje adresu do ROM paměti. Vzor je nalezen, pokud se z paměti vyčtený řetězec shoduje se zbytkem vstupních dat. Architektura dosahuje velmi dobrého poměru LUT/znak, a to zejména díky použití pamětí. Bohužel tento přístup nelze využít pro vyhledávání regulárních výrazů.



Obrázek 2.9: Architektura na bázi ROM paměti

Mezi velmi zajímavé přístupy patří využití Bloomových filtrů [11]. Tato architektura využívá rozptylovací funkci a je vhodná pro vyhledávání rozsáhlé množiny řetězců. Její nevýhodou je však možnost nesprávného označení řetězce, což může vyvolat falešný poplach. Tento přístup se však ukázal jako velmi vhodný pro implementaci předfiltrace provozu. Architektura [45] založená na tomto principu umožnila zredukovat tok 10 Gb/s na tok 1 Gb/s, který byl poté analyzován standardními metodami.

2.5.7 Srovnání jednotlivých přístupů

Výsledky prezentovaných architektur jsou shrnuty v následující tabulce 2.5. V prvním sloupci jsou uvedeni autoři se stručným názvem architektury. Další sloupce postupně znamenají použité hradlové pole, šířku vstupních dat v bitech, dosaženou frekvenci (f) a propustnost (P), velikost hledané množiny ve znacích, počet využitých generátorů logických funkcí (LUT) a průměrný počet generátorů logických funkcí na znak (LUT/znak).

Architektura	Použité FPGA	Data bit	f MHz	P Gb/s	Počet znaků	LUT	LUT/ znak
Baker et all. [4] KMP	Virtex2Pro-4	8	221	1,800	32	102	3,19
Sourdis et all. [46] komparátory	Virtex2-1000	32	344	11,008	489	8 132	16,6
	Virtex2-6000	32	252	8,064	2 457	47 686	19,4
Baker et all. [3] optimalizovaná asoc. paměť	Virtex2P-100	8	193	1,547	19 584	9 386	0,48
	Virtex2P-100	32	141	4,507	19 584	30 020	1,53
	Virtex2P-100	64	138	8,840	8 263	15 474	1,87
Clark et all. [9] NDKA	Virtex-1000	8	100	0,800	17 537	19 698	1,1
	Virtex2-8000	8	253	2,024	17 537	29 281	1,7
	Virtex2-8000	32	219	7,004	17 537	93 180	5,3
	Virtex2P-125	128	129	16,516	7 996	93 180	9,7
	Virtex2P-125	512	195	99,891	250	47 748	191
Tan et all. [9] Bit-Split	Virtex 4 FX 100	8	200	1,6	16 715	4 513	0,27
Cho et all. [6] ROM	Spartan3-2000	32	100	3,200	6 805	6 136	0,9
Lockwood et. all. Bloom filtr [11]	VirtexE-2000	8	63	0,502	420 000	32 640	0,08

Tabulka 2.5: Porovnání výkonů současných architektur

Z tabulky je patrné, že se současnými architekturami lze dosáhnout propustnosti až 10 Gb/s, ale za cenu velmi redukované množiny vzorů. Pro rozumně velkou množinu vzorů se však propustnosti pohybují mezi 4 Gb/s až 6 Gb/s. Nejlepších výsledků dosahuje přístup na bázi optimalizované asociativní paměti. Tento přístup však nelze využít pro vyhledávání regulárních výrazů. Pro identifikaci aplikačních protokolů není tento přístup vhodný, protože většina pravidel pro identifikaci obsahuje velké množství regulárních výrazů. Asi nejlepší architektura, která umožňuje vyhledávat řetězce i regulární výrazy je založena na použití NDKA. U tohoto přístupu je ale bohužel nutné počítat s rekonfigurací při změně vyhledávané množiny vzorů.

Kapitola 3

Rozpoznávání aplikačních protokolů pomocí chování

Problém rozpoznávání síťových aplikací podle chování lze zobecnit na problém rozpoznávání vzorů. K této problematice existuje veliký teoretický aparát a řada algoritmů. Popsat celou problematiku rozpoznávání vzorů je nad rámec této práce, proto zde bude uveden jen úvod do problematiky a budou popsány některé přístupy a algoritmy, které byly využity v oblasti identifikování síťových aplikačních protokolů. Detailní přehled problematiky rozpoznávání vzorů lze najít např. v knize Pattern classification [14].

Neformálně lze říci, že rozpoznávání vzorů je činnost, při které se ze surových dat (obrázek, zvuk) extrahují určité rysy, které pomohou určit do které kategorie (co je na obrázku) data spadají. Formálně lze činnosti rozpoznávání vzorů definovat následujícím způsobem: Mějme množinu vstupních dat $X = (x_1, x_2, x_3, ..., x_i)$. Každý prvek z množiny X lze charakterizovat pomocí n atributů $(a_{i1}, a_{i2}, a_{i3}, ..., a_{in})$. Nechť $C = (C_1, C_2, C_3, ..., C_m)$ je množina tříd. Cílem rozpoznávání vzorů je nalezení funkce $f : X \rightarrow C$, která mapuje vstupní data do jedné z předdefinovaných tříd.

Tuto definici můžeme analogicky použít k formální definici problému identifikace síťových aplikačních protokolů. Mějme množinu síťových toků $X = (f_1, f_2, f_3, ..., f_N)$, kde je každý ze síťových toků charakterizován n atributy $(a_{i1}, a_{i2}, a_{i3}, ..., a_{in})$. Rovněž mějme nadefinovanou množinu rozpoznávaných síťových aplikačních protokolů $C = (C_1, C_2, C_3, ..., C_m)$. Cílem identifikace je najít funkcí $f : X \rightarrow C$, která k jednotlivým tokům přiřadí právě jednu aplikaci.

Typické atributy síťového toku jsou např. průměrná délka paketu, průměrná délka spojení, velikost přenášených dat atp. Typické třídy síťových aplikací mohou zahrnovat jak konkrétní aplikační protokoly (http, ftp, SMTP, Kazza, Gnutella), tak třídy aplikací. Typické třídy jsou WWW, P2P, VoIP, Hry, Instant Messaging atp.

3.1 Získávání statistických dat o síťových tocích

Před začátkem samotného procesu klasifikace síťových aplikačních protokolů je třeba získat potřebné atributy síťových toků. To je možné provést pomocí na míru napsané aplikace nebo použitím existujících standardních řešení. Dnešním standardem v monitorování počítačové sítě na základě datových toků je technologie NetFlow, která byla vyvinuta firmou CISCO v roce 1996 [52]. Díky této technologii je možné získat agregované informace o probíhajících síťových komunikacích, které lze použít pro následnou klasifikaci provozu.

Technologie Netflow se skládá ze síťové sondy (exportér) a kolektoru. Exportér je softwarové nebo hardwarové zařízení, které v určitém bodě síťové infrastruktury sbírá informace o síťových tocích. Nasbírané informace jsou nejčastěji pomocí protokolu Netflow v5/9 posílány na kolektor, kde mohou být uchovány a později analyzovány. Mezi nejznámější softwarové sondy patří řešení nProbe [10]. Hardwarové sondy jsou většinou součástí Cisco zařízení a to hlavě směrovačů.

V současné době velká část Cisco zařízení podporuje export agregovaných informací pouze ve formátu NetFlow verze 5 [22]. Tento formát obsahuje fixní počet položek a nelze zde přidávat nové atributy. Obsažené atributy bohužel nedostačují k přesné statistické klasifikaci provozu. Nejnovější exportéry podporují vylepšený protokol NetFlow verze 9 [55], který je založen na systému zasílání šablon (templates), které popisují strukturu odesílaných dat. Šablony poskytují možnost snadného rozšíření NetFlow formátu o nové záznamy. Díky tomu lze do NetFlow formátu snadno přidat takové záznamy, které se hodí pro statistickou identifikaci provozu.

Pro měření a následný export některých speciálních atributů je nutná softwarová nebo hardwarová podpora na straně exportéru. Bohužel některé atributy často používané k činnosti identifikace protokolu nejsou v těchto zařízeních podporovány, proto je nutné se bez těchto atributů obejít nebo je vlastními silami doimplementovat.

3.2 Výběr atributů provozu

Jednou z nejdůležitějších částí procesu klasifikace síťových aplikací je výběr vhodných charakteristik (atributů) síťového provozu. Většina nynějších prácí zabývajících se klasifikací síťového provozu vybírá atributy spíše empirickým způsobem. Tento přístup je sice nenáročný, ale může znamenat opomenutí některých výrazně rozlišujících atributů.

Lepším řešením se může zdát vybrání všech atributů, které lze k danému problému získat. Pro klasifikaci síťového provozu bylo vytipováno celkem 248 vlastností [37], které zahrnují informace o paketech a mezipaketovém chování. Také jsou zde zahrnuty informace z transportní vrstvy, jako např. počet paketů s nastaveným SYN flagem. Tato obrovská množina však může pro daný klasifikační problém obsahovat množství nevýznamných a redundantních atributů, jejichž odstraněním lze nejen výrazně zlepšit přesnost klasifikace, ale také snížit výpočetní náročnost samotného klasifikačního algoritmu. Než budou zmíněny používané přístupy pro výběr atributů, je třeba nadefinovat pojem redundantního a nevýznamného atributu.

- Nevýznamný atribut nenese žádnou informaci o některé z tříd. Atribut nemé žádnou rozlišovací schopnost. Příkladem může být atribut, který nabývá jediné konstantní hodnoty.
- Atribut je redundantní pokud je ve vysoké korelaci s jiným atributem. Redundantní atributy často zhoršují přesnost a mají vliv na tzv. problém přeučení.

Pro výběr atributů existují dvě skupiny přístupů: filter a wrapper metody [29, 36]. Filter metody na základě předem poskytnutých trénovacích dat určují významnost jednotlivých atributů pro konkrétní klasifikační problém. Metrika pro významnost atributů může být např. stupeň korelace mezi atributy a třídou nebo míra separatibility tříd podle vybraného atributu. Wrapper metody jsou většinou založeny na iterativních algoritmech, které postupným přidáváním resp. odebíráním atributů hledají jejich optimální kombinaci pro použití s konkrétní klasifikační metodou (např. Naivní Bayesův klasifikátor).

K výběru atributů je možné naimplementovat některou ze známých metod nebo použít již implementované a velmi kvalitní řešení s názvem Weka [58]. Tento nástroj implementuje celou řadu metod pro výběr atributů, ale také nabízí implementaci řady nejznámějších klasifikačních a shlukovacích metod. Nástroj byl použit také při výběru vhodných atributů pro klasifikaci síťových aplikačních protokolů [36], kde zlepšil přesnost klasifikace o 10 %.

3.3 Klasifikační metody používané pro identifikaci provozu

3.3.1 Naivní Bayesův klasifikátor

Tato metoda [20] patří do rodiny pravděpodobnostních klasifikátorů. Vstupem této metody je vektor extrahovaných rysů. Na základě trénovací množiny je vstupnímu vektoru přiřazen stupeň příslušnosti pro jednotlivé třídy. Jako výsledek je vybraná třída pro níž je stupeň příslušnosti vstupního vektoru nejvyšší.

Tato klasifikační metoda vychází z Bayesova teorémů 3.1, což je jeden ze základních teorému teorie pravděpodobnosti. Tento teorém říká, že pravděpodobnost jevu H (např. síťový tok je P2P provoz) podmíněneho $\mathbf{X}$ (např. hodnota průměrné mezipaketové mezery rovné a) se rovná následujícímu zlomku. Jmenovatel zlomku představuje pravděpodobnost, že nastane jev $\mathbf{X}$ (např. pravděpodobnost s jakou se mezipaketová mezera rovná a). Hodnota $P(\mathbf{X} \mid H)$ v čitateli představuje pravděpodobnost, že nastane jev $\mathbf{X}$ podmíněný H (např. pravděpodobnost, že se mezipaketová mezera rovná konstantě a, pokud je provoz P2P komunikace). Hodnota $P(H)$ reprezentuje pravděpodobnost jevu H, což si lze představit jako pravděpodobnost s jakou se vyskytují určité druhy síťového provozu.

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})} \tag{3.1}$$

Problém výše uvedeného vzorce spočívá v tom, jak odhadnou pravděpodobnosti jednotlivých jevů $P(H)$, $P(\mathbf{X} \mid H)$, $P(\mathbf{X})$. Tento problém lze řešit poskytnutím trénovací množiny, ze které lze tyto pravděpodobnosti odhadnout. Na této myšlence také staví naivní Bayesův klasifikátor, jehož princip bude popsán dále.

Mějme množinu tříd C_1, C_2,...,C_m. Mějme trénovací množinu D, jež je tvořena n-dimenzionálními vektory atributů a přiřazením do jedné z tříd C_i. Pro vstupní vektor $\mathbf{X} = (x_1, x_2, ..., x_n)$ se snažíme najít třídu C_i kde je $P(C_i|\mathbf{X})$ největší. Snažíme se tedy maximalizovat $P(C_i|\mathbf{X})$.

Bayesův vzorec lze zjednodušit odstraněním jmenovatele, neboť je konstantní pro všechny třídy. Pokud je pravděpodobnost výskytu všech tříd stejná, lze získat zjednodušený Bayesův vztah $P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)$. Pokud jsou jednotlivé atributy vektoru $\mathbf{X}$ na sobě nezávislé, lze získat zjednodušený vztah pro pravděpodobnost příslušnosti vektoru $\mathbf{X}$ k třídě C_i viz 3.2.

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) \tag{3.2}$$

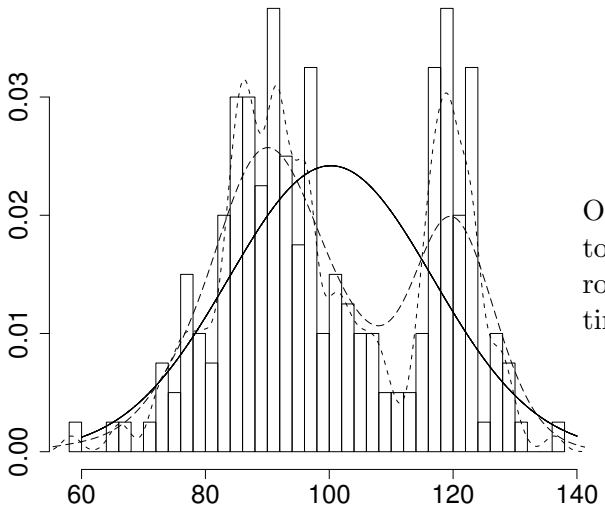
Výsledný vztah, který jednoznačně přiřazuje vektoru $\mathbf{X}$ jedinou třídu C_i je uveden dále (viz 3.3).

$$bayes(x_1, x_2, ..., x_n) = argmax_c P(C = c) \prod_{k=1}^{n} P(X_k = x_k \mid C = c) \tag{3.3}$$

První práce [63, 36, 24] zabývající se klasifikací internetového provozu na základě statistických vlastností úspěšně využívá naivní Bayesův klasifikátor pro přiřazení provozu do jedné z předdefinovaných tříd (interaktivní, email, WWW, hry, multimédia, P2P atp). Pro natrénování klasifikátoru používá trénovací množinu skládající se z 65,000 klasifikovaných toků, ze kterých extrahuje 218 diskriminantů [37]. Pomocí algoritmu FCFB je množina diskriminantů redukována se zachováním vysoké rozlišovací schopnosti.

Vzhledem k tomu, že je většina parametrů spojitá, je třeba aproximovat funkci hustoty pravděpodobnosti pro každý parametr a třídu z trénovacích dat. K tomu lze použít například aproximaci normálním rozložením 3.4 nebo dokonalejší aproximací zvanou Kernel estimation. Rozdíl mezi aproximacemi je vidět v následujícím grafu 3.1.

$$P(x_k|C_i) = \frac{1}{\sqrt{2\pi\sigma_{C_i}}} e^{-\frac{(x_k-\mu_{C_i})^2}{2\sigma^2_{C_i}}} \tag{3.4}$$



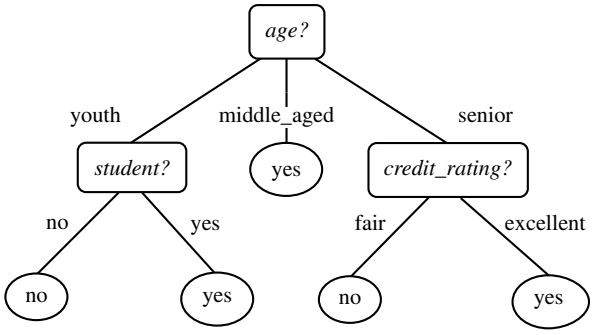
Obrázek 3.1: Srovnání aproximace funkce hustoty pravděpodobnosti pomocí normálního rozložení (silná čára) a algoritmu Kernel estimation (čarkovaná čára)

Naivní Bayesův klasifikátor ve spojení s redukcí diskriminantů a algoritmem Kernel Estimation dosáhl zhruba 93% přesnosti pro úlohu identifikace síťového provozu, což řadí naivní Bayesův klasifikátor mezi jednu z nejlepších metod pro identifikaci provozu.

3.3.2 Rozhodovací stromy

Jedním z dalších přístupů využitých k identifikaci síťových protokolů jsou rozhodovací stromy [20]. Metoda vytvoří na základě předem poskytnutých trénovacích dat stromovou strukturu. Každý vnitřní uzel stromu představuje test na jeden z atributů vstupních dat. Každá větev pak reprezentuje výsledek tohoto testu. Listové uzly stromové struktury reprezentují výsledek klasifikace tj. jednotlivé třídy C_i. Proces klasifikace je tedy založen na procházení stromu dokud se nedojde k listovému uzlu. Ukázka rozhodovacího stromu reprezentujícího rozhodnutí zákazníka při koupi určitého zboží je naznačena na obrázku 3.2.

Konstrukce rozhodovacího stromu je založena na postupném výběru atributů a následném vytvoření dvou (binární rozhodovací stromy) a více větví. Nejprve jsou vybírány atributy, které co nejlépe rozdělí trénovací množinu. To znamená takové atributy, podle kterých se v ideálním případě podaří rozdělit trénovací množinu na více podmnožin patřících do jedné třídy. Pro výběr atributů se používají nejčastěji tři metriky. V algoritmu ID3 je používaná metrika information gain (IG) založená na entropii informací. Metrika IG má tu



Obrázek 3.2: Rozhodovací strom

nevýhodu, že preferuje výběr atributů, které mají velký počet hodnot. Proto byla navržena nová metrika gain ratio (GR), která se používá v algoritmu C4.5. V algoritmu CART je používán gini index(GI), který umožňuje pouze konstrukci binárních stromů.

Jakmile je rozhodovací strom vytvořen, mnoho větví často reflektuje anomálie v trénovací množině (šum, osamocené vrcholy). Proto se často používá technika prořezávání (pruning), která tento problém, často označovaný jako přeučení (overfitting), řeší. Obecně se používají dva přístupy pro prořezávání stromů. Preprunning je technika, která zastaví vytváření některých větví v době konstrukce stromu. Technika postprunning naopak odstraňuje některé větve už z vytvořených rozhodovacích stromů.

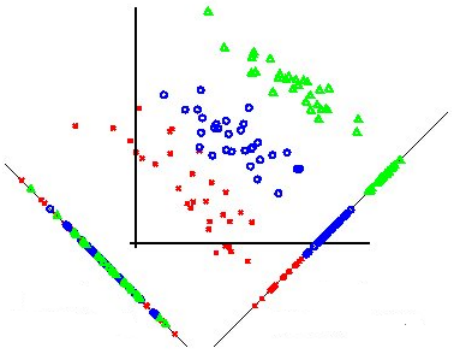
Rozhodovací stromy byly pro klasifikaci síťového provozu poprvé použity v [15] pro rozpoznávání základních protokolů jako HTTP, FTP, Telnet, SMTP a SSH. Vstupem do použitého algoritmu C5.0 pro konstrukci rozhodovacího stromu byly zvoleny následující atributy síťových toků: TPC flags, průměrná mezipaketová mezera a průměrná délka paketů. Vyjmenované veličiny nebyly měřeny v rámci celého síťového toku, ale jen v rámci pohyblivého okénka o velikosti n paketů. Empiricky zjištěná přesnost tohoto řešení pro různé velikosti parametru n je zobrazena v tabulce 3.1. Z tabulky je vidět, že přesnost tohoto řešení se pro různé aplikace pohybuje okolo 82-100%. Poměrně důležitý fakt je, že přesnost metody příliš nezávisí na počtu zpracovaných paketů toků. Přesných výsledkůů lze dosáhnout vypočtením požadovaných statistických údajů už nad 10 pakety daného toku.

Velikost okna	FTP	SSH	Telnet	SMTP	WWW
1000	100%	88%	94%	82%	100%
200	98%	96%	96%	84%	98%
100	100%	96%	96%	86%	100%
50	98%	96%	96%	82%	100%
20	100%	98%	98%	82%	98%
10	100%	100%	100%	82%	98%

Tabulka 3.1: Přesnost klasifikace pomocí rozhodovacích stromů pro různé velikosti okénka

3.3.3 Lineární diskriminační analýza a k-nejbližší soused

Metoda Linear Discriminant analysis (LDA) [40] používaná ve strojovém učení (lineární klasifikátor) je založena na hledání takové lineární kombinace $Z = a^T x$, která maximalizuje mezitřídní rozptyl (interclass variance) a zároveň minimalizuje rozptyl hodnot uvnitř stejné třídy. Nalezení této lineární kombinace znamená redukci vícedimenzionálního problému na jednodimenzionální, ve kterém je klasifikace přímočará. Na následujícím obrázku 3.3 je vidět, jak lze nejlépe a nejhůře redukovat konkrétní 2D problém do 1D prostoru.



Obrázek 3.3: Nejhorší a nejlepší redukce vícedimenzionálního problému pomocí LDA

Metoda k-nejbližší soused (k-NN) [20] přiřazuje prvku stejnou třídu, jakou má nejbližší prvek z poskytnuté trénovací množiny. Pro určení vzdálenosti je používaná Euklidovská vzdálenost 3.5. Pro větší přesnost této metody bývají hodnoty vstupu normalizovány podle vzorce 3.6. Parametr k v názvu metody určuje kolik nejbližších sousedů z trénovací množiny bude použito k určení výsledné třídy (vybere se podle většiny).

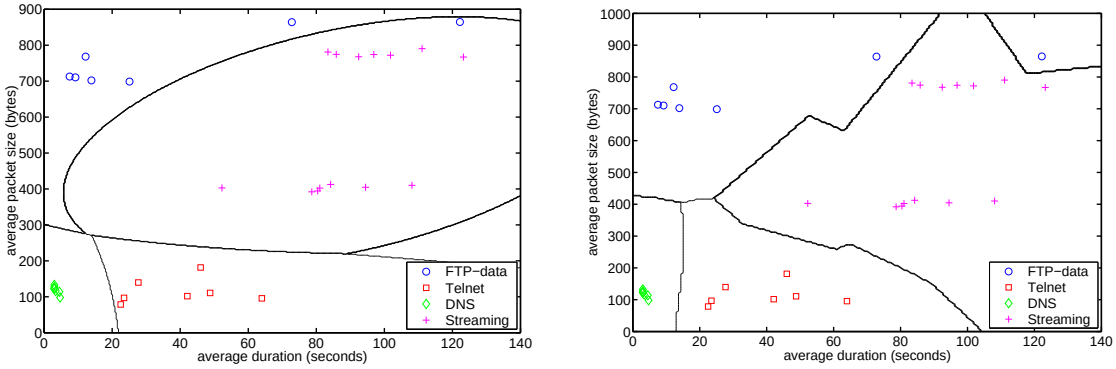
$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \tag{3.5}$$

$$v' = \frac{v - min_A}{max_A - min_A} \tag{3.6}$$

Klasifikační metody LDA a k-NN byly použity v práci [39] zabývající se kvalitou služeb k mapování provozu do jedné z mnoha tříd (interaktivní, streaming, bulk přenosy...). Jako statistické atributy byly zvoleny průměrná velikost paketu a doba trvání síťového toku. Na následujícím obrázku 3.4 jsou zobrazeny prvky trénovací množiny podle těchto atributů ve 2D grafu. Z grafu je vidět, že LDA dokáže třídy rozdělit oproti k-NN hladkou křivkou. Dosažená přesnost uvedena v této práci byla 100% pro LDA a 90.5%, 97,6%, 100% pro 1-NN, 3-NN a 5-NN.

3.3.4 Shlukovací techniky

Zatím byly popsány klasifikační přístupy využívající učení s učitelem. Rovněž existují přístupy založené na učení bez učitele. Tyto metody jsou schopny data podobných vlastností přiřadit do stejných skupin (shluků). K přiřazení výsledné třídy (aplikační protokol) k vzniklému shluku postačí klasifikovat jeden vzorek dat z příslušného shluku. Výhodou



Obrázek 3.4: Klasifikace pomocí LDA a 1-NN

těchto metod je, že nepotřebují kvalitně klasifikovanou trénovací množinu, která se zpravidla velmi těžce získává. Rovněž jsou schopny objevit shluky, které mohou reprezentovat nový typ aplikačního protokolu vyskytujícího se na síti.

Existuje celá řada shlukovacích algoritmů. Pro klasifikaci síťového provozu však byly použity pouze následující algoritmy: Em, DBSCAN a K-Means [34, 16, 17]. Nejrychlejším a také nejjednodušším shlukovacím algoritmem, který bude popsán v následující části je právě K-means.

Algoritmus K-means rozděluje prvky vstupní množiny do K disjunktních podmnožin. Pro každý shluk maximalizuje algoritmus homogenitu ve skupině tím, že minimalizuje kvadratickou chybu. Vzorec pro výpočet kvadratické chyby je uveden dále viz 3.7. Funkce *dist* ve vzorci představuje Euklidovskou vzdálenost (viz 3.5) mezi každým prvkem x a středem c jednotlivých shluků.

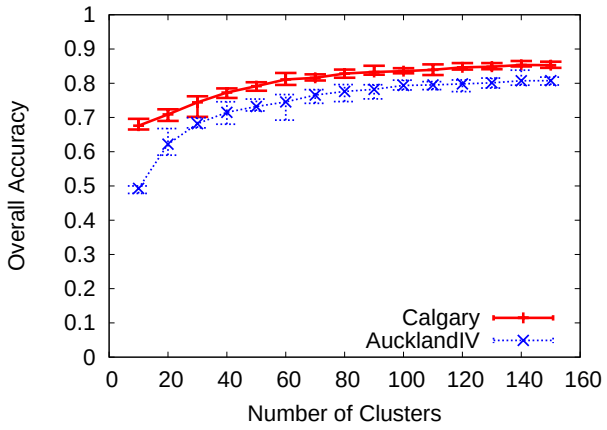
$$E = \sum_{i=1}^{K} \sum_{j=1}^{n} |dist(x_j, c_j)|^2 \tag{3.7}$$

Kvadratická chyba E je v K-Means minimalizována pomocí následujícího algoritmu. Středy jednotlivých K shluků jsou na začátku zvoleny náhodně. Data ze vstupní množiny jsou přiřazena vždy k nejbližšímu shluku. Algoritmus K-means iterativně přepočítává střed jednotlivých shluků do té doby, než se chyba E stabilizuje.

Bohužel shlukovací metody nedosahují takové přesnosti jako metody založené na učení s učitelem. V pracích využívajících shlukovací techniky [34, 16] se dosažená přesnost identifikace protokolů pohybuje mezi 80-90%. V grafu 3.5 je zobrazena závislost přesnosti na počtu shluků pro představený K-means algoritmus.

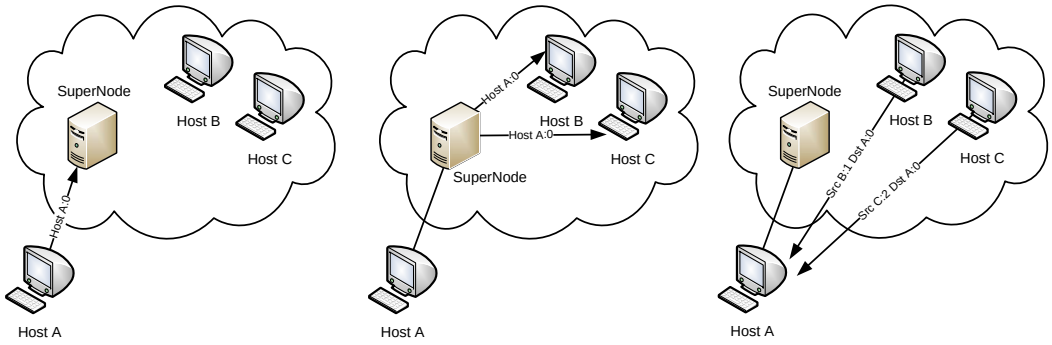
3.4 Identifikace s využitím agregovaných informací o síťových tocích

Dosud představené metody se snažily identifikovat aplikační protokoly jen z jednotlivých síťových toků. Pro identifikaci některých aplikačních protokolů je však vhodné zkoumat komunikaci na vyšší úrovni tj. dívat se na jednotlivé uzly v síti a zkoumat počet odchozích a příchozích spojení, typ navázaných spojení, počet odeslaných paketů a další. Tyto informace mohou být klíčové jak pro detekci některých útoků (reconnaissance, DOS, DDOS), tak pro detekci P2P sítí.



Obrázek 3.5: Přesnost K-means algoritmů pro klasifikaci síťového provozu

Před samotným připojením k P2P síti se musí klient přihlásit u jednoho z tzv. supernode, jehož IP adresu má uloženou ve své cache. Klient zašle informaci o portu, na kterém očekává spojení od ostatních P2P klientů. Supernode musí tuto informaci propagovat dále do sítě k ostatním klientům. Ostatní klienti mohou následně komunikovat s nově připojenou stanicí. Celý proces přihlášení je vidět na obrázku 3.6.



Obrázek 3.6: Přihlášení a komunikace v P2P síti

Karagiannis [26] si u P2P sítí všiml určitých vlastností v komunikaci, které použil pro jejich spolehlivou identifikaci. První vlastností, která zpravidla u P2P sítí platí je, že pokud dvě IP adresy mezi sebou komunikují pomocí TCP a UDP, tak se s vysokou pravděpodobností jedná o P2P komunikaci. Je ovšem třeba eliminovat komunikace typu DNS, NETBIOS, IRC, hry a streaming, které vykazují podobný charakter. Dále pokud ke dvojicím {IP, Port} hledáme počet navázaných spojení s rozdílnou IP adresou a počet navázaných spojení s rozdílným portem, tak u P2P komunikace platí, že se obě tyto statistiky rovnají. Pro dvojici {A,0} z obrázku 3.6 lze vidět, že daná vlastnost platí. Přesnost detekce P2P se při kombinaci těchto vlastností pohybovala okolo 95 procent.

Kapitola 4

Zhodnocení aktuálního stavu

V současné době existuje řada metod pro identifikaci aplikačních protokolů. V kapitole 2 a 3 byly představeny metody identifikující aplikační protokoly na základě portů, vyhledávání regulárních výrazů, statistických a agregovaných statistických vlastností a kompletní analýzy přenášených dat oproti specifikaci protokolů.

Ideální metoda identifikace aplikačních protokolů musí splňovat řadu parametrů. Metoda musí být vhodná pro identifikaci jak standardizovaných, tak uzavřených firemních formátů. Musí byt schopna identifikovat aplikační protokol jak v šifrovaných tak v otevřených datech. Dále musí být pro všechny typy protokolů dostatečně rychlá a přesná.

Žádná z analyzovaných metod však tyto vlastnosti nesplňuje. Každá metoda je vhodná k identifikaci jen určitých druhů aplikačních protokolů. Metody se dále liší v propustnosti, celkové přesnosti, rychlosti detekce. Pro zlepšení nejen těchto parametrů je potřeba kombinovat jednotlivé metody a maximalizovat jejich nejlepší vlastnosti. Tímto způsobem se lze přiblížit k ideální metodě identifikace aplikačních protokolů. Vhodnost jednotlivých metod pro dosažení určitých parametrů je znázorněna v následující tabulce.

Metoda	Rychlost detekce	Propustnost	Šifrovaný provoz	Firemní standard	Přesnost P2P	Přesnost stand.
Porty	☺	☺	-	-	☹	😐
Signatury	☺	☹	☹	😐	😐	☺
Statistiky	☹	☺	☺	☺	☺	😐
Agregované statistiky	☹	☺	☺	☺	☺	☹
Úplná analýza	☺	☹	☹	☹	😐	☺

Tabulka 4.1: Zhodnocení metod identifikace aplikačních protokolů

Kombinace výše uvedených metod přináší nejen vyšší přesnost, ale také vyšší nároky na výpočetní zdroje, protože je potřeba provádět řadu výpočetně náročných operací. Pokud tyto operace mají být prováděny nad daty přicházejícími na gigabitových rychlostech, lze se dostat do výrazných problémů s propustností takto vytvořeného systému. Jelikož ještě nikde nebyly analyzovány limity těchto operací, další práce se bude věnovat určení těchto limitů. Budou identifikovány operace u nichž je potřebná hardwarová akcelerace a bude provedeno mapování jednotlivých operací mezi hardware a software s cílem akcelerovat nynější nejlepší metody identifikace protokolů pro použití na 10 Gb/s sítích.

Kapitola 5

Model identifikace protokolů

Na základě analýzy existujících metod popsaných v kapitole 2 a 3 byly vybrány operace identifikace aplikačních protokolů, které byly využity k vytvoření obecného modelu identifikace aplikačních protokolů. Ve vytvořeném modelu jsou obsaženy všechny operace, které potřebují nynější nejlepší metody identifikace aplikačních protokolů. Model byl sestaven z důvodu analýzy výpočetní náročnosti jednotlivých operací na obecném procesoru. Tato analýza byla následně použita k mapování modelu na hardware a software s cílem dosáhnout dostatečné propustnosti pro 10 Gb/s sítě.

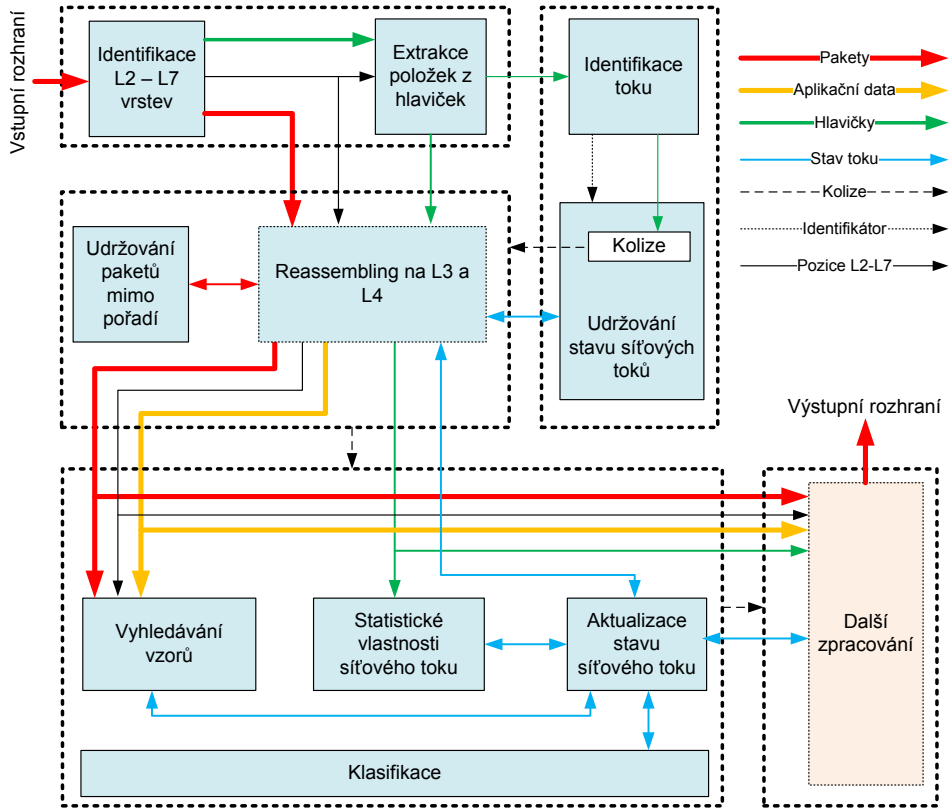
Vytvořený model se skládá z 5 navzájem závislých operací. Tyto operace řeší zpracování nižších vrstev protokolů, udržování stavových informací k síťovým tokům, skládání paketů/toků na L3 a L4 vrstvě, identifikaci síťového provozu a libovolnou uživatelskou aplikaci. Jednotlivé operace budou popsány v dalším textu.

Pakety získané ze vstupního rozhraní jsou předány operaci, která zajišťuje zpracování nižších vrstev protokolů. Tato operace vykonává dvě základní činnosti. Jednak je zodpovědná za určení offsetu a typu protokolů na jednotlivých vrstvách ISO/OSI. Dále také řeší extrakci vybraných položek z hlaviček protokolů. Pakety, extrahované hlavičky a informace o typech protokolů a jejich pozici jsou předány zavislým operacím k dalšímu zpracování.

Velmi důležitou činností, která musí být v rámci této aplikace řešena je udržování stavových informací k síťovým tokům, resp. ke spojením. Ke každému toku je potřeba udržovat řadu informací jako stav vyhledávání, statistické údaje, očekávané sekvenční číslo v tcp spojení, výsledek identifikace a další. Pro identifikaci aplikačních protokolů je vhodné uchovávat jak stavové informace ke spojením, tak k jednotlivým tokům. Informace ke spojení jsou vhodné vzhledem k charakteristice většiny aplikačních protokolů jako obousměrná komunikace (určení provozu jako HTTP, pokud byl nalezen vzor pro HTTP požadavek i HTTP odpověď). Informace k jednotlivým tokům je vhodné udržovat z důvodu různého statistického charakteru komunikace v různých směrech.

Další velmi důležitá operace řeší fragmentaci paketů na L3 a L4 vrstvě, která vzniká vlivem omezené MTU a zasíláním dlouhých zpráv aplikační vrsvou. Tato operace poskytuje přístup k nefragmentovanému TCP toku. TCP pakety, které přichází mimo pořadí jsou ukládány do vyrovnávací paměti a při vyplnění děr v TCP toku jsou následně vyčítány a obsah aplikační vrstvy je předán k dalšímu zpracování. Pakety, včetně těch které přišly mimo pořadí, jsou okamžitě předány k dalšímu zpracování. Jejich předávání v pořadí TCP toku může vést k velkým zpožděním - paket na který se čeká mohl vlivem asymetrického směrování již dorazit k cíli.

Jádrem modelu je operace identifikace aplikačních protokolů. Tato operace využívá přístupy k identifikaci založené na vyhledávání vzorů a získávání statistických vlastností



Obrázek 5.1: Model identifikace protokolů

o síťovém toku. Informace o chování toku a o nalezených vzorech v aplikačních datech jsou použity pro určení typu aplikačního protokolu v klasifikátoru. Typ aplikačního protokolu spolu s pakety, extrahovanými hlavičkami a dalšími informacemi může být předán k dalšímu zpracování. Typické aplikace, které tyto informace potřebují jsou aplikační firewally, IDS/IPS systémy, systémy pro vyvažovaní zátěže, traffic shaping a další.

5.1 Možnosti a limity identifikace aplikačních protokolů

V této sekci budou detailně rozebrány jednotlivé části modelu identifikace aplikačních protokolů. Bude popsáno, které operace nad jakými daty je potřeba provádět. Dále budou shrnuty výsledky simulací těchto činností na standardních počítačových architekturách. Tato analýza byla vytvořena z důvodu identifikace časově kritických částí algoritmu, které budou následně mapovány na hardware s cílem dosáhnout propustnosti pro 10 Gb/s sítě.

5.1.1 Příjem paketů ze síťového rozhraní

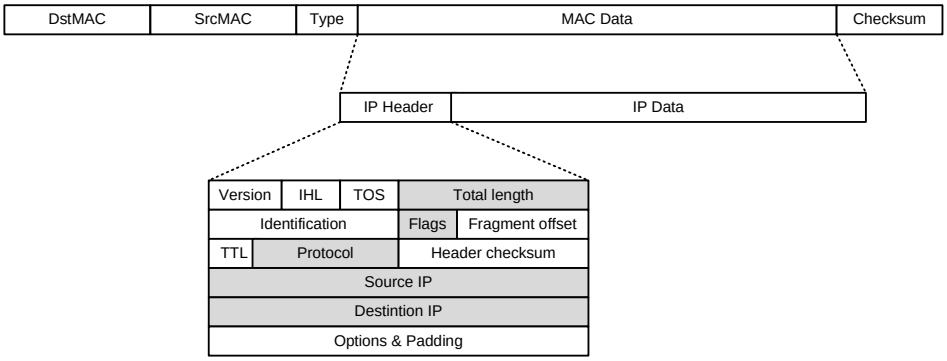
Před analýzou výpočetní náročnosti dílčích činností nad příchozím paketem je potřeba prozkoumat možnosti síťového rozhraní na běžných počítačových architekturách. Důležitým parametrem těchto rozhraní je množství dat (paketů), které jsou schopny přenést do hlavní paměti. Tato propustnost ovšem silně závisí na konkrétní architektuře síťové karty a jejím DMA řadiči. V práci [33], která se mimo jiné zabývá také architekturou síťové karty jsou

popsány limity propustnosti mezi síťovou kartou a hlavní pamětí. Při využití standardního rozhraní linuxového jádra je síťová karta schopna přenést až 10 miliónů nejkratších paketů. Při agregaci paketů do 4 Kb bloků je možné teoreticky přenést 327 Gbps. Z výše uvedených údajů je patrné, že samotný příjem paketů nepředstavuje v současných počítačových architekturách zásadní problém. Jeho propustnost je omezena zejména použitou sběrnicí. Současné sběrnice (např. PCIe 8x) však poskytují dostatečnou propustnost pro použití na 10 Gb/s sítích. Problém s propustností není tedy v příjmu, ale v dalším zpracování, kdy procesor nemusí mít dostatečný výkon ke zpracování takto velkého množství dat.

V souvislosti s příjmem paketů je nutné se zabývat také latencí, tj. dobou mezi příchodem a následným odesláním zpracovávaného paketu. Latence je způsobena jednak samotným zpracováním paketů, ale také přenosy mezi síťovou kartou a hlavní pamětí. U PCIe se latence pohybuje okolo 1 ms. U síťových aplikací zabývajících se monitorováním provozu nepředstavuje latence zásadní problém. Naproti tomu u síťových aplikací, které nějak do provozu zasahují se snažíme latenci minimalizovat.

5.1.2 Analýza protokolů na L2-L4 vrstvě

Struktury komunikačních protokolů jsou zpravidla organizovány do několika vrstev podle modelu ISO/OSI. Každá vrstva má přesně definovanou strukturu, která je obvykle tvořena hlavičkou a datovou částí. Výsledný komunikační protokol je následně sestaven z jednotlivých vrstev s využitím principu zapouzdření. Pro další činnosti identifikace aplikačních protokolů je nutné určit začátky jednotlivých vrstev ISO/OSI a extrahovat některé významné položky z hlaviček protokolů. Na následujícím obrázku 5.2 je naznačen princip zapouzdření pro IP protokol a označeny položky vhodné pro extrakci.



Obrázek 5.2: Zapuzdření protokolů a extrakce vybraných položek

Pokud se na problém zpracování struktury komunikačních protokolů podíváme z pohledu teorie formálních jazyků, většinu protokolů lze popsat pomocí regulárních nebo bezkontextových jazyků. Některé protokoly také obsahují vazby, které nelze vyjádřit pomocí bezkontextové gramatiky. Typicky se jedná o položky, které určují délku konkrétní části protokolů.

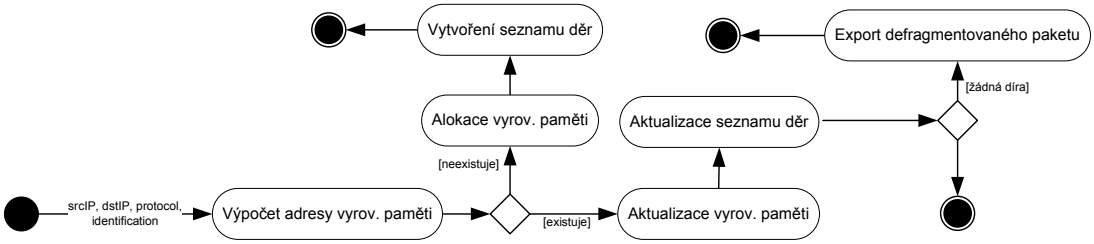
Samotná analýza protokolů a extrakce položek se pomocí programového vybavení počítače realizuje velmi jednoduše. Pro každý protokol zpravidla existuje datová struktura, která se mapuje na data přijatého rámce. Pomocí této struktury se jednoduše přistupuje k položkám dané vrstvy, které se extrahují nebo použijí k výpočtu offsetu další vrstvy.

Pro zjištění výpočetní náročnosti této činnosti, byl napsán jednoduchý program v jazyce C, který nad předem připravenými daty simuloval zpracování IPv4 paketů s TCP protokolem. Položky, které se v simulaci extrahovaly jsou uvedeny v příloze C. Tyto položky pokrývají všechny informace, které jsou potřebné pro provedení zbývajících operací v modelu identifikace protokolů. V simulacích rovněž docházelo k výpočtu délky a začátku jednotlivých vrstev ISO/OSI. Z výsledků simulací vyplývá, že na obecném procesoru by bylo možné zpracovat přibližně 21 000 000 paketů/s.

5.1.3 Skládání síťových toků na L3 a L4 vrstvě

Fragmentace síťového provozu představuje pro metody založené na vyhledávání signatur problém, který může ovlivnit výslednou přesnost těchto metod. V této kapitole bude analyzováno jak fragmentace vzniká, jak je možné provoz znovu defragmentovat a jak je daná operace výpočetně náročná.

První typ fragmentace možný se 4 verzi IP protokolu vzniká při průchodu paketu linkou s menším MTU než je velikost samotného paketu. V takovém případě musí vysílací strana (nejčastěji směrovač) zajistit rozbití paketu na fragmenty, které je možné po dané lince zaslat. Cílový uzel musí zajistit opětovné složení neboli defragmentaci na základě položek v IPv4 hlavičce. V diagramu 5.3 je zobrazena zjednodušená sekvence činnosti, které musí přijímací strana s příchodem fragmentovaných paketů vykonávat. Při tvorbě diagramu byly použity RFC 791 a RFC 815.



Obrázek 5.3: Diagram činností při defragmentaci IPv4 paketů

Proces přeskládávání je založen na udržování seznamu deskriptorů o existujících dírách. Každý deskriptor se skládá z ukazatele na začátek a konec díry. Příchozí fragmentovaný paket je pomocí dvou operací porovnán s deskriptorem díry. Může nastat několik možností. Pokud paket díru vůbec nevyplní pokračuje se ve zpracování další díry v pořadí. Pokud se paket s dírou nějakým způsobem překrývá, je díra vyplněna daty paketu. Po vyplnění může díra úplně zaniknout, můžou být modifikovány její parametry nebo v případě umístění paketu doprostřed díry vzniká díra nová. Pokud jsou všechny díry vyplněny, je paket poslán k dalšímu zpracování.

Další typ fragmentace vyskytující se na Internetu je fragmentace aplikační vrstvy přenášené přes TCP protokol. Aplikační data jsou fragmentovaná v závislosti na MSS, jež je ve výchozím nastavení 536 bytů. Takto fragmentované pakety mohou dorazit vzhledem k asymetrickému směrování v různém pořadí. Přeskládávání paketů probíhá prakticky identicky a se stejnými problémy jako u fragmentace na L3 vrstvě. Pořadí se ovšem neurčuje na základě fragment offset, ale s využitím parametru TCP protokolu sequence number. Další rozdíl oproti L3 fragmentaci je ten, že pro identifikaci aplikačních protokolů na základě vyhledávání signatur není potřeba složit kompletní zprávu aplikační vrstvy, ale předávat data

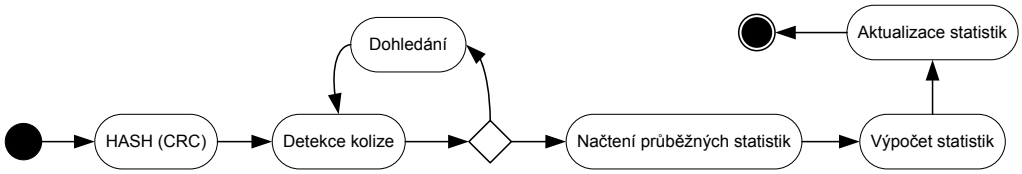
k vyhledávání v pořadí a uchovávat si stav vyhledání. Další informace je možné najít v RFC 793 nebo v článku [12] zabývajícím se skládáním TCP toků.

Vzhledem k velmi podobnému charakteru obou typu defragmentovacích algoritmů, byla provedena pouze jedna společná simulace. Pro simulaci byla zvolena právě 3% provozu jako fragmentovaného (resp. mimo pořadí). U tohoto provozu docházelo k uložení dat do vyrovnávací paměti. Pro každy fragmentovaný tok bylo nutné takto uložit 3 pakety. Zvolené parametry vycházejí z analýzy [12] fragmentace v síťovém provozu na páteřních linkách. Z výsledků simulací vyplývá, že je možné zpracovat provoz odpovídající zhruba 2 500 000 paketu/s.

5.1.4 Získávání statistických údajů a údržba stavu síťových toků

Jednou z nejdůležitějších činností v modelovaném systému je údržba stavu síťových toků spolu se získáváním a aktualizací jednotlivých statistik. Tyto činnosti jsou v navrženém modelu zobrazeny v jiných blocích, ale v této sekci budou vzhledem k úzkým vazbám analyzovány společně. V následujících odstavcích budou popsány výsledky analýzy limitů těchto činností na obecných procesorech.

Na obrázku 5.4 je zobrazena posloupnost činností, které musí být provedeny s příchodem každého paketu, který již prošel extrakcí důležitých položek. Běžným způsobem jak uchovávat informace k síťovým tokům je využití stránkování paměti spolu s hashováním vybraných položek z hlavičky identifikující síťový tok. Na základě vypočtené hodnoty rozptylovací funkce je zvolena stránka, ve které je uložen stav síťového toku. Při tomto mechanizmu je nutné řešit možné kolize v rozptylovací funkci. Kolize lze omezit vázáním kolizních položek do seznamu nebo zvětšením paměti. Při zpracování síťového provozu se v současné době nelze kolizím vyhnout a je potřeba s nimi počítat.

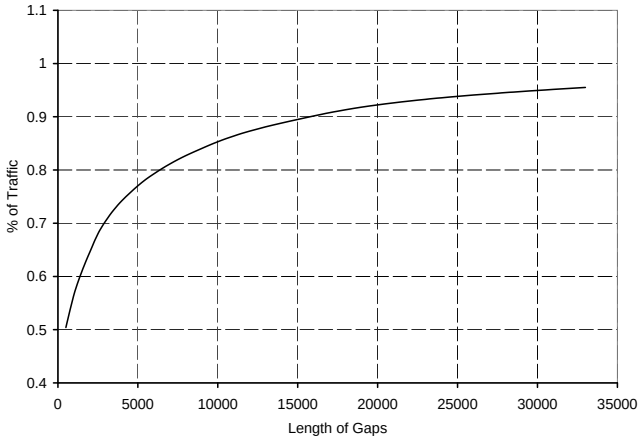


Obrázek 5.4: Diagram činností při získávání statistických údajů

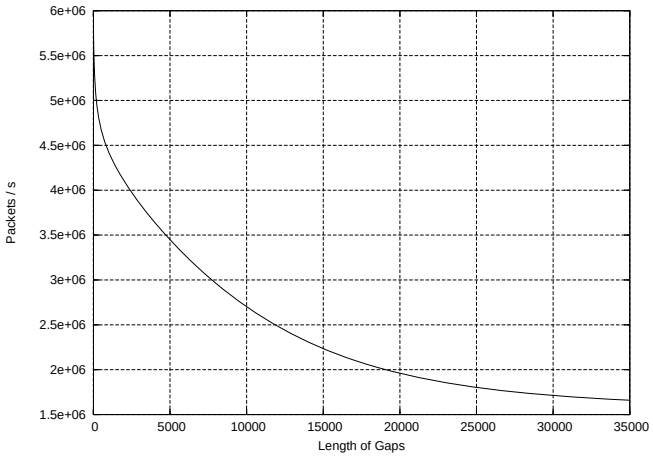
Pro výpočet průměrné doby zpracování každého paketu byl napsán jednoduchý program v jazyce C, ve kterém byly prováděny jednotlivé operece nad velkým množstvím předem připravených dat. Od doby provádění programu byly odečteny časy pro inicializaci náhodných dat a výsledný čas byl vydělen počtem zpracovaných paketů. V připravených datech se vyskytovalo celkem 100 tis paketů každý z jiného síťového toků. Tyto pakety byly několikrát odeslány k zpracování. Doba zpracování každého paketu se průměrně rovnala 6 ms.

Nízká propustnost je zejména způsobena poměrně náročnou operací výpočtu CRC a dále výpadky bloků v paměti cache při práci se stavem síťového toku díky nulové časové lokalitě dat. Skutečný síťový provoz však časovou lokalitu má. V diplomové práci [64] zabývající se monitorováním sítí pomocí NetFlow je zkoumáno, jaká se vyskytuje mezera (množství paketů z jiných síťových toků) mezi pakety stejného toku. Poznatky z této práce zobrazené v grafu 5.5 byly použity k upravení simulace, aby byla zohledněna daná časová lokalita síťových toků. V grafu 5.6 je zobrazeno kolik paketů je možné zpracovat při dané časové

lokalitě mezer. Simulace byla provedena na výkoném procesoru Intel Itanium se 4 Mb L2 cache.



Obrázek 5.5: Distribuční funkce časové lokality síťových toků



Obrázek 5.6: Propustnost v závislosti na časové lokalitě síťových toků

Dále byl analyzován výkon existujících softwarových systémů [1, 10] pro sběr NetFlow statistik. Tyto systémy musí rovněž provádět popsané operace a zároveň nabízejí dostatečnou flexibilitu k přidávání nových statistických údajů potřebných k identifikaci protokolů (lze vlastními silami doprogramovat). Propustnost těchto řešení byla analyzována jak vývojáři těchto systému, tak nezávislými zdroji [65] a pohybuje se pro nejkratší pakety okolo 1 Gb/s. Tato hodnota je velmi blízká hodnotě získané z provedených simulací.

5.1.5 Vyhledávání vzorů

Základní operací identifikace aplikačních protokolů na základě signatur je vyhledávání regulárních výrazů. V kapitole 2 byly uvedeny hardwarové přístupy pro rychlé vyhledávání vzorů, ale nebyly zhodnoceny možnosti vyhledávání regulárních výrazů na obecném procesoru. V této kapitole jsou díky provedeným simulacím a testům stanoveny konkrétní limity této činnosti pro oblíbenou knihovnu PCRE [47].

Pro analýzu propustnosti činnosti vyhledávání vzorů byl vytvořen testovací program, který nejprve alokoval paměť odpovídající počtu nejdelších paketů na gigabitovém ethernetu. Tato paměť byla vyplněna náhodnými daty. S využitím knihovny PCRE bylo v takto připravených paketech vyhledáváno různé množství regulárních výrazů a byla měřena doba zpracování. Na základě provedených experimentů byla stanovena maximální možná propustnost softwarového vyhledávání regulárních výrazů následujícím vzorcem 5.1. Parametr *regexp_throughput* byl z experimentů stanoven na 3700 Mb/s. Při předpokládaném použití až 200 regulárních výrazů propustnost vyhledávání dosáhne 18 Mb/s, což zdaleka nedostačuje pro použití ani na 100 Mb/s sítích. Tyto výsledky zhruba odpovídají i reálné propustnosti programu L7 filter uvedené na jeho webových stránkách.

$$Throughput[Mbps] = regexp\_throughput/regexp\_count \tag{5.1}$$

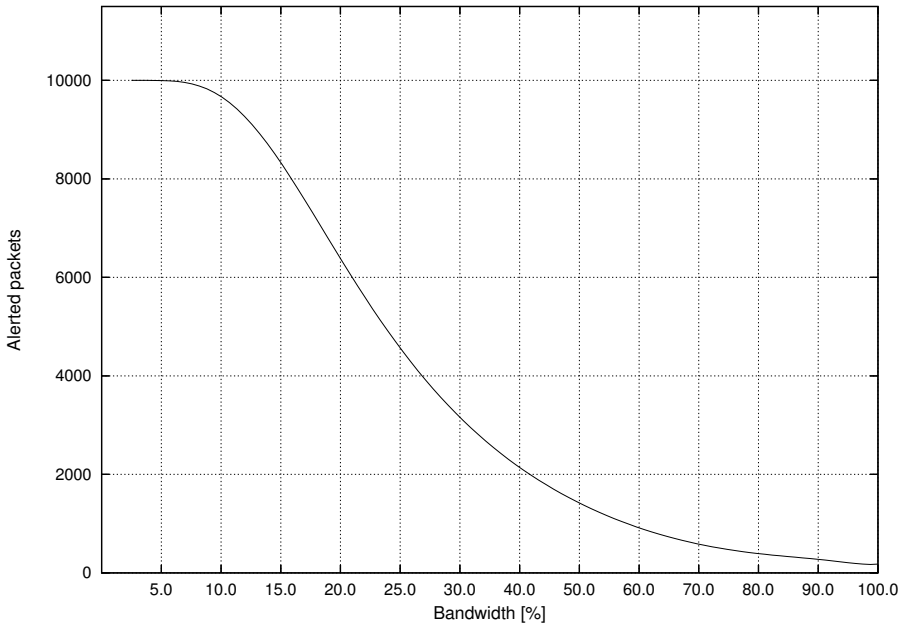
Operaci vyhledávání vzorů je možné vhodnými algoritmy dále optimalizovat a dosáhnout trošku lepších výsledků. Lze říct, že v současné době je potřeba nejvíce optimalizovat tuto činnost v IDS systémech, které v každém paketu vyhledávají tisíce vzorů. Proto byla analyzována propustnost nejrozšířenějšího softwarového IDS systému Snort. Pro analýzu byl systém nakonfigurován pro vyhledávání 200 vzorů a zároveň byly vypnuty všechny moduly, které řeší udržování stavových informací k síťovým tokům, ressembling atp. Na takto nakonfigurovaný Snort byl vysílán provoz na různých rychlostech a byla provedena měření kolik nebezpečných paketů je schopen systém detekovat. V následujícím grafu 5.7 jsou shrnuty dosažené výsledky. Z grafu je patrné, že IDS systém Snort dosahuje nepatrně větší propustnosti, než jaká byla zjištěna pomocí simulací s knihovnou PCRE. Bohužel ani tato propustnost nedostačuje pro použití na gigabitových sítích.

Dále bylo analyzováno, jaký vliv má na celkovou potřebnou propustnost v jakém množství dat síťového toku dochází k vyhledávání vzorů. Dle zkušebních měření se na páteřních linkách vyskytuje maximálně 100 tisíc nových toků za sekundu. Pokud se využije toho, že protokol lze poměrně úspěšně identifikovat na základě analýzy prvních 4 Kb dat, lze razantně snížit požadavky na propustnost vyhledávaní vzorů. Podle vzorce 5.2 a výše zvolených parametrů stačí pro analýzu páteřních sítí vyhledávací jednotka s propustností 3.2 Gb/s. Tento poznatek má významný dopad na návrh jednotky pro vyhledávání vzorů.

$$ReqThroughput[Gb/s] = 8 * FlowsPerSecond * 4.10^{-6} \tag{5.2}$$

5.1.6 Klasifikace

Poslední a z hlediska identifikace provozu také nejdůležitější činností v modelovaném systému je klasifikace. Pro analýzu výpočetní náročnosti a přesnosti klasifikace bylo potřeba najít vhodnou trénovací množinu, vybrat atributy síťového provozu s ohledem na přesnost detekce a složitost jejich výpočtu. V neposlední řadě bylo potřeba zvolit vhodný klasifikátor, který je schopný pracovat s rozsáhlými trénovacími množinami a klasifikuje dostatečné



Obrázek 5.7: Propustnost IDS systému Snort na 1 Gb/s pro 200 vyhledávaných vzorů

množství síťových toků za sekundu. V této kapitole jsou rozebrány výsledky této analýzy a provedených simulací.

Vzorek síťového provozu, který by byl dopředu klasifikován podle síťových aplikací je velmi problematické získat. Vzorek musí být získáván na gigabitových rychlostech, příchozí pakety musí být opatřeny přesnými časovými značkami, musí být vypočteny potřebné statistiky a musí být určen typ aplikačního protokolu. Rovněž bývá problém s přístupem k celým paketům (mohou obsahovat citlivé informace). Dostupné vzorky síťového provozu jsou proto často uchovávány jako netflow záznamy nebo pakety mimo dat na L5-L7 vrstvě ISO/OSI. IP adresy v takto uložených záznamech jsou také často anonymizovány. Vhodná sada síťového provozu byla nalezena na webových stránkách university v Cambridge[1]. Sada obsahuje anotovaný provoz včetně předpočítaných statistik (257 statistických vlastností) a tím představuje velmi vhodnou sadu pro zjištění možností statistických metod identifikace protokolů.

Před samotnou klasifikací je nutné vytipovat vhodnou sadu atributů (statistických vlastností) síťových toků. Je vhodné vybrat takové atributy, které lze poté aktualizovat on-line tj. bez nutnosti uchovávat si historii hodnot pro předchozí pakety. Nevhodným atributem je tedy např. výpočet mediánu, pro jehož výpočet je potřeba uchovat všechny předcházející hodnoty, seřadit je a vybrat středovou. Atributy by pokud možno měly být odvozeny ze standardních položek v IP/TCP/UDP paketech. Nevhodné jsou atributy, které jsou vypočteny z variabilních položek, které bývají složitěji parsovatelné (IPv4, IPv6, TCP options). Na základě těchto omezení bylo vybráno celkem 32 statistických vlastností síťových toků, které jsou popsány v příloze A.

Takto velká množina atributů ovšem vyžaduje nemalé paměťové nároky jak na celkovou kapacitu paměti pro uložení stavových informací k síťovým tokům, tak na její propustnost. Proto je vhodné množinu atributů dále zmenšovat, ale zároveň zachovávat i vysokou

[1] http://www.cl.cam.ac.uk/research/srg/netos/nprobe/data/papers/sigmetrics/index.html

přesnost detekce. K redukci množiny atributů je možné využít nástroj Weka, který obsahuje algoritmy pro seřazení atributů podle jejich významnosti na proces klasifikace. V následující tabulce je ze zvolené trénovací množiny vybráno 10 nejvýznamnějších atributů jak pro komunikaci mezi klientem a serverem tak v opačném směru.

Server → Client		Client → Server	
Atribut	Využití	Atribut	Využití
var_data_wire_ba	100 %	mean_data_ip_ab	100 %
pushed_data_pkts_ba	100 %	pushed_data_pkts_ab	99 %
actual_data_bytes_ba	94 %	max_data_wire_ab	97 %
pure_acks_sent_ba	91 %	mean_data_wire_ab	90 %
max_win_adv_ba	88 %	FIN_pkts_sent_ab	86 %
SYN_pkts_sent_ba	83 %	actual_data_pkts_ab	86 %
var_data_control_ba	83 %	mean_IAT_ab	85 %
min_win_adv_ba	66 %	var_data_control_ab	85 %
min_IAT_ba	26 %	actual_data_bytes_ab	75 %
mean_IAT_ba	24 %	SYN_pkts_sent_ab	67 %

Tabulka 5.1: Redukovaná množina atributů síťového provozu

Na základě analýzy existujících klasifikátorů byl zvolen klasifikátor založený na rozhodovacích stromech. Existující implementace See5 splňuje všechny vlastnosti, které byly po klasifikátoru požadovány. Klasifikátor je možné natrénovat s využitím trénovací množiny obsahující stovky tisíc trénovacích vstupů. Výstup klasifikátoru představuje lidmi čitelnou datovou strukturu, kterou lze v případě potřeby jednoduše realizovat v hardware. Natrénovaný klasifikátor lze načíst pomocí přiloženého kódu v jazyce C a následně použít pro libovolnou aplikaci. Klasifikátor je velmi rychlý, doba klasifikace závisí pouze na maximální hloubce vytvořeného stromu. Ukázka výstupu programu See5 je zobrazena v příloze B.

Se zvoleným klasifikátorem, různě omezenou množinou atributů a vybranou trénovací množinou byla analyzována přesnost detekce aplikačních protokolů. Část trénovací množiny byla použita pro natrénování klasifikátoru, druhá část množiny byla použita pro ověření přesnosti detekce. V tabulce 5.2 jsou zobrazeny výsledky pro oba směry provozu pro různě velkou množinu atributů. Rovněž je zde uvedena přesnost při použití všech vybraných atributů. Jak můžeme vidět z výsledků, velké množství atributů neimplikuje současně vysokou přesnost. I s malým počtem atributů lze dosáhnou opravdu velké přesnosti identifikace protokolů.

Dále bylo analyzováno kolik síťových toků je schopna klasifikační jednotka zpracovat za sekundu. Pro tento účel byl vytvořen testovací program, který načetl natrénovaný klasifikátor a trénovací množinu do operační paměti. Poté bylo spuštěno měření času a všechny toky v paměti byly klasifikovány. Z naměřených výsledků vyplývá, že jeden síťový tok se v průměru klasifikuje 1 ms.

5.2 Rozdělení mezi hardware a software

Výpočetní model a analýzu výpočetní náročnosti lze použít k mapování jednotlivých operací mezi hardware a software s cílem dosáhnout určité propustnosti. Výsledky je možné použít také pro efektivní mapování jednotlivých operací na víceprocesorový systém, čímž

Počet atributů	Server → Client	Client → Server
32	4.1 %	3.6 %
10	4.1 %	1.8 %
9	4.1 %	1.8 %
8	4.1 %	4.6 %
7	4.1 %	4.8 %
6	4.4 %	5.1 %
5	4.3 %	5.1 %
4	4.7 %	5.1 %
3	4.8 %	5.1 %
2	5.2 %	6.3 %
1	6.2 %	9.1 %

Tabulka 5.2: Chybně klasifikované síťové toky

je možné pro některé operace dosáhnout větší propustnosti. V této sekci bude provedeno rozdělení modelu mezi hardware a software s cílem provádět identifikaci protokolů na multigigabitových sítích.

Před samotným mapováním jednotlivých operací na hardwarové nebo softwarové vybavení počítače je nutné se zabývat požadavky na propustnost zařízení umístěných na různých typech sítí. Pokud má zařízení pracovat na linkových rychlostech musí dosahovat parametrů uvedených v tabulce 5.3. V uvedených parametrech pro různé druhy Ethernetu jsou rovněž započteny potřebné mezery mezi Ethernetovými rámci. Důležitým parametrem je zde minimální čas potřebný ke zpracování nejkratšího paketu. Ten musí být splněn, aby zařízení bylo schopno pracovat na linkových rychlostech za všech okolností (i v případě útoku). Parametr datový tok je spočten z velikosti zdrojové a cílové MAC adresy, Length/Type, velikosti přenášených dat na L3-L7 vrstvě a CRC.

	Min. velikost paketu		Max. velikost paketu		Doba zpracování	
Typ sítě	Paket/s	Datový tok	Paket/s	Datový tok	Min	Max
10Base-TX	14 880	7,62 Mb/s	813	9,87 Mb/s	67,2 μs	1 230 μs
100Base-TX	148 800	76,2 Mb/s	8 130	98,7 Mb/s	6,72 μs	123 μs
1000Base-T	1 488 000	762 Mb/s	81 300	987 Mb/s	672 ns	12 300 ns
10GBase-TX	14 880 000	7,62 Gb/s	813 000	9,87 Gb/s	67 ns	1 230 ns
40GBase	59 520 000	30,48 Gb/s	3 252 000	39,5 Gb/s	16 ns	307 ns

Tabulka 5.3: Parametry vstupního rozhraní pro různé typy Ethernetu

V tabulce 5.4 jsou shrnuty výsledky provedených simulací a je určeno, které operace je potřeba akcelerovat, aby bylo možné dosáhnout dostatečné propustnosti pro daný typ sítě. Z výsledků vyplývá, že největším problémem identifikace protokolů je vyhledávání regulárních výrazů, které je potřeba akcelerovat už na gigabitových sítích. U více gigabitových sítí je nutné akcelerovat prakticky veškeré operace. Pouze klasifikaci síťových toků je možné provádět v programovém vybavení počítače, pokud není nutné klasifikovat více než 1 milión toků, což při běžném zatížení sítě nenastává.

Operace	Propustnost	1000Base-T	10GBase-TX	40GBase
Analýza protokolů	21 000 000 p/s	✓	✓	✗
Defragmentace	2 500 000 p/s	✓	✗	✗
Získávání statistik	3 000 000 p/s	✓	✗	✗
Vyhledávání vzorů	18 Mb/s	✗	✗	✗
Klasifikace	1 000 000 toků/s	✓	✓	✓

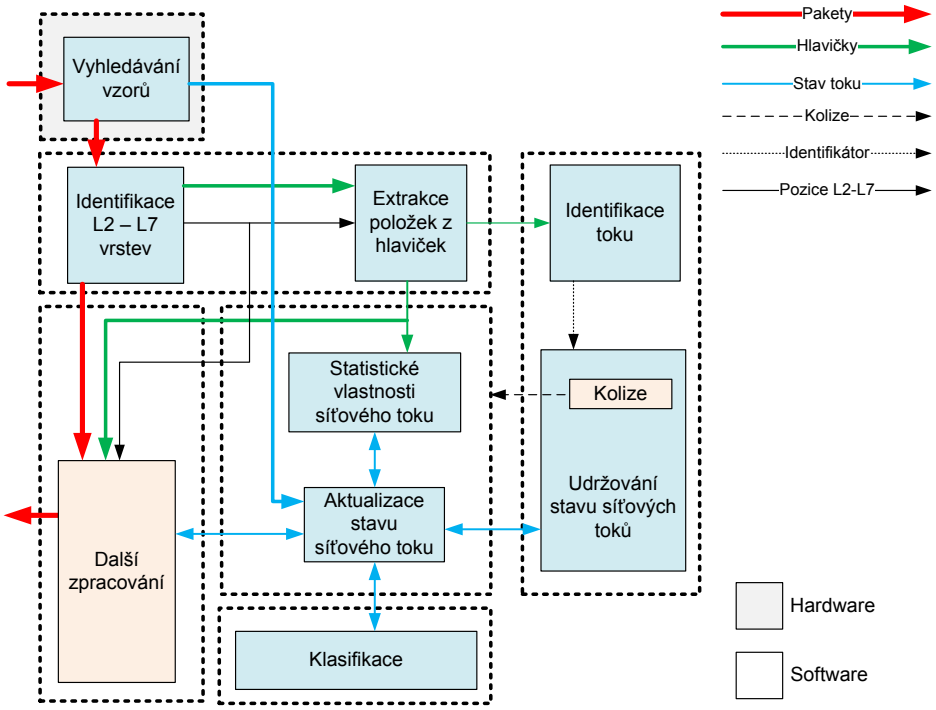
Tabulka 5.4: Možnosti identifikace protokolů na obených počítačových architekturách

Pro použití identifikace aplikačních protokolů na gigabitových sítích je nutné akcelerovat vyhledávání regulárních výrazů. Tuto činnost však nelze vzhledem k úzkým vazbám předřadit před ostatní operace. Nejprve je nutné složit tok a následně vyhledávat. Posílání složeného toku na akcelerační kartu a přijmutí výsledku je také problematické, vzhledem k velké latenci, kterou není možné díky zpětným smyčkám v modelu překrýt. Odstranění operace defragmentace L3 a L4 vrstvy by tento problém vyřešilo. Její odstranění je také velmi výhodné z hlediska hardwarové realizace. Tyto operace totiž představují implementačně nejnáročnější část - je potřeba spravovat seznam děr, alokovat a uvolňovat paměť pro pakety mimo pořadí, což jsou hardwarově obtížně realizovatelné operace.

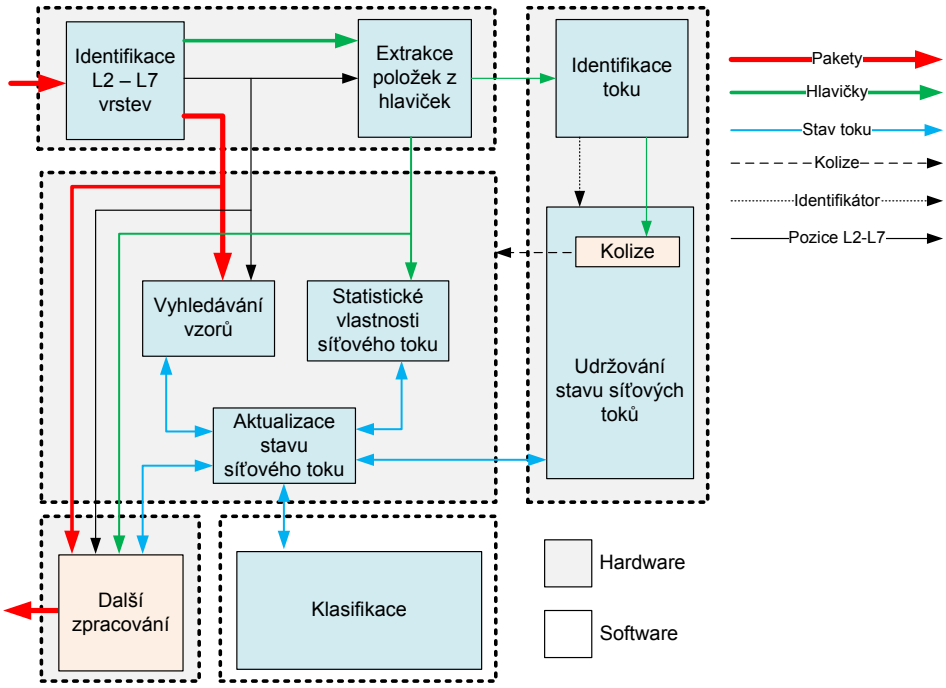
Pro odstranění této operace z výsledné realizace hovoří kromě její realizační náročnosti, také její poměrně malý vliv na zpřesnění detekce protokolů. Rovněž skládání paketů do toků nelze uspokojivě řešit uprostřed sítě vzhledem k asymetrickému směrování. Také existují techniky [21], jak systém donutit ke složení toků tak, aby nebyl vyhledávaný výraz nalezen (např. vhodným nastavením TTL). Existující implementace (L7filter a IPP2P) rovněž skládaní toků neobsahují. Z těchto důvodů nebyla operace skládání síťových toků do výsledných modelů zařazena.

Na obrázku 5.8 je zobrazeno mapování jednotlivých operací mezi HW a SW pro 1 Gb/s sítě. Po odstranění operace skládání síťových toků postačí v paketech v HW vyhledávat konkrétní vzory a informace o nalezených vzorech předat spolu s paketem k dalším operacím identifikace aplikačních protokolů, které jsou realizovány v software. Toto řešení je velmi jednoduché na realizaci, ale v současné době už nedostačuje pro použití na nynějších páteřních linkách. Proto je nutné se zabývat mapováním operací tak, aby bylo výsledné řešení použitelné na 10 až 40 Gb/s sítích.

Pro 10 a 40 Gb/s bylo provedeno rozdělení mezi hardware a software způsobem zachyceným na obrázku 5.9. Prakticky veškeré operace jsou realizovány v hardware, pouze na procesoru je realizovaná klasifikace síťových toků na základě nalezených vzorů a získaných statistických informací. V další kapitole bude popsáno jak je možné realizovat jednotlivé operace v HW, konkrétně s využitím technologie FPGA a akceleračních karet rodiny COMBO.



Obrázek 5.8: Mapování operací mezi HW a SW pro 1 Gb/s



Obrázek 5.9: Mapování operací mezi HW a SW pro 10 a 40 Gb/s

Kapitola 6

Hardwarová realizace identifikace aplikačních protokolů

Vytvořený model identifikace aplikačních protokolů, byl namapován na hardwarové a softwarové prostředky. Jako akcelerační platforma byla zvolena rodina karet COMBO6 a výsledná aplikace byla vybudována nad hardwarovou abstraktní vrstvou NetCOPE. Navržená architektura sondy pro akceleraci aplikačních protokolů vychází z architektury sondy FlowMon [64], která je schopna sbírat NetFlow statistiky na multigigabitových rychlostech. Stávající architektura byla rozšířena zejména o možnost rychle vyhledávat regulární výrazy a tím podporovat detekci protokolů na základě signatur. Rovněž byly vybrány statistické vlastnosti, které jsou vhodné pro hardwarovou realizaci a zároveň zachovávají vysokou přesnost detekce aplikačních protokolů. V této kapitole bude popsána navržená architektura a dosažené výsledky implementace jednotky pro vyhledávání regulárních výrazů.

6.1 Platforma

Cílovou platformou pro realizaci firmwarové části sondy pro identifikaci aplikačních protokolů byla zvolena desetigigabitová 2XFP karta, kterou lze osadit na základní karty rodiny COMBO6. V následujících odstavcích budou obě karty popsány.

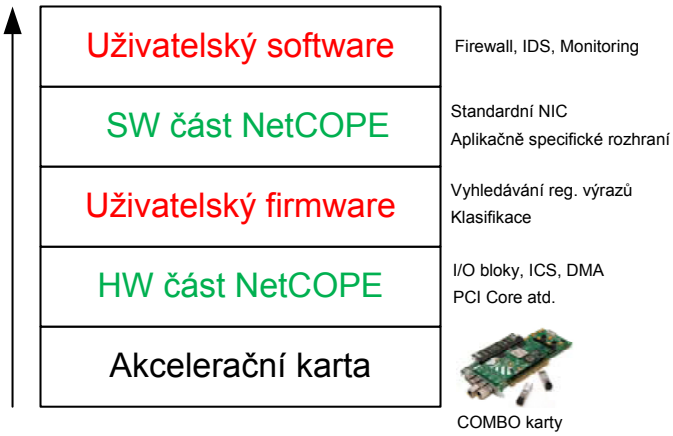
Rodina karet COMBO6 vyvíjených v rámci projektu Liberouter [31] představuje univerzální platformu pro akceleraci řady síťových aplikací. Koncept karet je založen na myšlence hardware-software codesign. Časově kritické operace jsou realizovány jako firmware na akceleračních kartách, kdežto výpočetně méně náročné části jsou realizovány pomocí programového vybavení počítače. Přenos mezi kartou a hostitelským systémem probíhá po rychlých sběrnicích (PCI-X, PCI-Express), které nabízí dostatečný výkon k přenosu velkého množství dat k procesoru.

Základní karta se skládá z programovatelného hradlového pole Xilinx Virtex-II Pro XC2VP50 [62, 61], 2 Mb ternární asociativní pamětí (TCAM), konektoru pro dynamickou paměť a ze tří 2 MB statických pamětí (SSRAM). Karta je určena pro zapojení do osobního počítače pomocí sběrnice PCI (COMBO6X) nebo PCI Express (COMBO6E). Komunikaci se sběrnicí usnadňuje další FPGA obsahující PCI/PCI Express core. Díky rozšiřujícímu slotu ji lze vybavit přídavnou kartou se síťovým rozhraním.

Pro základní kartu COMBO vznikla celá řada rozšiřujících karet (COMBO-4MTX, COMBO-2XFP, COMBO-4SFP, COMBO-4SFPRO), které se liší zejména druhem použitých síťových konektorů a kapacitou osazeného FPGA. Díky těmto kartám je možné

použít měděné (MTX) nebo optické (SFP) gigabitové rozhraní. Desetigigabitové rozhraní je možné najít na XFP kartě. Navržený systém bude realizován na desetigigabitové 2XFP kartě, která je vybavena dvěmi XFP klecemi pro 10GE. Tato karta je osazena Xilinx Virtex-II Pro XC2VP20 a 2x 2 Mb SSRAM.

Nad kartami COMBO, ale také nad kartami jiných výrobců, byla vytvořena hardwarová abstraktní vrstva nazvaná NetCOPE [33]. Účelem této vrstvy je vytvořit univerzální rozhraní mezi hardwarem a softwarem a zároveň implementovat HW a SW bloky společné pro většinu síťových aplikací. Mezi tyto bloky patří zejména bloky pro příjem a vysílání paketů, přístup k paměťovým prvkům, přístup k akcelerační kartě z uživatelské aplikace a rychlé DMA přenosy. Vrstvový model abstraktní vrstvy NetCOPE je znázorněn na obrázku 6.1.

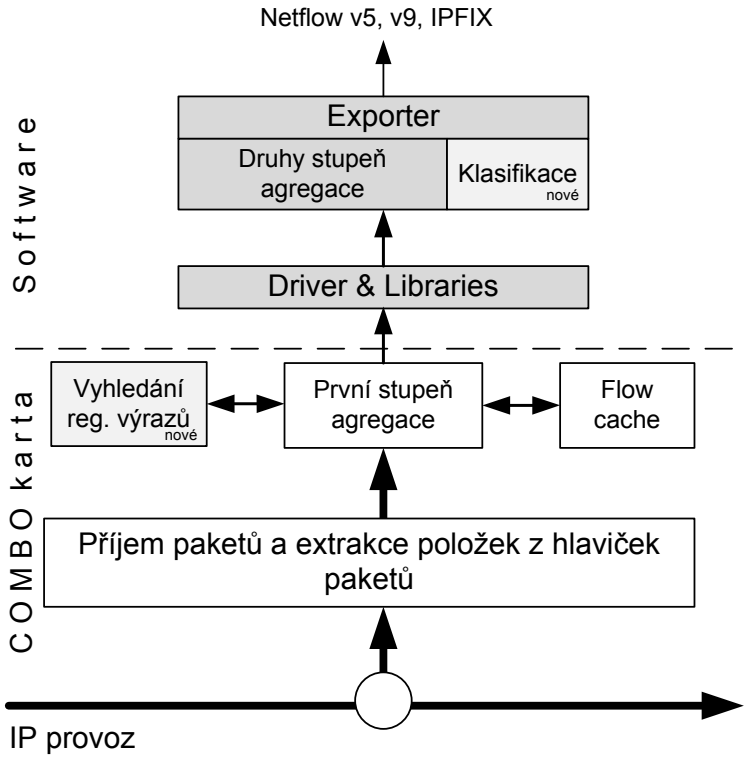


Obrázek 6.1: Abstraktní vrstva NetCOPE

6.2 Systémová architektura

Navržená systémová architektura zachycená na obrázku 6.2 se skládá z hardwarové a softwarové části. Hardwarová část je připojena k monitorovanému segmentu sítě v in-line režimu pomocí dvou portů zapojených na monitorovanou 10 GE linku. Sonda může být také připojena ke dvěma 10 GE replikačním portům aktivního prvku v síti, čímž je jí umožněno monitorovat veškerý síťový provoz procházející aktivním prvkem. Samotný firmware sondy je odpovědný za zpracování paketů na gigabitových rychlostech, agregaci informací o paketech do záznamů o tocích (včetně statistických informací potřebných k identifikaci protokolů) a identifikaci protokolů na základě vyhledání regulárních výrazů v aplikační vrstvě zpracovávaných paketů. Informace o tocích a nalezených regulárních výrazech jsou po delší době neaktivity (inactive timeout) nebo po uplynutí uživatelem nastaveného času (active timeout) přeneseny přes sběrnici k softwarovému zpracování. Exportované záznamy jsou zpřístupněny softwarovým vrstvám pomocí driveru a knihoven platformy NetCOPE.

Software je primárně odpovědný za export informací o tocích (včetně identifikovaných aplikačních protokolů) na kolektor. Sekundárně také snižuje počet fragmentovaných toků tj. toků, které byly exportovány dříve než měly (kolize, nedostatek paměti v hardware). Snížení počtu fragmentovaných toků je možné díky druhému stupni agregace. Informace o síťovém toku jsou před odesláním na kolektor předány klasifikačnímu bloku, který určí



Obrázek 6.2: Systémová architektura rozšířené sondy Flowmon

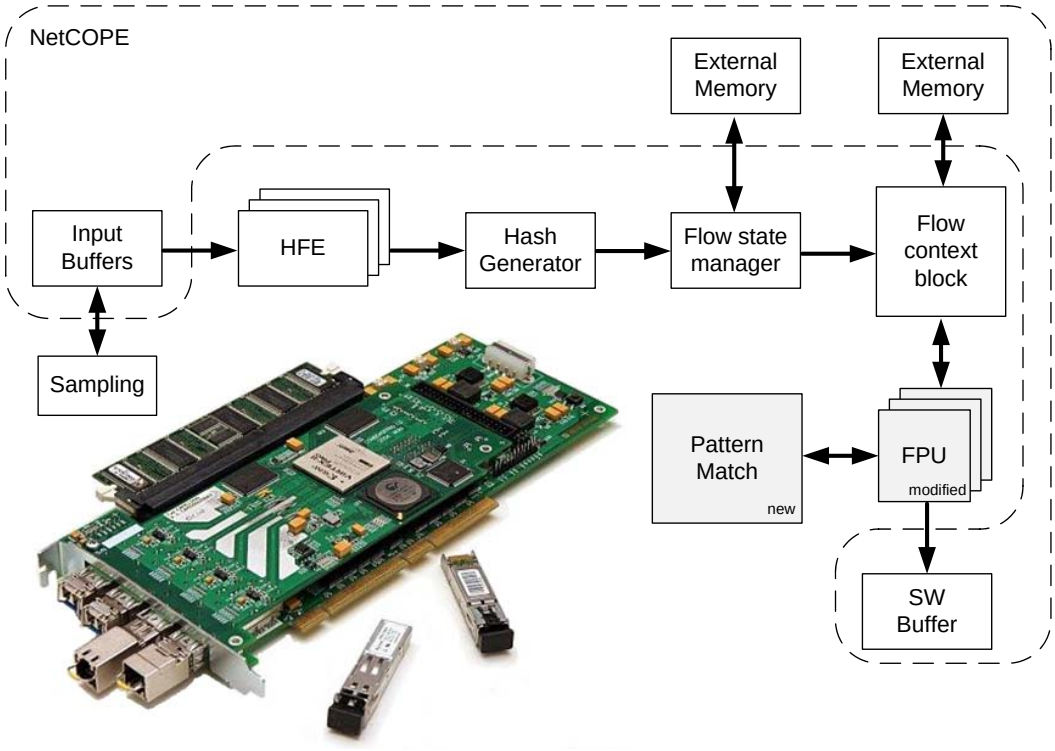
typ aplikačního protokolu. Ke klasifikaci jsou použity rozhodovací stromy, jejíchž parametry jsou popsány v předchozí kapitole. Informace o tocích a použitých aplikačních protokolech jsou pomocí protokolu Netflow v.9 s konfigurovatelným formátem odeslány na kolektor, kde mohou být použity k rekonfiguraci firewallu, routeru (traffic shaping), vizualizovány nebo použity jako vstup do řady bezpečnostních aplikací (annomaly based IDS).

6.3 Firmware

Firmware sondy FlowMon využívá řady síťových IP core vyvinutých v rámci projektu Liberouter. Firmware rovněž obsahuje řadu aplikačně specifických bloků, vytvořených speciálně pro tuto sondu. Blokové schéma firmwarové architektury (obrázek 6.3) sondy FlowMon rozšířené o možnosti identifikace aplikačních protokolů bude popsáno v dalších odstavcích.

Příchozí pakety jsou přijaty ze vstupního rozhraní a následně uloženy do vyrovnávací paměti vstupních bufferů *(Input Buffers)*. Pro každý příchozí paket je ve vstupních bufferech vypočteno CRC, a to je porovnáno s uloženou hodnotou. Pouze pakety se správným CRC jsou předány dál. Mezi další činnosti, které musí vstupní buffery řešit, patří např. odstranění zarovnání do minimální délky, odstranění položek ethernetového rámce jako je Preambule, SDF a CRC atd. K paketu je rovněž přidáno časové razítko, které je dále použito k výpočtu statistik (např. průměrná mezipaketová mezera) vhodných k identifikaci aplikačních protokolů.

Pakety ve vstupních rozhraních mohou být zahozeny samplovací jednotkou *(Sampling)* v závislosti na zaplnění pamětí nebo rychlosti příchodu paketů. Tím lze získat poměrně



Obrázek 6.3: Architektura firmware rozšířené sondy FlowMon

přesné informace o tocích vyskytujících se v síti i v případě útoku na paměť sondy (velké množství toků). Aby bylo při samplování možné získat přibližnou představu např. o počtu přenesených bytů, je nutné tento parametr násobit koeficientem samplování. Samplování má bohužel zásadní vliv na identifikaci protokolů na základě vyhledávání signatur, kde vede na průměrně n krát menší množství klasifikovaných síťových toků, kde n je koeficient samplování. Přesnost identifikace aplikačních protokolů na základě statistických informací bude samplováním také ovlivněna, proto se nedoporučuje ho při běžném provozu používat. Pakety, které nebyly samplovací jednotkou zahozeny jsou dále zarovnány na 16 bitů a předány k dalšímu zpracování.

Jako komunikační protokol mezi jednotlivými bloky je zvolen FrameLink, který vychází z protokolu LocalLink používaného firmou Xilinx. Jedná se o obousměrně potvrzovaný protokol, který zasílá data v rámci skládajícího se z generického počtu paketů (typicky hlavička, tělo, patička). Protokol má generickou šířku a poslední slovo každého paketu je navíc označeno binárně zakódovaným příznakem platnosti (DREM). Oproti protokolu LocalLink není tolik náročný na hardwarové zdroje při práci s více paketovými rámci. K tomuto protokolu vznikla celá řada pomocných komponent, které umožňují měnit šířku protokolu, spojovat výstup z více procesních jednotek, rozdělovat rámce mezi více procesních jednotek a tím vyvažovat zátěž atd.

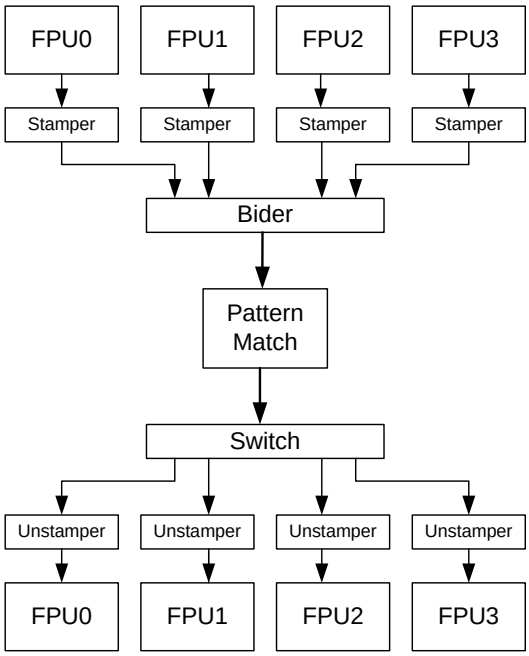
Právě vyvažování zátěže mezi více procesních jednotek je využito při extrakci hlaviček. Pro extrakci využitý IP core *HFE-C*, totiž nedosahuje potřebné propustnosti pro zpracování 10 GE. Proto musí být využito několik těchto IP core, aby tato část zpracování nebyla úzkým místem navržené architektury. Položky z hlaviček protokolů, které se mají extrahovat

jsou zvoleny před syntézou firmware. Pokud je potřeba extrahovat jiné položky je potřeba celý systém znova syntetizovat.

Extrahované položky a samotný paket je předán do jednotky *Hash generator*. Tato jednotka z vybraných položek vypočítá adresu paměti, na které se nachází odpovídající záznam o síťovém toku. Výpočet je realizován jako CRC z IP adres, portů a typu protokolu. Vypočtený identifikátor a již dříve získaná časová značka je ještě před načtením uložených informací k síťovému toku použita v jednotce *Flow state manager*. Tato jednotka hlídá tzv. neaktivní timeout. Pokud síťový tok nebyl delší dobu aktivní, je potřeba uvolnit a exportovat záznam o síťovém toku. Hlídání neaktivity je realizováno jako procházení seznamu časových značek s poslední aktivitou síťového toku.

Uchovávání informací k síťovým tokům realizuje IP core *FlowContext* [28]. Tento síťový core se snaží o minimální přístup k hlavní paměti, tím že realizuje vyrovnávací paměť v interních BlockRam pamětech. FlowContext zároveň umožňuje vyvažovat zátěž mezi více procesními jednotkami, které jsou odpovědné za aktualizaci kontextové informace k síťovému toku. Tím je možné překrýt latenci hlavní paměti a také zvyšovat propustnost pouhým přidáváním dalších výpočetních elementů.

Stav síťového toku, extrahované hlavičky a samotný paket jsou zpřístupněny jednotkám pro zpracování síťových toků *FPU*. Tyto jednotky jsou odpovědné za správu informací a statistik k síťovým tokům. Tato jednotka také přeposílá aplikační data do vyhledávací jednotky *Pattern Match* a na základě výsledků vyhledávání aktualizuje uložený bitový vektor nalezených regulárních výrazů. Vyhledávací jednotka je v případě použití více procesních elementů sdílena způsobem naznačeným na obrázku 6.4. Procesní jednotka v případě naplnění aktivního/pasivního timeoutu nebo ukončení spojení exportuje informace o síťovém toku pomocí DMA s využitím NetCOPE aplikačně specifického rozhraní.

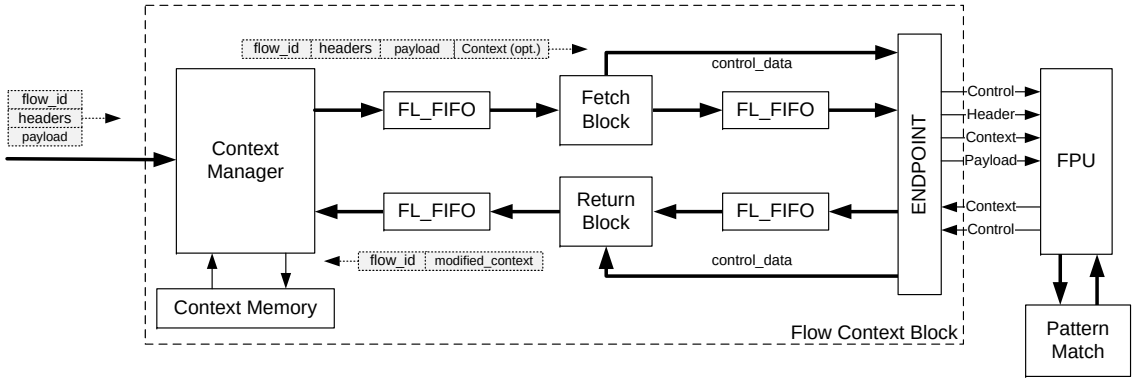


Obrázek 6.4: Sdílení vyhledávací jednotky

6.3.1 Flow processing unit

V této části bude podrobněji rozebrána architektura jednotky FPU, která je nejdůležitější a také nejsložitější části navrhované sondy. Tato architektura je detailně popsána v práci [30]. Popis uvedený zde se bude soustředit na základní koncepty této architektury a modifikace s ohledem na identifikaci protokolů.

Na obrázku 6.7 je znázorněno zapojení FPU k vyhledávací jednotce a FlowContextu. FlowContext přijímá pakety, extrahované hlavičky a identifikátor síťového toků. Pokud je kontext k příchozímu toku uložen v některé z vyrovnávacích pamětí umístěných v Endpoint blocích, jsou paket a hlavičky předány ke zpracování do odpovídající FPU. Pokud se kontext ve vyrovnávací paměti nenachází, musí se vyčíst z externí paměti.



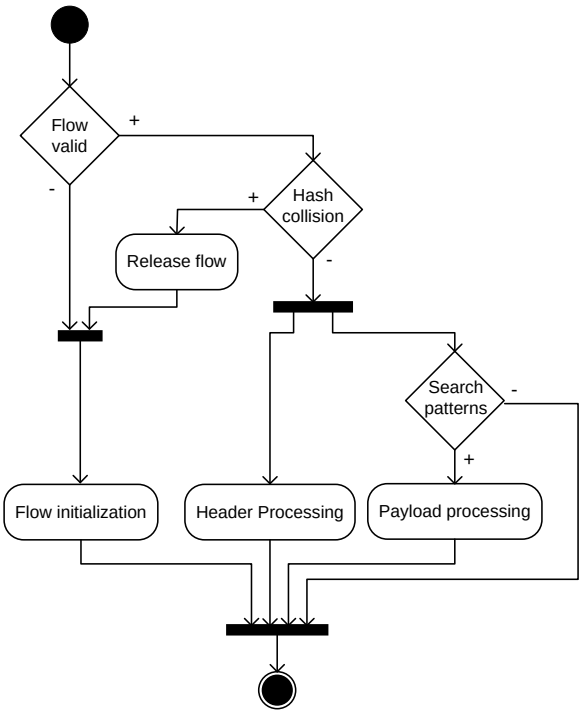
Obrázek 6.5: FlowContext a Flow Processing Unit

Data směrem k FPU procházejí Fetch blokem. Tento blok extrahuje z kontextu řídící data, která jsou předána FPU pomocí nezávislého řídicího rozhraní. Fetch blok extrahuje informace o platnosti kontextu (namapování na prázdné místo v paměti) a kolizích (namapování na jiný existující síťový tok). Na základě těchto informací provede FPU odpovídající činnosti (založení nového toku, export kolizního záznamu, aktualizace toku). Právě tento blok se jeví jako vhodný pro umístění mechanizmu, který rozhodne o vyhledání regulárních výrazů v datech paketu. Toto rozhodnutí může probíhat na základě počtu již prohledaných paketů (resp. bytů) daného toku a typu protokolu na L3 a L4 vrstvě.

Kontext, extrahované hlavičky a aplikační data jsou předány FPU jednotce, která může streamovým zpracováním aktualizovat nebo exportovat kontext do SW. Endpoint umožňuje streamové zpracování dvou kontextů najednou. Aktualizovaný kontext je uložen ve vyrovnávací paměti v endpoint bloku a zároveň je předán k uložení do paměti kontextů. Return blok na základě nových řídících informací aktualizuje kontext (platnost, počet prohledaných paketů resp. bytů).

Samotná FPU pracuje podle control flow diagramu naznačeného na obrázku 6.7. V případě, že není kontext platný (jedná se o první paket toku), je založen kontext nový. U platného kontextu může dojít ke kolizi. V takovém případě je nutné exportovat informace o starém síťovém toku a založit tok nový. Pokud se nejedná o kolizní záznam, tak je na základě extrahovaných položek z hlaviček paketů aktualizován kontext. Regulární výrazy se v aplikační vrstvě vyhledávají pouze v několika prvních paketech síťového toku.

Blokové schéma architektury FPU je naznačeno na obrázku 6.7. Nejdůležitějším blokem je aritmeticko-logická jednotka. Tato jednotka pracuje s kontextem a extrahovanými položkami z hlaviček paketů. Kontext muže být použit také z vnitřní paměti, ve které jsou

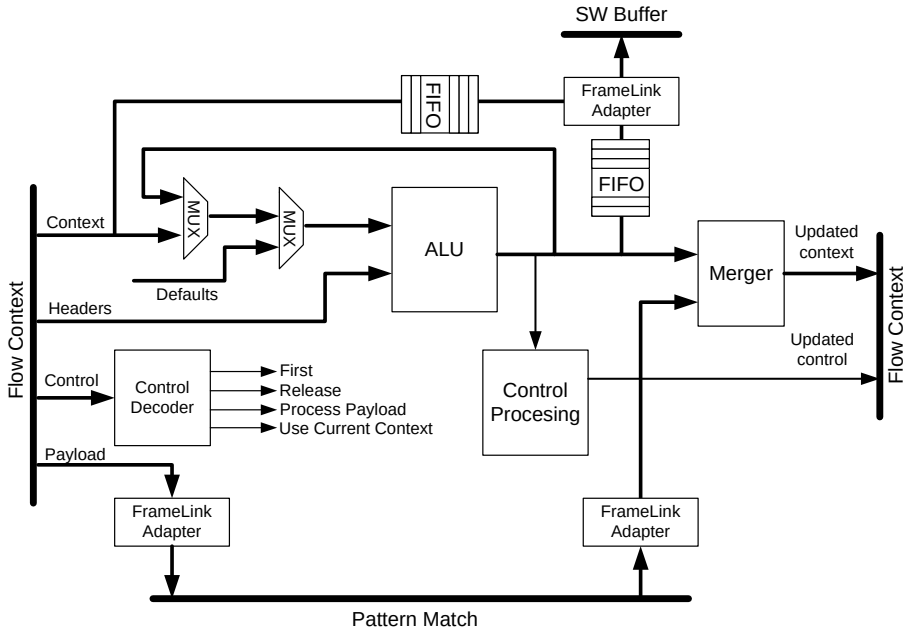


Obrázek 6.6: Control flow graf pro FPU

uloženy výchozí hodnoty kontextu při jeho zakládání. Při řetězeném zpracování dvou paketů ze stejného toku, je kontext použit přímo z výstupu ALU. Kontext se kromě zpracování v ALU také ukládá do front před SW buffrem. Na konci zpracování se rozhodne, jestli má kontext ve frontě zůstat nebo se má zahodit. Pokud řídící dekodér rozhodne o zpracování aplikačních dat, jsou tato data předána do vyhledávácí jednotky, jejichž architektura bude popsána v další kapitole. Tato jednotka vrátí bitový vektor nalezených regulárních výrazů, který je s využitím operace OR uložen do aktualizovaného kontextu.

Operace nad kontextem jsou potřebné nejen pro získávání statistik, ale také pro správu řídicích informací, podle kterých se rozhoduje o budoucím stavu kontextu. Mezi tyto informace patří zejména správa aktivního timeoutu (exportování kontextu po uplynutí určité doby po začátku toku), exportování kontextu po uzavření spojení (FIN flag), export před přetečením některých čítačů (počet odeslaných bytů), aktualizace počtu prohledaných paketů, aktualizace bitu platnosti kontextu a další. Tyto činnosti řídí Control processing jednotka. Výstupem této jednotky jsou jednak řídící signály pro FPU, ale také řídící informace, které později uloží do kontextu Return block.

Prakticky celá FPU jednotka je generována softwarově na základě XML schématu. V tomto schématu je definováno v jakém formátu jsou předány položky z extrahovaných hlaviček. Dále je ze uveden formát samotného kontextu a operace, které se mají nad kontextem provádět. Tento formát je univerzální a umožňuje jednoduché přidání prakticky libovolných statistických údajů, které lze poté použít k identifikaci aplikačních protokolů. Statistické údaje, které jsou vhodné pro HW zpracování a zároveň poskytují vysokou přesnost identifikace aplikačních protokolů byly představeny v kapitole 5. Popis principu generování je nad rámec této práce, a je blíže popsán v práci [30].



Obrázek 6.7: Architektura jednotky FPU

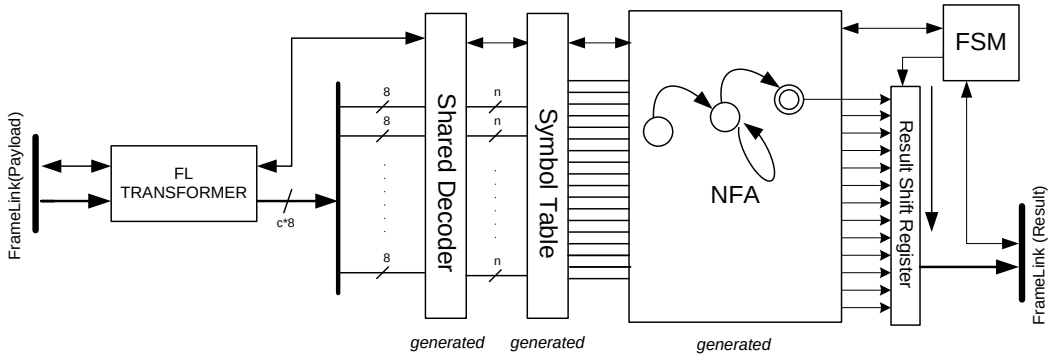
6.3.2 Jednotka pro vyhledávání regulárních výrazů

V této sekci bude prezentována architektura jednotky pro vyhledávání regulárních výrazů, která je nutná pro realizaci identifikace aplikačních protokolů na základě vyhledávání vzorů. Z prezentovaných přístupů pro vyhledávání vzorů byl zvolen přístup na bázi nedeterministických konečných automatů [44, 8, 9]. Tento přístup nabízí vysokou propustnost s možností hledat jak řetězce, tak pro identifikaci aplikací klíčové regulární výrazy. Kromě architektury vyhledávací jednotky bude také představen způsob transformace regulárních výrazů do hardwarové reprezentace. V závěru budou prezentovány parametry této implementace na platformě Virtex 2 Pro a Virtex 5.

Architektura jednotky pro vyhledávání vzorů je zachycena na obrázku 7.1. Díky generickému návrhu může být tato jednotka konfigurovaná pro libovolnou cílovou propustnost a pro použití se vstupním a výstupním rozhraním různé datové šířky. Část architektury je genericky popsána pomocí VHDL a druhá část realizující samotné hledání řetězců je generována softwarově.

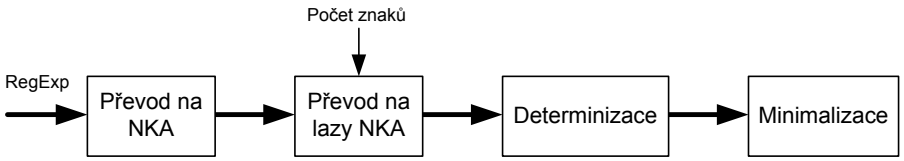
Vyhledávací jednotka pracuje s protokolem FrameLink, ve kterém očekává data, která se mají prohledávat na výskyt regulárních výrazů. Na vstupu jednotky je umístěn Transformer, který převádí prakticky libovolně široké datové rozhraní na rozhraní s šířkou, která odpovídá počtu bytů zpracovávaných jádrem jednotky. Nalezené regulární výrazy jsou označeny v bitovém vektoru, jehož délka odpovídá počtu právě vyhledávaných regulárních výrazů. Tento vektor je poslán na výstup jednotky ve FrameLink formátu.

Před popisem samotného jádra vyhledávací jednotky je nutné popsat, jak jsou transformovány regulární výrazy do reprezentace, která je vhodná pro mapování na hardware. Výsledná reprezentace je ve formátu konečného automatu, u kterého je potřeba minimalizovat počet stavů a přechodů. Z experimentů, při kterých byl zjišťován počet stavů a přechodů různých reprezentací výsledného automatu (lazy, deterministický, minimalizo-



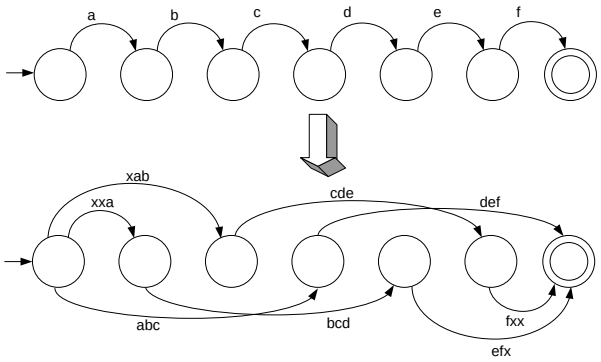
Obrázek 6.8: Architektura jednotky pro vyhledávání vzorů

vaný) vyplývá, že nejmenší počet stavů a přechodů má minimalizovaný deterministický automat, který je vytvořen nezávisle pro každý regulární výraz.



Obrázek 6.9: Transformace regulárního výrazu před mapováním na HW

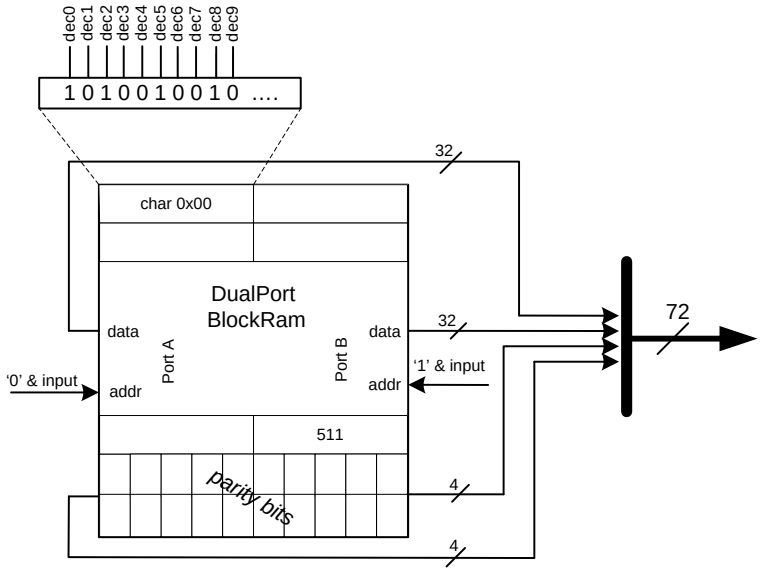
Pro každý regulární výraz je tedy provedena sekvence transformací znázorněna na obrázku 6.9. Nejprve je z regulárního výrazu pomocí parseru napsaného s využitím nástrojů flex a bison vytvořen nedeterministický konečný automat. Tento automat je dále převeden na tzv. lazy NDKA, který s jedním přechodem přijímá více znaků, jak je naznačeno na obrázku 6.10. Tato transformace výrazně zvyšuje propustnost vyhledávácí jednotky. Takto upravený automat je dále determinizován a minimalizován. Pro práci s automaty byly použity knihovny v jazyce C++ vyvinuté v rámci projektu Liberouter [31].



Obrázek 6.10: Převod na automat zpracovávající více znaků najednou

Jádro vyhledávací jednotky je tvořeno třemi bloky generovanými na základě předaných regulárních výrazů. První stupeň tvoří znakový dekodér. Lze si ho představit jako funkci,

která převádí každý znak na kód 1 z 256. Díky tomuto překódování je výskyt každého znaku reprezentován jediným bitem. Dekodér umí kromě znaků pracovat i se znakovými třídami - tedy nastaví odpovídající bit, pokud znak patří do dané třídy (např. alfanumerický znak).



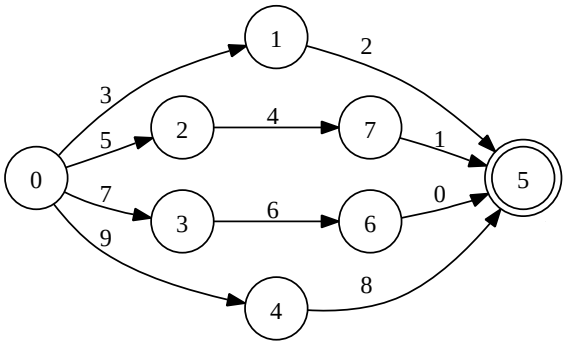
Obrázek 6.11: Znakový dekodér

Pro úspornou FPGA zdrojů, byl tento dekodér implementován pomocí BlockRam pamětí, integrovaných na FPGA čipu. S využitím paritních bitů, lze pomocí jediné paměti dekódovat až 72 znaků resp. znakových tříd. Experimenty ukazují, že pro množinu do 200 regulárních výrazů stačí pro dekódování 2 až 3 BlockRam paměti. Tento počet odpovídá zpracování jednoho znaku v hodinovém cyklu. Při zpracování n znaků se počet potřebných pamětí násobí konstantou n. Pro realizaci požadované funkce jsou BlockRam paměti inicializovány při generování dekodéru. Způsob inicializace je naznačen na obrázku 6.11.

Druhý blok vyhledávací jednotky tvoří tabulka symbolů. Tabulka symbolů je množina všech přechodů vyskytujících se v daném DKA (resp. NDKA). Na obrázku 6.12 je znázorněn automat pro identifikaci SSH protokolu přijímající 4 znaky v jednom kroku a odpovídající tabulka symbolů. Hardwarově je tabulka symbolů reprezentovaná n-vstupým AND členem, kde n odpovídá počtu najednou zpracovávaných znaků. Na vstupy tohoto logického členu jsou napojeny odpovídající výstupy znakového dekodéru. Výstup tabulky symbolů může být registrován pro dosažení vyšší frekvence.

Poslední blok představuje nejdůležitější část vyhledávací jednotky a to samotný automat. Vytvořená reprezentace automatu je mapována na HW zdroje stejným způsobem, který je naznačen na obrázku 6.13. Na obrázku je naznačena tabulka symbolů realizována AND členem a samotný automat. Pro každý stav automatu je vygenerován D-klopný obvod. Na vstup počátečního stavu je přivedena logická jednička a po resetu je odpovídající klopný obvod rovněž inicializován do této logické hodnoty. Koncové stavy jsou realizovány SR-klopnými obvody. Jakmile se na vstup tohoto registru dostane logická jednička, je v něm tato hodnota zachována do příchodu dalšího paketu.

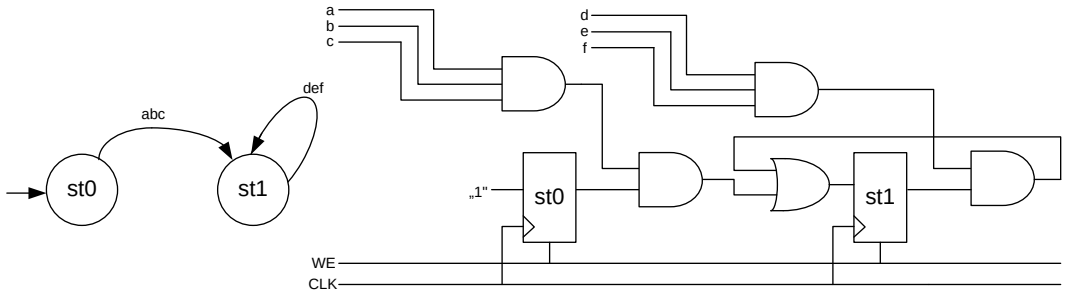
Pro každý přechod je generován dvou-vstupý logický člen AND. Do tohoto logického členu je přiveden odpovídající výstup z tabulky symbolů a výstup z klopného obvodu reprezentujícího stav, ze kterého přechod vychází. Výstup z logického členu AND je připojen



Symbol	Hodota			
0	[0-9]	X	X	X
1	.	[0-9]	X	X
2	[1-2]	.	[0-9]	X
3	[sS]	[sS]	[hH]	-
4	[sS]	[hH]	-	[1-2]
5	X	X	X	[sS]
6	[hH]	-	[1-2]	.
7	X	X	[sS]	[sS]
8	-	[0-1]	.	[0-9]
9	X	[sS]	[sS]	[hH]

Obrázek 6.12: Automat pro identifikaci SSH protokolů na 3.2 Gb/s (/ssh-[12]\.[0-9]/i)

na vstup následujícího stavu. V případě, že do cílového stavu vede více přechodů je vstupu předřazen logický člen *OR*. Podobným způsobem lze realizovat i epsilon přechody. Ty jsou ale už při převodu na rozšířený automat eliminovány, takže je není potřeba řešit.



Obrázek 6.13: Hardwarová reprezentace automatu

Výstup z registru reprezentujícího koncový stav automatu je přiveden na vstup posuvného registru, do kterého je tato hodnota zapsána po zpracování všech vstupních dat. Výsledný bitový vektor je postupně vysouván na výstup vyhledávácí jednotky. Paralelně s tím mohou být na výskyt regulárních výrazů prohledávaná další vstupní data.

Parametry jednotky pro vyhledávání regulárních výrazů

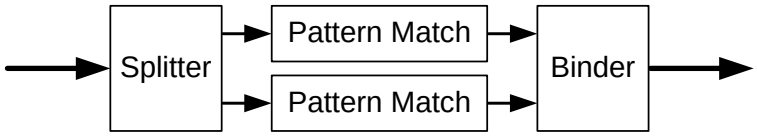
Navržená jednotka pro vyhledávání regulárních výrazů byla implementovaná v jazyce VHDL. Část zdrojových kódu je také generována softwarově na základě předaných regulárních výrazů. Funkčnost výsledné implementace byla ověřena v simulacích, kde byla jednotka schopna na základě regulárních výrazů identifikovat (WWW, SSH, SSL, SIP a SMTP) provoz. Výslednou realizaci jednotky na FPGA čipu je možné nalézt v příloze D.

Velmi důležitými parametry pro vyhledávání vzorů jsou dosažená propustnost a počet potřebných logických členů. Proto byla provedena evaluace implementace na FPGA čipech Virtex 2 Pro a Virtex 5 pro různý počet přijímaných znaků a pro různé množství regulárních výrazů. Do grafů byly zaznačené závislosti mezi dosaženou propustností, využitými zdroji na čipu a počtem najednou zpracovávaných znaků. Naměřené výsledky po place and route jsou v grafech shrnuty v příloze E.

Nejlepších výsledků se podařilo dosáhnout s novějšími čipy Xilinx Virtex 5. Pro malé množství regulárních výrazů (do 30) se podařilo dosáhnout propustnosti až 12 Gb/s. Při větším množství regulárních výrazů se propustnost pohybovala okolo 7 Gb/s. Tyto propustnosti představují až 400 násobné zrychlení oproti softwarové realizaci s využitím knihovny PCRE.

Dále byl analyzován poměr propustností k využitým zdrojům na FPGA čipu. Pro různé počty přijímaných znaků bylo analyzováno nejoptimálnější množství přijímaných znaků, aby byly maximálně využity zdroje na čipu. Z výsledků vyplývá, že optimální počet přijímaných znaků závisí na použité technologii FPGA. Čipy Virtex 2 a Virtex 5 se liší v parametrech generátorů logických funkcí (LUT). Virtex 5 obsahuje 6-vstupé LUT, kdežto Virtex 2 pouze 4-vstupé LUT. Pro technologii Virtex2 je tedy vhodné přijímat tři znaky najednou, kdežto pro Virtex 5 je to pět znaků. Maximální dosažená propustnost při tomto nastavení se pohybuje okolo 3.2 Gb/s (Virtex2) a 6.5 Gb/s (Virtex 5).

Pokud je potřeba vyšší propustnost, je nutné vzít v úvahu, že další zvyšování příjmu znaků nevede k odpovídajícímu nárůstu propustnosti, ale naopak vede k exponenciálnímu nárůstu využitých zdrojů. Propustnost je proto vhodné dále zvyšovat zapojením více vyhledávacích jednotek paralelně, jak je naznačeno na obrázku 6.14.



Obrázek 6.14: Škálovatelnost jednotky pro vyhledávání vzorů

6.4 Využití navrženého řešení v síťové infrastruktuře

Byla navržena HW architektura sondy, která dokáže kromě sbírání NetFlow záznamů také identifikovat aplikační protokoly s využitím vyhledání signatur a statistických vlastností. Sonda je schopna pracovat na 10 Gb/s sítích a může být rekonfigurována pro vyhledávání libovolných regulárních výrazů a sbírání libovolných statistik. Navržené řešení lze použít v řadě síťových aplikací, které jsou uvedeny dále.

- **Správa sítě**

  Možnost identifikace aplikačních protokolů přináší větší možnosti při monitorování a správě velkých sítí. Díky této vlastnosti je možné lépe plánovat topologii a kapacitu sítě. Je možné efektivněji rozdělit zátěž mezi více aplikačních serverů. Lze monitorovat aktivity uživatelů a dodržování firemních politik.

- **Traffic shaping**

  Sonda může být s výhodou využita při omezování nebo zvýhodňování některých komunikací v závislosti na typu aplikačního protokolu. Informace o síťových tocích se ve formě Netflow protokolu exportují na kolektor, kde jsou data zpracována a odpovídajícím způsobem je rekonfigurován aktivní síťový prvek podporující QoS. Nevýhodou tohoto přístupu může být dlouhá latence od detekce po odpovídající akci, proto je možné tento přístup použít jen pro správu dlouho trvajících toků (P2P, SSH, streaming atp.).

- **Blokování provozu na základě aplikací**

  Stejným způsobem jako lze síťové toky omezovat je možné je také blokovat. Zde je ovšem nutné brát zřetel na míru false-positives u detekce blokovaných protokolů. Navržené řešení umožňuje míru false-positives snížit kombinací dvou přístupů k detekci aplikačních protokolů.

- **Anomaly based IDS**

  Netflow statiky se také používají k detekci anomálií na síti [66]. Metody detekce nejčastěji fungují na principu vytváření modelu normálního chování. Komunikace, které se od tohoto modelu odchylují jsou označeny jako nebezpečné. Tímto způsobem lze detekovat celou řadu bezpečnostních hrozeb (skenování portů, DOS, DDOS, internetové červy). Přidání další vlastnosti (typ aplikačního protokolu) tyto metody vylepšuje a umožňuje jim detekovat celou řadu nových bezpečnostních hrozeb [15]. Např. detekce IRC protokolu na stanici, kde se historicky nevyskytoval může znamenat, že je stanice zneužívána k odesílání spamu nebo k DOS útokům.

- **Detekce šíření internetových červů**

  Vzhledem k výkonné vyhledávací jednotce umístěné v sondě je možné detekovat šíření internetových červů, ke kterým byla vytvořena signatura. Signatury obvykle obsahují regulární výraz, na jehož základě lze odhalit šíření internetového červa. Signatury je možné nalézt například na stránce s pravidly pro IDS systém Snort [48]. Množství podporovaných signatur je omezeno pouze kapacitou čipu.

- **Detekce exploitu, malware, bootnetu, atp.**

  Stejným způsobem lze detekovat celou řadu jiných bezpečnostních hrozeb, ke kterým existují odpovídající signatury.

Kapitola 7

Závěr

Byla nastudována problematika identifikace aplikačních protokolů na základě prohledávání obsahu a statistické analýzy. V souvislosti s tím byla nastudována problematika klasifikačních metod a rychlého vyhledávání vzorů. Dále byly analyzovány existující volně dostupné programy k omezování provozu podle typu aplikační vrstvy. Na základě této analýzy byly zjištěny nedostatky nynějších přístupů identifikace aplikačních protokolů, které jsou zejména nízká propustnost a nedostatečná přesnost. Tyto nedostatky mohou být odstraněny hardwarovou akcelerací kritických operací a kombinací několika přístupů identifikace aplikačních protokolů.

Aby bylo možné akcelerovat nynější přístupy pro identifikaci protokolů, byla zkoumána úzká místa, která brání použití těchto metod na multigigabitových sítích. Pro tento účel byl vytvořen model, který modeluje operace prováděné nynějšími metodami identifikace protokolů. S využitím simulací a testů volně dostupného software (Snort, L7 filter) byly určeny limity jednotlivých operací. Limitující byla zejména činnost vyhledávání regulárních výrazů, která omezuje použití metod založených na vyhledávání signatur pouze na 100 Mb/s sítě. Omezující z hlediska 10 Gb/s sítí je operace získávání statistických údajů o síťových tocích, která je potřená pro identifikaci protokolů na základě statistických vlastností. Nynější softwarová řešení umožňují provádět tuto činnost pouze na 1 Gb/s sítích.

Na základě analýzy výpočetní náročnosti bylo provedeno rozdělení jednotlivých operací mezi hardware a software s cílem akcelerovat identifikaci aplikačních protokolů na 10 Gb/s sítě. Na základě tohoto rozdělení byla navržena hardwarová architektura sondy pro identifikaci aplikačních protokolů. Navržené řešení vychází z architektury sondy FlowMon, která byla rozšířena zejména o jednotku pro rychlé vyhledávání regulárních výrazů. Navržený systém je schopen vyhledávat regulární výrazy až na 10 Gb/s, sbírat relevantní statistiky pro přesnou identifikaci na základě statistických vlastností a s využitím klasifikátoru založeného na rozhodovacích stromech dosáhnout přesnosti identifikace aplikačních protokolů okolo 95 %.

Navržené řešení lze využít v celé řadě síťových aplikací jako je trafic shaping, blokování aplikačních protokolů, anomaly based IDS systémy, detekce internetových červů, exploitu, malware, bootnetů atd. Další možné použití sondy je popsáno na konci kapitoly 6.

Vlastnosti navrženého řešení je možné dále zlepšovat zvyšováním přesnosti identifikace protokolů důkladnější analýzou přenášených aplikačních dat. Toho lze docílit zejména využitím TCP reassemblingu a parsováním některých aplikačních protokolů. Dále je možné se zabývat také problematikou získávání anotovaného síťového provozu, tak aby jeho použití při trénování klasifikátoru vedlo k vysoké přesnosti identifikace bez ohledu na lokalitu, kde bude sonda nasazena.

Literatura

[1] S. Astashonok. fprobe - netflow probe. `http://fprobe.sourceforge.net/`, prosinec 2007.

[2] Z. K. Baker and V. K. Prasanna. A Methodology for the Synthesis of Efficient Intrusion Detection Systems on FPGAs. In *Proceedings of the Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2004 (FCCM '04)*, 2004.

[3] Z. K. Baker and V. K. Prasanna. Automatic Synthesis of Efficient Intrusion Detection Systems on FPGAs. In *Proceedings of the 14th Annual International Conference on Field-Programmable Logic and Applications (FPL '04)*, 2004.

[4] Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on fpgas. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 223–232, New York, NY, USA, 2004. ACM Press.

[5] CacheLogic. Cachepliance 4000. `http://www.grazierenterprises.com/products/cachelogic/support/cp4000ds.pdf`, prosinec 2007.

[6] Y. H. Cho and W. H. Mangione-Smith. Deep Packet Filter with Dedicated Logic and Read Only Memories. In *12th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, pages 125–134, Napa, CA, 2004.

[7] Inc. Cisco Systems. Network-based application recognition and distributed network-based application recognition. `http://www.cisco.com`, prosinec 2007.

[8] Ch. Clark and D. Schimmel. Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *Field Programmable Logic and Application, 13th International Conference*, pages 956–959, Lisbon, Portugal, 2003.

[9] Ch. Clark and D. Schimmel. Scalable Pattern Matching for High-Speed Networks. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 249–257, Napa, California, 2004.

[10] L. Deri. nprobe - an extensible netflow v5/v9/ipfix gpl probe for ipv4/v6. `http://www.ntop.org/nProbe.html/`, prosinec 2007.

[11] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro*, 24(1):52–61, 2004.

[12] S. Dharmapurikar and V. Paxson. Robust tcp stream reassembly in the presence of adversaries. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[13] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[14] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, 2 edition, 2000.

[15] J. P. Early, C. E. Brodley, and C. Rosenberg. Behavioral authentication of server flows. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 46, Washington, DC, USA, 2003. IEEE Computer Society.

[16] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286, New York, NY, USA, 2006. ACM.

[17] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 883–892, New York, NY, USA, 2007. ACM.

[18] S. Golson. State Machine Design Techniques for Verilog and VHDL. *Synopsys Journal of High-Level Design*, pages 1–48, 1994.

[19] P. Haffner, S. Senand O. Spatscheck, and D. Wang. Acas: automated construction of application signatures. In *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202, New York, NY, USA, 2005. ACM.

[20] J. Han and M. Kamber. *Data Mining, Second Edition, Second Edition : Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems) (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, January 2006.

[21] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 9–9, Berkeley, CA, USA, 2001. USENIX Association.

[22] Cisco CNS NetFlow Collection Engine Installation and 3.0 Configuration Guide. Netflow export datagram format. `http://www.cisco.com/en/US/docs/net_mgmt/netflow_collection_engine/3.0/user/guide/nfcform.pdf`, prosinec 2007.

[23] Ipoque. Hardware p2p filter based on a protocol based classification of p2p traffic to shape or block p2p, voip and skype traffic. `http://www.ipoque.com/`, prosinec 2007.

[24] H. Jiang, A. W. Moore, S. Jin, and J. Wang. Lightweight application classification for network management. In *Proceedings of the SIGCOMM Workshop on Internet Network Management 2007: The Five-Nines Workshop*, August, 2007.

[25] H-J Jung, Z. K. Baker, and V. K. Prasanna. Performance of FPGA Implementation of Bit-split Architecture for Intrusion Detection Systems. In *Proceedings of the Reconfigurable Architectures Workshop at IPDPS (RAW '06)*, 2006.

[26] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy. Transport layer identification of p2p traffic, 2004.

[27] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput*, 6(2):323–350, 1977.

[28] M. Košek and J.Kořenek. Flowcontext: Flexible platform for multigigabit stateful packet processing. In *2007 International Conference on Field Programmable Logic and Applications*, pages 804–807. IEEE Computer Society, 2007.

[29] Y. Lei and L. Huan. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of ICML*, pages 856–863, 2003.

[30] O. Lengál. Automated Generation of Processing Elements for FPGA. *bakalářská práce, Brno, FIT VUT v Brně*, 2008.

[31] Liberouter. Liberouter Project WWW Page. http://www.liberouter.org, 2006.

[32] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis. Exclusion-Based Signature Matching for Intrusion Detection. In *IASTED International Conference on Communication and Computer Network (CCN'021)*, 2002.

[33] T. Martínek and M. Košek. Netcope: Platform for rapid development of network applications. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, 2008.

[34] A. Mcgregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *Fifth Passive and Active Measurement Workshop*, April 2004.

[35] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. In *Proceedings of the Passive and Active Measurement Workshop (PAM2005)*, March/Apri 2005.

[36] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, New York, NY, USA, 2005. ACM Press.

[37] A. W. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical Report, RR-05-13, Department of Computer Science, Queen Mary, University of London, August, 2005.

[38] Packeteer. Packetshaper. `http://www.packeteer.com/products/packetshaper/`, prosinec 2007.

[39] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148, New York, NY, USA, 2004. ACM.

[40] A. Ganapathiraju S. Balakrishnama. Linear discriminant analysis - a brief tutorial. http://lcv.stat.fsu.edu/research/geometrical_representations_of_faces/PAPERS/lda_theory.pdf, prosinec 2007.

[41] SCAMPI. SCAMPI (IST-2001-32404) Project WWW Page. http://www.ist-scampi.org, 2005.

[42] L. Schaelicke, T. Slabach, B. Moore, and C.Freeland. Charakterizing the Performance of Network Intrusion Detection Sensors. In *Sixth International Symposium on Recent Advances in Intrusion Detection (RAID'03)*, 2003.

[43] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 512–521, New York, NY, USA, 2004. ACM.

[44] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching using FPGAs. In *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238, April 2001.

[45] H. Song, T. S. Sproull, M. Attig, and J. W. Lockwood. Snort offloader: A reconfigurable hardware nids filter. In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications (FPL '05)*, pages 493–498, 2005.

[46] I. Sourdis and D. N. Pnevmatikatos. Fast, Large-Scale String Match for a 10Gbps FPGA-Based Network Intrusion Detection System. In *Field Programmable Logic and Application, 13th International Conference*, pages 880–889, Lisbon, Portugal, 2003.

[47] WWW stránky. Pcre - perl compatible regular expressions. http://www.pcre.org, duben 2008.

[48] WWW stránky. Bleeding edge threats. http://www.bleedingthreats.net/rules/, květen 2008.

[49] WWW stránky. Application layer packet classifier for linux. http://l7-filter.sourceforge.net/, prosinec 2007.

[50] WWW stránky. Bro intrusion detection system. http://www.bro-ids.org/, prosinec 2007.

[51] WWW stránky. Hi-performance protocol identification engine. http://hippie.oofle.com/, prosinec 2007.

[52] WWW stránky. Introduction to cisco ios netflow - a technical overview. http://www.cisco.com/en/US/products/ps6601/products_white_paper0900aecd80406232.shtml , prosinec 2007.

[53] WWW stránky. Linux advanced routing and traffic control. http://lartc.org/, prosinec 2007.

[54] WWW stránky. Netfilter/iptables project homepage. http://www.netfilter.org, prosinec 2007.

[55] WWW stránky. Netflow version 9 flow-record format. http://www.cisco.com/en/US/products/ps6601/products_white_paper09186a00800a3db9.shtml, prosinec 2007.

[56] WWW stránky. Official ipp2p homepage. http://ipp2p.org/, prosinec 2007.

[57] WWW stránky. Snort - the de facto standard for intrusion detection/prevention. http://www.snort.org/, prosinec 2007.

[58] WWW stránky. Weka 3 - data mining with open source machine learning software in java. http://www.cs.waikato.ac.nz/ml/weka/, prosinec 2007.

[59] L. Tan, B. Brotherton, and T. Sherwood. Bit-split string-matching engines for intrusion detection and prevention. *ACM Trans. Archit. Code Optim.*, 3(1):3–34, 2006.

[60] J. Tobola. Vyhledávání řetězců v payloadu paketu s využitím TCAM. *Fakulta informačních technologií, Vysoké učení technické v Brně*, 2005.

[61] Xilinx. Virtex-II Platform FPGA User Guide, březen 2005. Dokument dostupný na http://direct.xilinx.com/bvdocs/userguides/ug012.pdf.

[62] Xilinx. Virtex-II Pro Platform FPGAs: Complete Data Sheet, říjen 2005. Dokument dostupný na http://direct.xilinx.com/bvdocs/publications/ds083.pdf.

[63] D. Zuev and A. W. Moore. Traffic classification using a statistical approach. In *Proceedings of Sixth Passive and Active Measurement Workshop (PAM 2005)*, March/April 2005.

[64] M. Žádník. Design of Flow Monitoring Probe. *Fakulta informačních technologií, Vysoké učení technické v Brně*, 2007.

[65] P. Čeleda, M. Žádník, and V. Krmíček. Monitorování provozu ve vysokorychlostních sítích na bázi ip toku. In Širokopásmové sítě a jejich aplikace. Olomouc: Univerzita Palackého v Olomouci, 2007.

[66] M. Řehák, M. Pěchouček, P. Čeleda, and V. Krmíček. Camnep: An intrusion detection system for high-speed networks. In *Progress in Informatics*, pages 65–74, 2008.

Příloha A

Atributy síťového provozu vhodné ke statistické identifikaci aplikačních protokolů v síťových tocích s ohledem na snadnou hardwarovou realizaci.

Číslo atrib.	Název	Popis
1	duration	Trvání spojení
2	total_packets_ab	Celkový počet paketů pozorovaných mezi klientem a serverem
3	ack_pkts_sent_ab	Celkový počet paketů s nastaveným TCP ACK bitem pozorovaných mezi klientem a serverem
4	pure_acks_sent_ab	Celkový počet paketů s nastaveným TCP ACK bitem pozorovaných mezi klientem a serverem, které neobsahují žádný TCP payload
5	actual_data_pkts_ab	Celkový počet paketů s alespoň jedním přenášeným bytem v TCP payloadu pozorovaných mezi klientem a serverem
6	actual_data_bytes_ab	Celkový počet bytů přenesených mezi klientem a serverem
7	outoforder_pkts_ab	Celkový počet paketů mimo pořadí přenesených mezi klientem a serverem
8	pushed_data_pkts_ab	Celkový počet paketů s nastaveným TCP PUSH bitem pozorovaných mezi klientem a serverem
9	SYN_pkts_sent_ab	Celkový počet paketů s nastaveným TCP SYN bitem pozorovaných mezi klientem a serverem
10	FIN_pkts_sent_ab	Celkový počet paketů s nastaveným TCP FIN bitem pozorovaných mezi klientem a serverem
11	urgent_data_pkts_ab	Celkový počet paketů s nastaveným TCP URG bitem pozorovaných mezi klientem a serverem
12	urgent_data_bytes_ab	Celkový počet bytů přenesených v paketu s nastaveným TCP ACK bitem pozorovaných mezi klientem a serverem
13	max_win_adv_ab	Maximální velikost window pozorovaná mezi klientem a serverem
14	min_win_adv_ab	Minimální velikost window pozorovaná mezi klientem a serverem
15	zero_win_adv_ab	Počet paketů pozorovaných mezi klientem a serverem s nulovou velikostí window

Číslo atrib.	Název	Popis
16	avg_win_adv_ab	Průměrná velikost window pozorovaná mezi klientem a serverem
17	min_data_wire_ab	Minimální počet bytů v Ethernetových paketech pozorovaných mezi klientem a serverem
18	mean_data_wire_ab	Průměrný počet bytů v Ethernetových paketech pozorovaných mezi klientem a serverem
19	max_data_wire_ab	Maximální počet bytů v Ethernetových paketech pozorovaných mezi klientem a serverem
20	var_data_wire_ab	Rozptyl počet bytů v Ethernetových paketech pozorovaných mezi klientem a serverem
21	min_data_ip_ab	Minimální počet bytů v IP paketech pozorovaných mezi klientem a serverem
22	mean_data_ip_ab	Průměrný počet bytů v IP paketech pozorovaných mezi klientem a serverem
23	max_data_ip_ab	Maximální počet bytů v IP paketech pozorovaných mezi klientem a serverem
24	var_data_ip_ab	Rozptyl počtu bytů v IP paketech pozorovaných mezi klientem a serverem
25	min_data_control_ab	Minimální počet kontrolních bytů v paketech pozorovaných mezi klientem a serverem
26	mean_data_control_ab	Průměrný počet kontrolních bytů v paketech pozorovaných mezi klientem a serverem
27	max_data_control_ab	Maximální počet kontrolních bytů v paketech pozorovaných mezi klientem a serverem
28	var_data_control_ab	Rozptyl počtu kontrolních bytů v paketech pozorovaných mezi klientem a serverem
29	min_IAT_ab	Minimální mezipaketová mezera pozorovaná mezi klientem a serverem
30	mean_IAT_ab	Průměrná mezipaketová mezera pozorovaná mezi klientem a serverem
31	max_IAT_ab	Maximální mezipaketová mezera pozorovaná mezi klientem a serverem
32	var_IAT_ab	Rozptyl mezipaketových mezer pozorovaných mezi klientem a serverem
33-63	-"-_ba	-"- mezi serverem a klientem

Příloha B

Ukázka části rozhodovacího stromu pro klasifikaci síťového provozu vytvořeného programem See5 s využitím atributů popsaných v příloze A.

```
pushed_data_pkts_ab > 2:
:...total_packets_ab > 520:
:   :...max_win_adv_ab <= 38640: FTP-DATA (98/3)
:   :   max_win_adv_ab > 38640: DATABASE (4)
:   total_packets_ab <= 520:
:   :...mean_IAT_ab > 1.824802:
:       :...min_IAT_ab > 0.000222:
:       :    :...zero_win_adv_ab > 0: WWW (2)
:       :    : zero_win_adv_ab <= 0:
:       :    :    :...max_IAT_ab <= 76.92414: P2P (120/4)
:       :    : max_IAT_ab > 76.92414: MULTIMEDIA (4/1)
:       :    min_IAT_ab <= 0.000222:
:       :    :...max_win_adv_ab > 17040:
:       :        :...max_data_wire_ab <= 83: FTP-CONTROL (2)
:       :        : max_data_wire_ab > 83: MAIL (46/7)
:       :        max_win_adv_ab <= 17040:
:       :        :...min_win_adv_ab <= 12328: FTP-CONTROL (7/1)
:       :            min_win_adv_ab > 12328:
:       :            :...pure_acks_sent_ab <= 2: MULTIMEDIA (2)
:       :                pure_acks_sent_ab > 2: P2P (4)
:       mean_IAT_ab <= 1.824802:
:       :...pure_acks_sent_ab <= 3:
:           :...min_IAT_ab <= 0.000196:
:           :   :...actual_data_bytes_ab > 118446:
:           :   :   :...min_win_adv_ab > 20426: MAIL (24)
:           :   :   :   min_win_adv_ab <= 20426:
:           :   :   :   :...max_win_adv_ab <= 16560: P2P (3/1)
:           :   :   :       max_win_adv_ab > 16560: FTP-DATA (4)
:           :   :   actual_data_bytes_ab <= 118446:
:           :   :   :...pushed_data_pkts_ab > 5: MAIL (2504/9)
:           :   :       pushed_data_pkts_ab <= 5:
:           :   :       :...FIN_pkts_sent_ab > 2: FTP-DATA (4)
:           :   :           FIN_pkts_sent_ab <= 2:

atd.
```

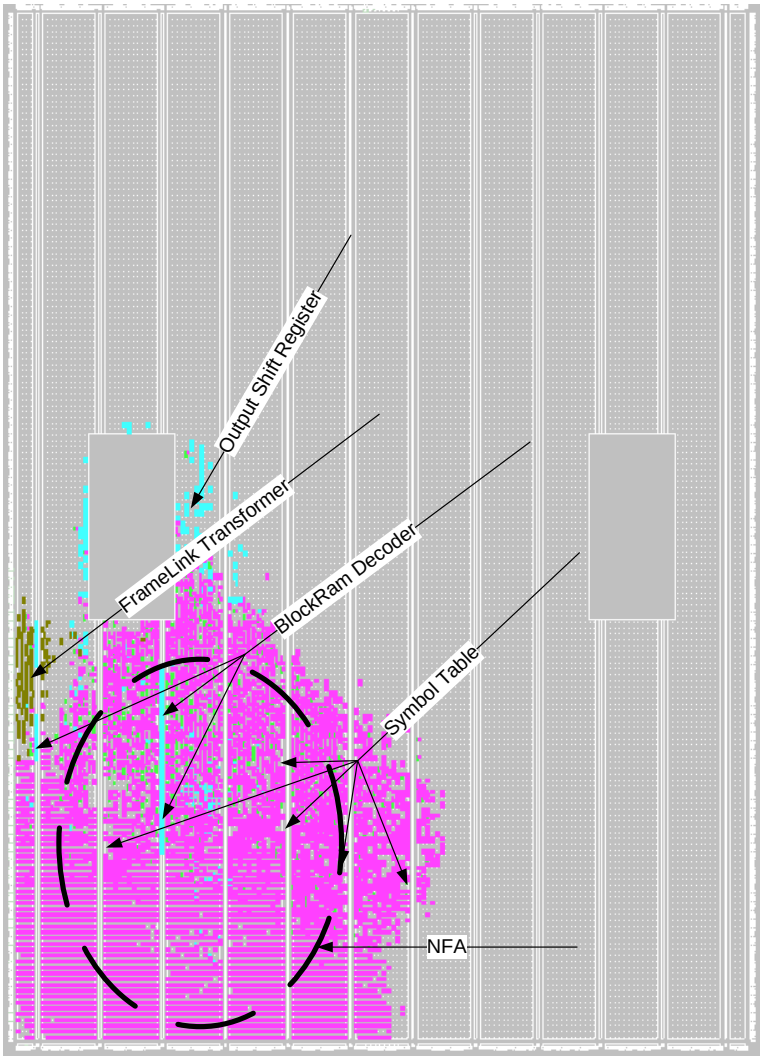
Příloha C

Položky hlaviček protokolů na L2-L3 vrstvě potřebné k identifikaci protokolů. Vybrané položky jsou seřazeny do jednotlivých kategorií. Do položek nejsou zahrnuty informace o délce a začátku jednotlivých vrstev ISO/OSI, které se musí dopočítat v závislosti na typech protokolů. Pro výpočet statistických vlastností jsou také použity položky uvedené v jiných kategoriích (délky jednotlivých vrstev, tcp_flags atd.)

Číslo položky.	Název	Velikost [bit]	Popis
Identifikace toku			
1	ipv4_protocol ipv6_next_header	8	Typ protokolu na vyšší vrstvě
2	ip_src_ip	32 / 128	Zdrojová IP adresa
3	ip_dst_ip	32 / 128	Cílová IP adresa
4	tcp_udp_src_port	16	Zdrojový port
5	tcp_udp_dst_port	16	Cílový port
IPv4 fragmentace			
6	ipv4_identification	16	Identifikace fragmentu
7	ipv4_flags	3	Informace o fragmentaci
8	ipv4_fragment_offset	13	Umístění fragmentu ve složeném paketu
Skládání TCP toků			
9	tcp_seq_number	32	TCP sequence number
10	tcp_ack_number	32	TCP acknowledgment number
11	tcp_control_bits	6	TCP Flags (U,A,P,R,S,F)
Statistické vlastnosti			
12	tcp_window	32	Window size

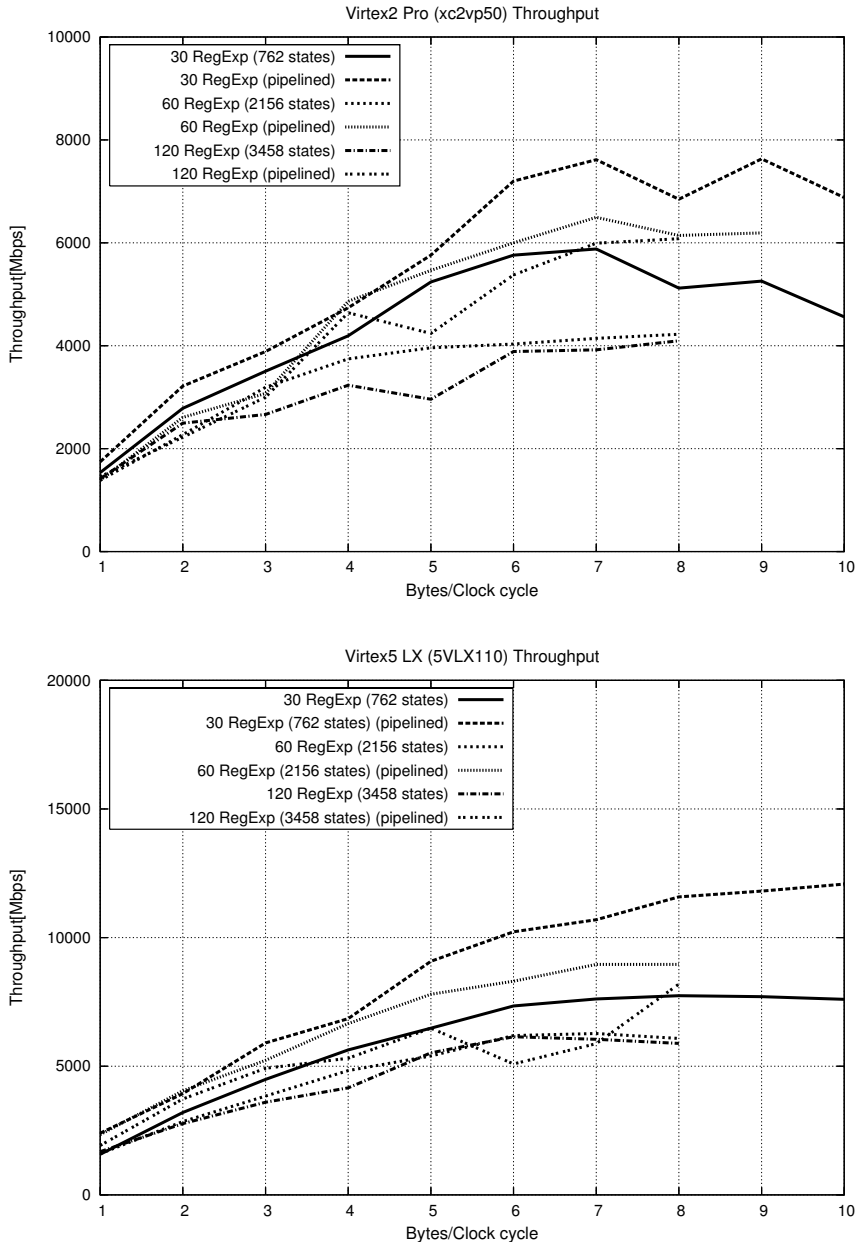
Příloha D

Namapování jednotky pro vyhledávání regulárních výrazu na čip Virtex2 Pro (xc2vp50). Vyhledávácí jednotka byla nakonfigurována pro vyhledávání 200 regulárních výrazů na rychlosti 1.6 Gb/s.

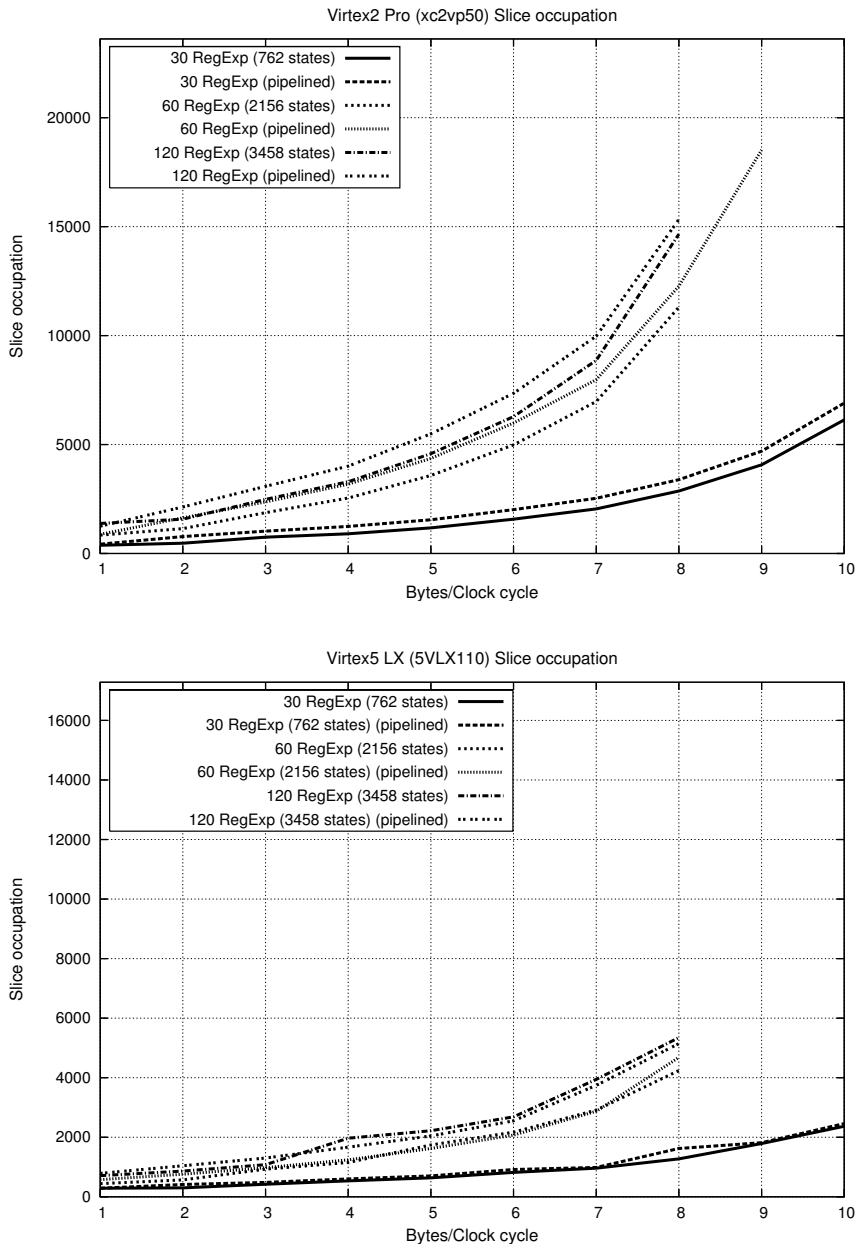


Obrázek 7.1: FloorPlanner pro vyhledávací jednotku

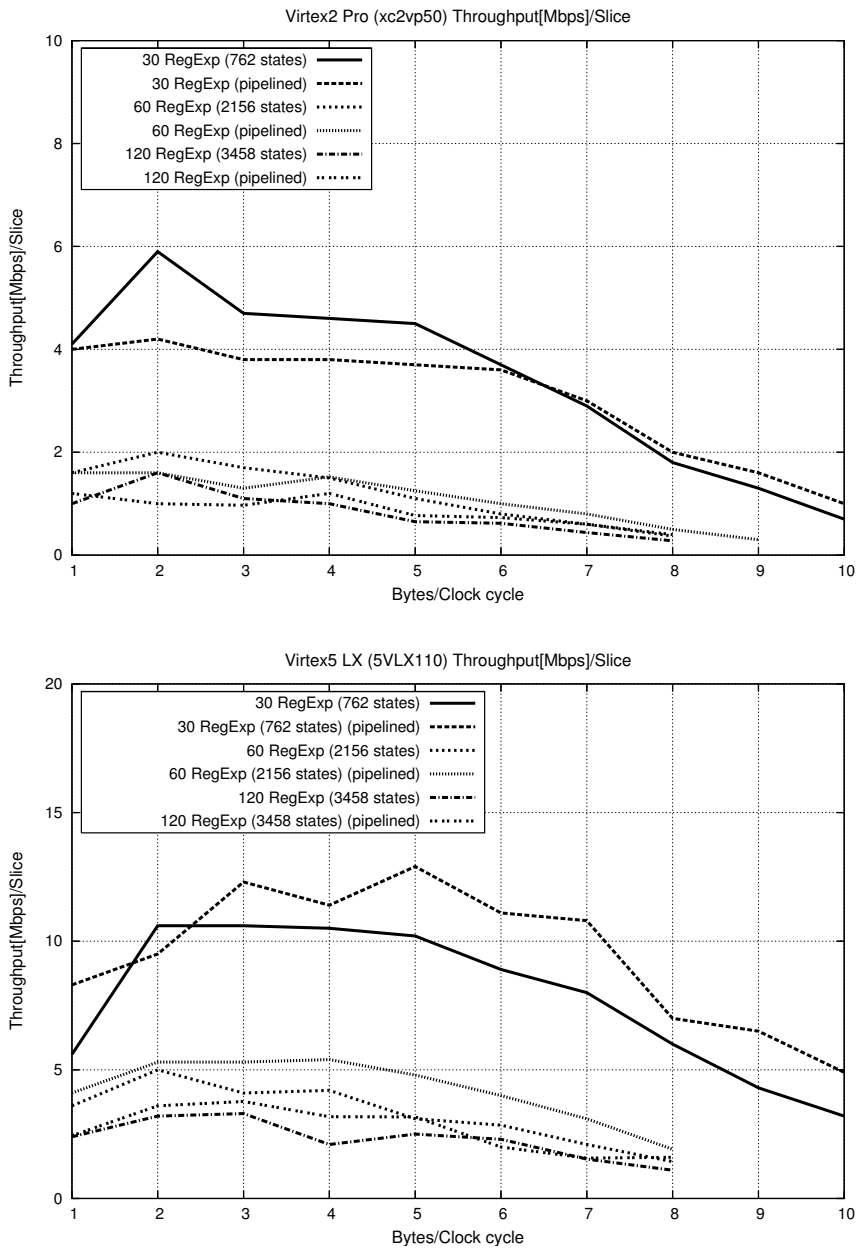
Příloha E



Obrázek 7.2: Dosažená propustnost jednotky pro vyhledávání regulárních výrazů



Obrázek 7.3: Využité zdroje FPGA



Obrázek 7.4: Poměr propustnost / využité zdroje