VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

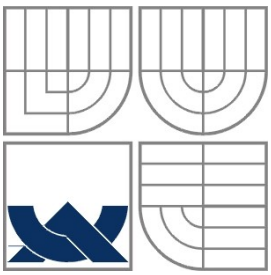
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HW AKCELERACE UNIXOVÝCH SÍŤOVÝCH NÁSTROJŮ

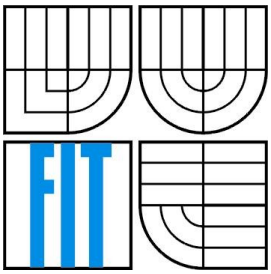
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETER BARTOŠ
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

HW AKCELERACE UNIXOVÝCH SÍŤOVÝCH NÁSTROJŮ

HW ACCELERATION OF NETWORK UNIX TOOLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETER BARTOŠ
AUTHOR

VEDOUCÍ PRÁCE — ING. JAN KOŘENEK
SUPERVISOR

BRNO 2009

Abstrakt

Na světě se objevují stále rychlejší technologie pro síťovou komunikaci. Některé síťové nástroje nejsou schopné pracovat při tak vysokých rychlostech, nadměrně zatěžují systém, takže nestíhají vykonávat potřebné funkce. Nedokážou dokonale monitorovat sítě a zajistit bezpečný provoz. Práce podrobně analyzuje síťové nástroje, jejich operace a hledá kritická místa pro budoucí hardwarovou akceleraci. V souvislosti s tím představuje efektivní platformy pro hardwarové programování. Na základě měření vyhodnocuje omezení nástrojů a poukazuje na možnosti akcelerace, které jsou návrhem pro další práci.

Abstract

In the world, there always appear faster technologies for network communication. Some network tools are not capable of working in high-speed, they are overloading system, therefore they are not able to fulfill their functions. They can not fully monitor the whole traffic and ensure secured services. Thesis analyses network tools, their operations and researches critical spaces for future hardware acceleration. In this context, it introduces effective hardware programming platforms. Using measurements, it evaluates the limits of the tools and mentions the possibilities of acceleration, which are suggested for another thesis.

Klíčová slova

síťové nástroje, operace, analýza, profilování, propustnost, platformy, hardvérová akcelerace

Keywords

network tools, operations, analysis, profiling, throughput, platforms, hardware acceleration

Citace

Peter Bartoš: HW akcelerace unixových síťových nástrojů, bakalářská práce, Brno, FIT VUT v Brně, 2009

HW akcelerace unixových síťových nástrojů

Prohlášení

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

.....
Peter Bartoš
20.5.2009

Poděkování

Chcel by som sa poďakovať Ing. Janu Kořenkovi za užitočné rady a za čas venovaný konzultáciám pri vedení mojej bakalárskej práce. Ďakujem aj kolegom z projektu Liberouter za pomoc pri riešení problémov počas riešenia práce. V neposlednom rade sa chcem poďakovať mojej rodine a priateľom, ktorí mi boli oporou počas štúdia.

Obsah

1 Úvod

V súčasnosti sa siete neustále vyvíjajú a dosahujú stále väčšie rýchlosti. Sú využívané obrovským množstvom ľudí a aplikácií, ktoré vyžadujú bezpečný prenos dát. Stále však existujú hrozby v podobe škodlivých aplikácií, nepriateľských užívateľov, ktoré môžu narušiť bezpečnú prevádzku.

Na monitorovanie, diagnostikovanie, zabezpečenie sietí existuje množstvo nástrojov. Vďaka nim je možné sieť kontrolovať, merať rôzne informácie o prevádzke a zabezpečiť jej správny chod. Problémom, ktorý sa objavuje, je nemožnosť fungovania týchto nástrojov pri rýchlostiach, ktoré siete dnes dosahujú. Aj napriek rýchlemu vývoju hardvéru, dnes sieťové nástroje nie sú schopné účinne spracovať celú sieťové prevádzku a nadmerne zaťažujú systém počítača. Taktiež kvôli narastajúcemu počtu užívateľov sieťových služieb, rastú nároky na samotné spracovávanie, zabezpečenie prenosu dát. Hardvérová akcelerácia mnohých funkcií sa už v blízkej dobe ukazuje ako nevyhnutná.

Účelom tejto práce by mala byť podrobná analýza sieťových nástrojov s úmyslom objaviť najkritickejšie miesta aplikácií, vhodné na hardvérovú akceleráciu. Je potrebné namerať zaťaženie jednotlivých častí nástrojov na systém, odmerať priepustnosť ich funkcií. Tak bude možné vypočítať nárast výkonu a dať tak námet na realizáciu akcelerácie.

Po úvode, v 2. kapitole práca zhrňuje súčasné najpoužívanejšie sieťové nástroje, poukazuje na ich jednotlivé funkcie a operácie. Mala by zhrnúť aj prostriedky, využívané týmito aplikáciami a odhadnúť náročnosť nástrojov na jadro systému. V 3. kapitole sú popísané platformy, ktoré ponúkajú efektívne možnosti pre hardvérové riešenia. Umožňujú implementovať rôzny sieťový hardvér využitím vývojových nástrojov, simulovať jeho funkcionalitu a otestovať pred konečnou realizáciou. Aby bolo možné nástroje účinne analyzovať, je nevyhnuté vytvoriť potrebné softvérové vybavenie. Tým sa zaoberá 4. kapitola, ktorá detailne popíše postup a výsledky testovania. Výstupom práce by mali byť podrobne analyzované operácie sieťových nástrojov a nástrel návrhu hardvérovej akcelerácie, ktorý by mal byť základom pre ďalšiu prácu. Obsahom 5. kapitoly je zhodnotenie výsledkov testov a výpočet možností akcelerovania nástrojov. V závere by som mal zhrnúť celkové dosiahnuté výsledky a prínosy tejto práce. Určiť, či práca splňuje zadanie a opäť poukázať na nedostatky sieťových nástrojov a odporučiť ich akcelerovanie.

2 Sieťové nástroje

V dnešnom informatickom veku sa siete enormne rozširujú. Už to nie je ako v polovici 80. rokov, keď sa objavili prvé osobné počítače, kedy väčšina ľudí neuvažovala o ich prepojení. Dnes už sa ťažko nájde organizácia, ktorá by počítače nemala prepojené. Zároveň sú ich siete začlenené do globálneho *Internetu*. Kvôli tomu sa objavujú stále rýchlejšie technológie pre komunikáciu. Vždy však existuje hrozba nepriateľských užívateľov, útokov na siete, ale aj nové zákony týkajúce sa bezpečnosti. Práve tieto faktory spôsobujú vysoko narastajúce nároky na spoľahlivosť a z toho vyplývajúcu potrebu správu sietí.

Riešením týchto požiadavok sú rôzne softvérové aj hardvérové nástroje slúžiace predovšetkým na administráciu a diagnostiku sietí. Ponúkajú rozličné funkcie ale mnohé sú limitované rýchlosťou spracovania. V ďalšej časti práce bude nasledovať stručný popis existujúcich nástrojov spolu s rozpisom ich operácií.

2.1 Prehľad sieťových nástrojov

Na Internete je možné nájsť stovky až tísíce sieťových nástrojov ponúkané pod rozličnými licenciami, vybavené rôznymi funkciami. Pekný prehľad najpoužívanejších nástrojov ponúka kniha [1], na ktorej si zakladá táto kapitola alebo elektronický zdroj [2], kde je podľa ankety pravidelne zostavený zoznam 100 najobľúbenejších nástrojov. Stránka [3] ponúka vyše 2000 užitočných sieťových programov na stiahnutie.

Podľa týchto zdrojov je možné sieťové nástroje rozdeliť do niekoľkých základných kategórií podľa ich funkcie. Jednotliví zástupcovia sa potom líšia výkonnosťou, druhom užívateľského rozhrania (grafické, textové) a licenciou. Väčšina rozoberaných nástrojov bude softvér s otvoreným kódom – *Open source software* [4], čo prináša medzi inými aj výhodu použitia zdarma.

Základné kategórie sieťových nástrojov sú: programy pre správu zariadení v sieti, nástroje pre dohľad služieb, niekoľko najzákladnejších diagnostických, konfiguračných nástrojov, aplikácie na analýzu sieťovej prevádzky na úrovni paketov, systémy pre detekciu útokov a nástroje pre zber informácií o podrobnostiach sieťovej prevádzky.

2.1.1 Vzdialená správa zariadení v sieti

Na vzdialenú správu sieťových zariadení sa používa štandardizovaný protokol *SNMP,* ktorého skratka predstavuje názov *Simple Network Management Protocol.* Tento výraz vystihuje jeho vlastnosti, pretože jeho architektúra je skutočne jednoduchá, preto je implementovaný takmer vo všetkých

zariadeniach pripojených v sieti. Riadi sa štandardom a preto je možné prostredníctvom protokolu *SNMP* komunikovať so všetkými zariadeniami jednotným spôsobom. Zariadenie sa spravuje pomocou základných operácií, ktoré umožňujú buď získať informácie od zariadenia, alebo poslať príkaz na zmenu konfigurácie.

V súčasnosti existuje protokol *SNMP* v troch verziách, pričom prvé dve využívajú na overenie totožnosti názov komunity. Keďže *SMTP* využíva protokol *UDP* bez zabezpečenia, znamená to, že je možné tento prihlasovací údaj jednoducho odchytiť. Preto je dôležité zvážiť prípadné použitie protokolu verzie 3, ktorá už ponúka skutočný autentifikačný mechanizmus.

Open-source predstaviteľom tejto skupiny je balík *Net-SMNP*, ktorý obsahuje nástroje ako *snmpget*, *snmpwalk* (získanie údajov), *snmpset* (nastavenie konfigurácie), *snmptrapd* (automatizované získanie notifikácií) a ďalšie.

2.1.2 Nástroje na dohľad služieb

Častým problémom v oblasti sieti je, že určité zariadenie prestane pracovať alebo pracuje nesprávne. Aby sa zamedzilo chybám spôsobených výpadkom je dôležité, aby sa o problémovom zariadení dozvedeli administrátori sietí čo najskôr. To umožňujú nástroje na dohľad služieb.

Takýto druh softvéru zvykne bežať na viacerých strojoch súčasne a funguje na princípe pravidelnej kontroly sledovaných zariadení. V nastavených intervaloch tieto programy napríklad overujú dostupnosť smerovačov využitím základných sieťových nástrojov. Môžu tiež využiť protokol *SNMP* alebo iné testy na dostupnosti služieb.

Hlavným predstaviteľom je program *Sysmon*, ktorý je voľne šíriteľný. Ďalším zástupcom je *Nagios*, ktorý ponúka niekoľko rozšírení, ale je zložitejší.

2.1.3 Základné nástroje pre správu siete

V tejto časti zhrniem niekoľko nástrojov, ktoré si nezískali samostatnú podkapitolu. Nie je to kvôli tomu, že by boli menej používané, ale naopak sú najpoužívanejšie. Funkciou sú jednoduchšie a akonáhle sa objaví pri správe sieti problém, sú prvé, ktoré nastupujú. Takýchto nástrojov je mnoho, preto nasleduje stručný prehľad niekoľkých vybraných:

- *ping:* Ponúka základný test konektivity s iným počítačom. Informuje, či je dostupný, vypisuje štatistiky siete k vzdialenému systému.
- *traceroute:* Vypisuje zoznam smerovačov, cez ktoré sieťová premávka prechádza na ceste k vybranému cieľu.
- *telnet*, *netcat:* Umožňujú priamo komunikovať s niektorými sieťovými protokolmi.
- *ssh*, *sftp*, *scp:* Ponúkajú zabezpečený komunikačný protokol a vzdialený prístup k súborovému systému. Podporujú šifrovanie rôznymi algoritmami.

- *netstat*: Vypisuje informácie o aktívnych sieťových spojeniach.
- *ifconfig*, *ethtool*: Umožňujú konfiguráciu sieťového rozhrania.

2.1.4 Analýza sieťovej prevádzky na úrovni paketov

Niekedy je vhodné zanalyzovať sieťovú prevádzku až na úrovni *paketov*, t. j. blok prenášaných informácií počítačovou sieťou [6]. Týmto spôsobom je možné odhaliť niektoré sieťové problémy a navyše to prináša možnosť dozvedieť sa rôzne podrobnosti niektorých sieťových protokolov.

Tieto programy ponúkajú veľmi podrobné informácie o prenášaných paketoch sieťou, ale narážajú aj na niektoré obmedzenia. Typická sieťová karta zahadzuje pakety s chybným kontrolným súčtom, takže pri hľadaní takýchto chýb je nevyhnutný špecializovaný hardvér. Ďalším nedostatkom je fakt, že pri rýchlych sieťových prenosoch programy nestihnú zachytiť všetku sieťovú prevádzku.

Klasickými predstaviteľmi je konzolová aplikácia *tcpdump* a modernejší nástroj *Wireshark* (dávnejšie známy ako *Ethereal*), ktorý ponúka rozšírenú funkcionalitu. Oba umožňujú zachytávať sieťovú prevádzku, analyzovať ju (pričom *Wireshark* podporuje analýzu veľkého množstva rôznych protokolov) alebo filtrovať na základe rôznych požiadavok. Ďalší program z tejto kategórie je *ngrep*, ktorý ponúka filtrovanie analyzovanej prevádzky na základe *regulárnych výrazov*.

Hlavná funkcia týchto nástrojov je založená na operáciách: zachytávanie paketov, dekódovanie, filtrovanie na základe smerovacích údajov paketu alebo na základe vyhľadávania výrazov v dátovej časti. *Wireshark* umožňuje *reassembling* sledovanej prevádzky.

2.1.5 Systémy pre detekciu útokov

Pod názvom *Network intrusion detection system* (*NIDS*) alebo skrátene *IDS,* vystupuje skupina nástrojov, ktoré sa pokúšajú zachytiť škodlivú prevádzku ako napríklad útok zahltením siete, skenovanie portov, alebo monitorovanie siete za účelom prelomenia do cudzieho systému. Dokážu detekovať prítomnosť vírusov alebo trójskych koňov pomocou vyhľadávania špecifických *signatúr* v sieťovej prevádzke. Signatúry označujú špecifické vzory, ktoré často odhaľujú aktivitu škodlivého softvéru.

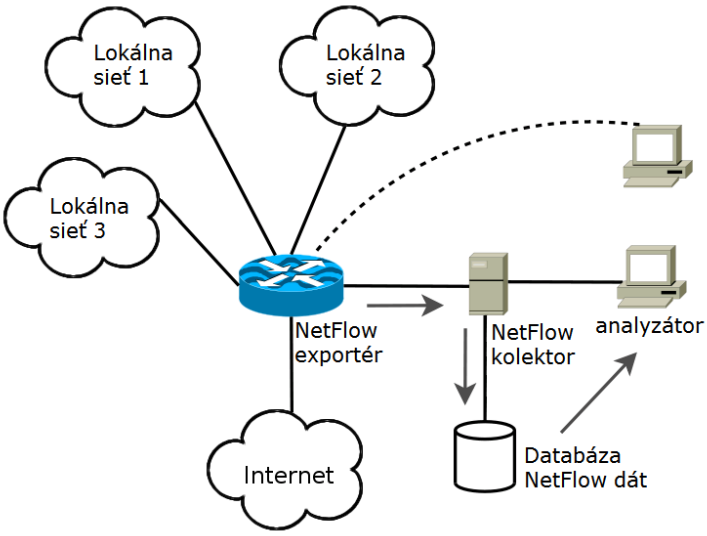
Táto funkcia pracuje na základe analyzovania obsahu jednotlivých paketov, čo kladie podobné nároky ako programy z kapitoly 2.1.4. Objavuje sa teda problém so zaťažením systému.

Voľne šíriteľným predstaviteľom je program *Snort*, ktorý obsahuje kvalitnú protokolovú analýzu a na základe vyhľadávaní signatúr umožňuje zabezpečiť systém pred rôznymi druhmi útokov. Dokáže dekódovať aplikačnú vrstvu paketov a vyhľadávať v nich vzorky. Umožňuje nastaviť aktívne zasiahnutie v prípade detekcie problému a to napríklad zablokovaním nebezpečnej časti sieťovej prevádzky. Druhou možnosťou je nastavenie reportovania ohrození.

2.1.6 Nástroje na zber informácií o podrobnostiach sieťovej prevádzky

Ďalšou skupinou sú nástroje pracujúce s architektúrou *NetFlow,* ktorá bola vyvinutá firmou *Cisco*. Umožňujú monitorovať sieťovú prevádzku z pohľadu *tokov,* ktoré sú podľa štandardu [7] definované ako sekvencie paketov s určitými spoločnými vlastnosťami, ktoré prechádzajú sieťovým zariadením.

NetFlow architektúra sa môže skladať z niekoľkých *exportérov* a jedného *kolektoru*. Exportér je napojený na monitorovanú linku a analyzuje prechádzajúce pakety, na základe ktorých vytvára štatistiky. Tie sa exportujú na kolektory, kde sa ďalej spracovávajú a ukladajú. Ďalej je ich potom možné vizualizovať, generovať grafy a takýmto spôsobom efektívne analyzovať sieťovú prevádzku.



Obr. 1: Architektúra NetFlow

Tradične sú *NetFlow* kolektory súčasťou smerovačov, čo avšak značne zaťažuje ich výkon. Preto sú nútené fungovať na základe *vzorkovania,* čo označuje prístup, kedy sa analyzuje len určitý paket v poradí.

Cisco ponúka softvérový balík *flow-tools*, ktorý obsahuje kompletné vybavenie pre prácu s *NetFlow* dátami. Existujú open-source softvérové exportéry ako *nprobe*, *fprobe*, ktoré sú však limitované výkonom a pri veľkej sieťovej prevádzke značne zaťažujú systém. Program *nfdump* predstavuje kolektor s výkonným filtrovaním. *NfSen* je grafická nadstavba, ktorá umožňuje pohodlne zobraziť vybrané *NetFlow* dáta.

2.2 Prehľad operácií sieťových nástrojov

Cieľom tejto časti by malo byť zhrnutie jednotlivých druhov operácií, ktoré môžu sieťové nástroje vykonávať. Tieto funkcie by mali byť predstavené ako logické celky a neskôr budú predmetom dôkladnej analýzy.

Keďže niektoré operácie sú založené na sofistikovaných algoritmoch, budú tu predstavené. Na základe ich vlastností je potom možné čiastočne uvážiť možnosti ich hardvérovej akcelerácie. Ak má

algoritmus veľkú pamäťovú náročnosť, nie je to ideálna vlastnosť pre HW realizáciu. Naopak, ak výpočet ponúka možnosť paralelizácie, je to veľkou výhodou.

Nasledujúcej časti opíšem operácie: zachytávanie paketov, dekódovanie paketov, filtrovanie sieťovej prevádzky, vyhľadávanie regulárnych výrazov, reassembling, overenie kontrolného súčtu a šifrovanie.

2.2.1 Zachytávanie paketov

Prvá významná operácia, ktorá je nevyhnutná súčasť mnohých nástrojov je zachytávanie paketov. Predstavuje proces, kedy sa zo sieťového rozhrania (*NIC – Network Interface Card*) kopírujú pakety do systémového priestoru.

Operácia sa označuje aj ako *pcap* (*packet capture* – zachytávanie packetov) a prostredníctvom knižníc predstavuje základ aplikácií na analyzovanie siete. Využívajú ju programy ako *tcpdump*, *tcpreplay*, *snort*, *wireshark*, *tshark*, *ngrep* a mnohé iné.

Za normálnych okolností, keď *NIC* zachytí paket, overí, či cieľová adresa vyhovuje danému zariadeniu. Ak áno, uloží ho do zásobníku a informuje systém, že potrebuje byť spracovaný. Jadro paket skontroluje, či neobsahuje chyby, odstráni určité nepotrebné vrstvy a nasmeruje ho užívateľskej aplikácii.

Ak nástroje chcú analyzovať všetky pakety, *NIC* je nutné uviesť do *promiskuitného režimu*, čo narozdiel od predchádzajúceho prípadu vylúči kontrolu, či je paket určený tomuto zariadeniu. Spracujú sa všetky prijaté pakety, určené aj iným zariadeniam, avšak tentoraz sa žiadne vrstvy neodstránia. Úplná kópia sa posiela do systémového jadra (kde je však zvyčajne zahodená, pretože sa nevyužije) a aj aplikácii. Pri vysokých rýchlostiach môže dochádzať k strate paketov, kvôli preplneniu zásobníku *NIC*, pretože systém ich nestíha odoberať a spracovávať.

Existuje niekoľko rozšírení, ktoré stavajú práve na knižnici *pcap*, kvôli využívaniu množstvom nástrojov. Jedným z rozšírení je *PF_RING* [8], ktorý prichádza ako modul unixového jadra a je rozšírením *pcap*. Aplikáciám ponúka vylepšené zachytávanie paketov vďaka kruhovému zásobníku a použitiu zdieľanej pamäte. Vylepšenie *PF_RING* je efektívnejšie aj vďaka tomu, že zamedzuje kopírovaniu (a neskôr zahodeniu) paketov do vyšších vrstiev jadra systému a tým znižuje zaťaženie operačného systému. Podobné efektívne rozšírenie, postavené nad platformou *COMBO,* nazývané *szedata2,* je predstavené v práci [9].

2.2.2 Dekódovanie paketov

Dekódovanie predstavuje proces, kedy sa z postupnosti bitov získavajú jednotlivé informácie uložené v paketoch. To sa deje vo viacerých úrovniach, už od sieťového rozhrania. Operácia sa realizuje využitím rozsiahleho vetvenia podmienok a preto je vhodná pre spracovanie v softvéri.

Z pohľadu tejto práce je zaujímavé dekódovanie aplikačnej vrstvy paketov, ktoré je podporované napríklad nástrojmi *snort* a *Wireshark*. Týmto nástrojom to umožňuje filtrovať prevádzku na základe inteligentných pravidiel. Táto operácia je kvôli veľkému druhu aplikačných protokolov náročná na systémové prostriedky, pri vysokých rýchlostiach to celkový výkon nástroja zaťažuje a nastáva strata paketov. Preto *Wireshark* ponúka možnosť zachytenia všetkých paketov a neskoršie dekódovanie.

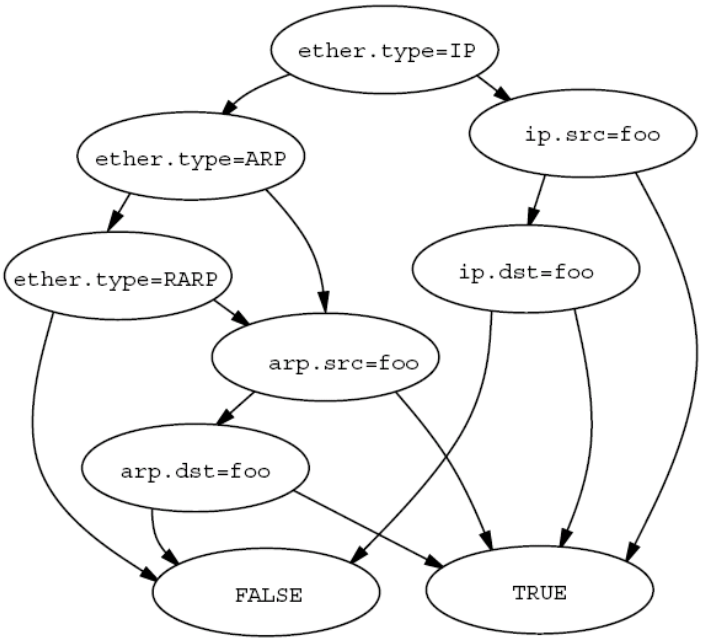
Bolo by určite užitočné, ak by tieto programy stíhali túto funkciu v behu aj pri vysokých rýchlostiach, preto bude operácia bližšie preskúmaná v ďalšej analýze.

2.2.3 Filtrovanie prevádzky

Filtrovanie je operácia, kedy pomocou špecifikovaných pravidiel vyberáme zo sieťovej prevádzky iba vybranú časť paketov. Pravidlá sa môžu týkať napríklad sieťových adries, portov, protokolov a iných údajov. Ich popis sa nachádza v manuálovej stránke *tcpdump* [10].

V súčasných unixových systémoch existuje priamo v jadre služba umožňujúca filtrovanie sieťovej prevádzky. Je to *BSD Packet Filter* (*BPF*) [11], ktorý je využitý aj v knižnici *libpcap*. Práve túto funkciu využívajú všetky spomenuté nástroje, ktoré umožňujú filtrovať zachytávané pakety.

Filtrovanie prebieha na základe zadaného výrazu, ktorý sa môže týkať protokolov, zdrojových a cieľových adries, portov atď... Filtrujúci výraz sa prekladom spracuje do vhodnej formy, s ktorým dokáže pracovať pseudo stroj pre filtrovací mechanizmus.



Obr. 2: Funkcia CFG filtra pre "host foo"

Na realizáciu tohto procesu sa využíva *CFG* (*directed acyclic control flow graph*) schéma, ktorej využitie je znázornené na Obr. 2. Tento príklad funkcie má filtrovať prevádzku hostiteľa *foo*. Za predpokladu, že uvažujeme len protokoly *IP*, *ARP*, *RARP,* tak graf pokrýva všetky možné prípady a dokáže teda rozhodnúť o každom pakete, či vyhovuje filtru alebo nie. Pritom najdlhšia cesta grafom je dlhá 5 porovnaní a priemerná má 3 porovnania. Toto riešenie je pomerne efektívne a bolo postupne optimalizované.

Keďže je však stále súčasťou jadra, ponúka sa riešenie, presunúť túto funkciu do sieťového rozhrania alebo inej sieťovej komponenty. Možným riešením je algoritmus pre klasifikáciu paketov využívajúci technológiu *FPGA,* ktorý je predstavený v práci [12].

Využíva rozhodovacie stromy, pomocou ktorých hľadá najdlhší zhodný prefix s filtrovacím pravidlom. Ich výhoda je, že ich výpočty je možné efektívne paralelizovať a každé pravidlo dekomponovať na jednotlivé časti. Hĺbka týchto stromov je konečná, takže aj klasifikácia paketu je ukončená vždy v konečnom čase. Pri vhodne postavených stromoch vzniká veľmi výkonný algoritmus, ktorý sa však stretáva s problémami, ak na vstupe dostane príliš veľké množstvo filtrovacích pravidiel. Pri konštantnej časovej náročnosti môže dochádzať až k exponenciálnej pamäťovej náročnosti. Preto je nutné hľadať optimalizácie.

2.2.4 Vyhľadávanie regulárnych výrazov

Ďalšou operáciou, ktorá spĺňa podobnú funkciu ako predchádzajúca, je filtrovanie prevádzky na základe r*egulárnych výrazov* (*RV*). Podľa definície zo špecifikácie [13] je *RV* reťazec, ktorý popisuje celú množinu reťazcov, konkrétne regulárny jazyk. Formálna definícia vraví: Nech Σ je abeceda. Regulárne výrazy nad abecedou Σ sú definované:

- ak $a \in \Sigma$, tak *a* aj *(a)* sú *RV*
- *Ø* je *RV* označujúci prázdny jazyk
- ak *a*, *b* sú regulárne výrazy, tak *a.b* označuje *RV*, kde *b* nasleduje priamo za *a*
- *a*+*b* je *RV*, ktorý označuje reťazec buď *a*, alebo *b*
- *a** je *RV* označujúci jazyk, kde za sebou nasleduje $0 \ldots \infty$ reťazcov *a*

V súčasnosti existujú dve verzie pre *RV*. Jednoduché a rozšírené, ktoré umožňujú navyše systém spätného odvolávania sa. Hlavným predstaviteľom tejto funkcie je nástroj *ngrep*, ktorý je podobný ako *tcpdump*, ale jeho výhodou je možnosť prehľadávania obsahu dátovej časti paketov pomocou regulárnych výrazov. Funkcie pre prácu s *RV* sa nachádzajú priamo v štandardných knižniciach unixových systémov (*GNU regex*) alebo je možné *ngrep* preložiť spolu s knižnicou *PCRE* [14]. Tieto knižnice predstavujú zástupcov dvoch rozdielnych prístupov. *PCRE* využíva na vyhľadávanie *RV* algoritmus *Backtracking* (spätné vyhľadávanie), ktorý rieši problémy na základe princípu pokus – umyl. Jeho nevýhoda je veľmi veľká časová náročnosť pre dlhšie regulárne výrazy, keďže čas potrebný pre vyhľadanie výrazu rastie až exponenciálne vzhľadom na dĺžku *RV*. Avšak len s týmto prístupom je možné využívať rozšírené *RV*, ktoré podporujú spätnú referenciu [13].

Druhý prístup, využívajúci aj *GNU regex,* zakladá na použití deterministického konečného automatu a označuje sa ako *DFA algoritmus*. Snaží sa automat vytvárať optimálne, aby ušetril priestor a zároveň aby nedegradoval nadmerne časovú zložitosť. Jeho nevýhoda je, že zatiaľ neumožňuje funkcionalitu rozšírených *RV*.

2.2.5 Reassembling

Výraz *reassembling* v angličtine označuje znovuzloženie a pri sieťovej prevádzke môže označovať dve významné operácie.

Jednou z nich je zloženie *fragmentovaných* paketov – t. j. paketov, ktoré boli na ceste k cieľu rozložené na menšie časti, lebo zariadenia dokážu spracovať iba pakety určitej veľkosti. Keď dorazia do cieľového systému je nutné ich opäť zložiť, čo môže spôsobiť istú záťaž na výkon.

Druhý význam predstavuje nástroj *Wireshark*, ktorý dokáže sledovať vybraný sieťový prúd. Vybraním jedného paketu, vyhľadá všetky ostatné, ktoré prislúchajú rovnakej relácii, usporiada ich a umožňuje zobraziť celú dátovú časť komunikácie. Túto funkciu neponúka za behu sieťovej analýzy, iba nad uloženými zachytenými paketmi.

Ak by sa mala táto operácia akcelerovať, vyžadovala by vysoké nároky na hardvér. Pri veľkých množstvách sieťových tokov by potrebovala veľkú kapacitu rýchlej pamäti, čo je aj finančne náročné.

Pokus o hardvérovú realizáciu predstavuje práca [15], ktorá popisuje modul na *reassembling* fragmentovaných paketov. Spracuje až 1024 rôznych tokov súčasne, ale má obmedzenia. Napríklad, že pakety musia prichádzať v správnom poradí.

2.2.6 Overenie kontrolného súčtu

Kontrolný súčet paketov slúži na overovanie ich bezchybnosti, či počas prenosu, nedošlo k nežiadanej zmene paketu. Na základe vybraného výpočtu sa z obsahu paketu vygeneruje hodnota, ktorá sa prenáša spolu s paketom. Pri spracovávaní príjemcom sa táto hodnota počíta znova a kontroluje sa s obdržanou.

Najpoužívanejší algoritmus pre výpočet kontrolných súčtov je *CRC*, ktorý bol publikovaný už v roku 1961 [16]. Na vstupe akceptuje ľubovoľne dlhú postupnosť bitov a výstupom je hodnota so zvoleným rozsahom. Najčastejšie 32 bitová hodnota, preto sa hovorí o *CRC32*.

Najväčšou výhodou tohto algoritmu je jeho výpočet, ktorý je veľmi vhodný pre hardvérovú realizáciu. Preto je v moderných sieťových rozhraniach súčasnosti realizované aj overovanie kontrolných súčtov. Táto operácia je teda už akcelerovaná v hardvéri, takže systémové prostriedky sú menej zaťažované.

2.2.7 Šifrovanie

Šifrovanie je proces, ktorý má zabezpečiť bezpečnosť citlivých údajov. Projekt *OpenSSL* má snahu vyvíjať univerzálny open-source balík šifrovacích nástrojov, ktorý by bol postavený na zabezpečených protokoloch. Súčasťou balíka je knižnica *crypto*, ktorú využívajú mnohé sieťové nástroje (napr. *ssh*, *scp*).

Vo výbave knižnice *crypto* je veľké množstvo šifrovacích algoritmov (*Blowfish*, *DES*, *AES*, *Triple DES*, *RC4*, *IDEA*), kryptografických hashovacích funkcií (*MD5*, *SHA*) a funkcií pre kryptografiu verejných kľúčov (*RSA*, *DSA*), ktoré sú odlíšené hlavne stupňom bezpečnosti.

DES – Dávnejšie považovaný za štandard, je dnes už nespoľahlivý. Jeho nástupca *Triple DES* je už bezpečný, ale náročný na výpočetnú silu. Modernejší štandard šifrovania – algoritmus *AES* je novým štandardom na základe vyhratej súťaže vyhlásenej americkým Národným ústavom pre štandardy a technológie (*National Institute of Standards and Technology*) .

2.3 Programové prostriedky vybraných nástrojov

V tejto kapitole by mali byť zhrnuté softvérové balíky a knižnice využité vybranými sieťovými nástrojmi. Tieto knižnice sú základom pre viaceré nástroje súčasne a mnohé z nich spotrebujú veľa výpočetného času, preto ich efektivita výrazne ovplyvňuje rýchlosť rôznych nástrojov.

Funkcie týchto knižníc už boli popísaná spolu s operáciami, takže nasledujúci prehľad obsahuje len zoznam možných realizácií:

- Všetky vybrané nástroje pracujú pod unixovými systémami, takže budú využívať služby tohto operačného systému a jeho štandardné knižnice (napríklad pre prácu s pamäťou alebo s ovládačmi sieťových rozhraní). Významnou súčasťou jadra z pohľadu sieťových nástrojov je *BSD Packet Filter* (*BPF*) [11], určený na filtrovanie prevádzky.
- Dôležitou súčasťou mnohých z týchto nástrojov je knižnica pre zachytávanie paketov zo sieťového rozhrania do softvéru označovaná ako *pcap*. Implementácia knižnice pre unixové systémy sa nazýva *libpcap* [17]. Existuje aj verzia pre operačné systémy Windows – *WinPcap* [18], ale tá nespadá do obsahu tejto práce.
- Jedným z kľúčových prvkov mnohých sieťových nástrojov je programové vybavenie *OpenSSL* s podporou kryptografických algoritmov vďaka knižnici *crypto*. Vytvára zabezpečenie pre webové služby, pre virtuálne privátne siete alebo aj pre softvérový balík *OpenSSH,* ktorého súčasťou sú nástroje *ssh*, *scp*.
- Pre prácu s jednoduchými regulárnymi výrazmi sa môže využiť štandardná knižnica unixových systémov – *GNU regex*, ktorá je veľmi efektívna. Menej výkonná knižnica *PCRE* umožňuje včak prácu s rozšírenými regulárnymi výrazmi.

2.4 Náročnosť nástrojov na systémové zdroje

Cieľom tejto práce je analyzovať existujúce sieťové nástroje a hľadať možnosti ich akcelerácie. Preto je vhodné hneď spočiatku uvážiť, ktoré aplikácie sú potencionálnymi kandidátmi na akceleráciu a vylúčiť tie, pri ktorých to nemá význam.

Mnohé zo spomenutých nástrojov nie sú natoľko náročné na systémové prostriedky, že by nadmerne zamestnávali systém. Príkladom sú programy zo skupiny základných nástrojov, ako *ping*, *traceroute* alebo *ssh*, ktoré využívajú na činnosť rádovo malé množstvo paketov. Rovnako sa dajú označiť aj programy pre vzdialenú správu zariadení, ale aj nástroje pre dohľad služieb (ak sa zanedbá zvolenie nevhodne krátkeho intervalu kontroly, ktorý by mohol spôsobiť zahltenie siete).

Jednoznačne zaujímavou skupinou, vhodnou na akceleráciu, sú nástroje pre analýzu sieťovej prevádzky, vrátane systémov pre detekciu útokov. Obe kategórie vyžadujú veľmi výkonné systémy, aby boli schopné odchytiť každý paket sieťovej prevádzky. Pri vysokorýchlostných sieťach tieto nástroje značne zaťažujú systém a aj preto nie sú schopné zabezpečiť garantovanú kontrolu nad prevádzkou. Je to spôsobené náročnosťou rôznych operácií, ktoré sa musia aplikovať na každý analyzovaný paket.

Značnú záťaž na systémové prostriedky predstavujú aj nástroje, ktoré zabezpečujú šifrovaný prenos veľkého množstva dát. Takýmito aplikáciami sú napríklad *sftp* alebo *scp*, pričom druhý z nich ponúka na výber viacero šifrovacích algoritmov. Šifrovanie vo všeobecnosti môže zaťažovať ďalšie nástroje, komunikačné protokoly ako napríklad protokol *HTTPS* alebo architektúru *IPsec*.

Ďalší softvér, vhodný na bližšie analyzovanie, sú softvérové exportéry *NetFlow* dát. Existencia hardvérových riešení potvrdzuje tvrdenie, že tieto nástroje zaťažujú systém a sú vhodné na určitú akceleráciu. Softvérové riešenie z tohto pohľadu nevyhovuje a využitie smerovačov s funkciou exportéru *NetFlow* tiež nie je ideálna, kvôli zaťaženiu operácie smerovania. Preto už existujú špecializované sondy, ktoré dokážu monitorovať siete aj pri vysokých rýchlostiach (1 – 10 Gbps) prevádzky. Ponúkaným riešením je sonda *FlowMon* od firmy *Invea-Tech*.

2.5 Požiadavky správcov sietí

Na základe rozhovoru s ľuďmi, aktívne využívajúcich sieťové nástroje som dospel k niekoľkým základným návrhom, požiadavkom:

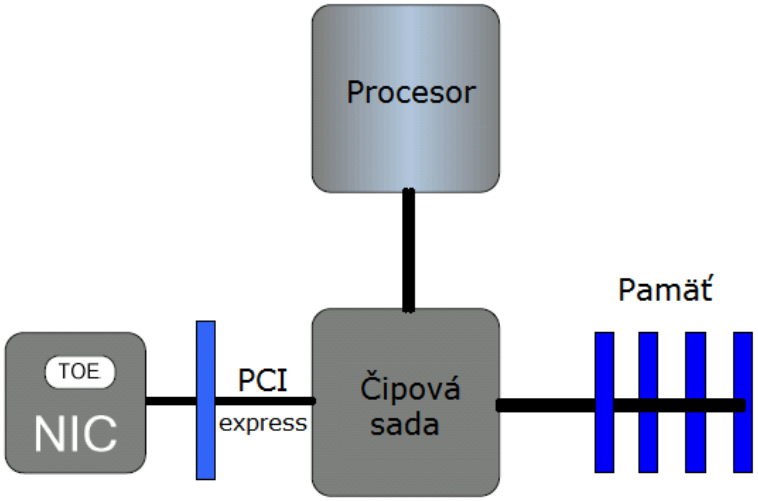
- Akcelerovať sa oplatí sieťové analyzátory ako *tcpdump* a *Wireshark*.
- Oplatilo by sa zrýchliť softvérové filtrovanie paketov.
- Akcelerované nástroje by mali ponúkať rovnaké užívateľské rozhranie, na aké sú správcovia sietí zvyknutí.

3 Platformy

Ako sa vo svete objavujú 40 Gbit a 100 Gbit rýchlostné štandardy, dizajnéri serverových platforiem budú čeliť novým možnostiam a rovnako aj extrémnym výzvam v oblasti dizajnu sieťových rozhraní. Rýchlosť spracovania paketov v takýchto sieťových zariadeniach presahujú možnosti bežných procesorov a taktiež sú kritické požiadavky na rýchlosť a odozvu vstupno-výstupných zberníc. Štandardný procesor využije na spracovanie sieťovej prevádzky o rýchlosti 1 Gbps približne 0.5 – 1 GHz zo svojej kapacity.

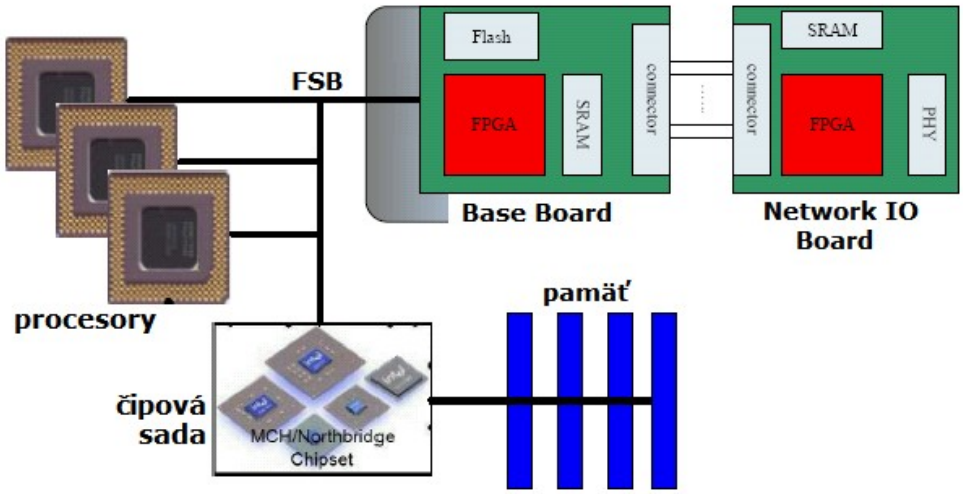
Túto úvahu a návrh riešenia prináša práca [19], ktorá predstavuje výskumnú platformu pre sieťové dizajny na multijadrových platformách. Sieťové rozhranie, realizované s technológiou FPGA, ktoré funguje na princípe koprocesoru. Výhodou tohto prístupu je rýchly priamy prístup do operačnej pamäti a krátka odozva. Umožňuje to využitie zberníc typu *PCI Express 1.0*, *2.0* a v blízkej dobe aj *PCI Express 3.0*. Nevýhodou je proprietárnosť tohto riešenia, keďže stavia na module *Intel Quick Assist base module* [20].

Procesor je pri klasických sieťových kartách zbytočne zaťažovaný operáciami, ktoré spracovávajú každý byte komunikácie (kopírovanie zásobníkov, výpočet kontrolného súčtu). Tiež je zaťažovaný prerušeniami a manažmentom zásobníku. Preto by nové modely sieťových kariet mali zahrňovať aj tieto operácie. Tento prístup sa nazýva *TCP Offload Engine – TOE (Obr. 3)* a znamená presun značnej časti úkonov pracujúcich so sieťovou prevádzkou priamo do sieťového rozhrania [19].



Obr. 3: TCP Offload Engine

Ďalšiu novinku predstavuje *40 Gbps NIC platform* [19], ktorá je postavená na serveri *Intel Xeon* 4-jadrovom systéme, ktorý využíva vysokorýchlostné spojenie *Front Side Bus* (*FSB*). Karta (*Base Board*) je zapojená priamo nahradením jedného z procesorov, takže to umožňuje komunikáciu s priepustnosťou až 68 Gbps a s veľmi malou odozvou. Komplikované aplikácie sú implementované na rozširujúcej karte (*Network IO Board*) s využitím *FPGA*. Táto architektúra je načrtnutá na Obr. 4.



Obr. 4: 40 Gbps NIC platform

3.1 NetFPGA

NetFPGA [21] je platforma, ktorá umožňuje študentom vytvoriť skutočný sieťový hardvér využitím rozšírených štandardných dizajnovacích nástrojov (napríklad *Verilog*), potom otestovať a odladiť tento hardvér priamo na sieti. *NetFPGA-v1*, ktorá sa používala na Standfordskej univerzite niekoľko rokov, bola nahradená s platformami *NetFPGA-v2* a *NetFPGA 2.1,* ktoré sa ukazujú veľmi efektívne pre študijné a výskumné účely.

Základom platformy je programovateľné hradlové pole (*FPGA – Field Programmable Gate Array*), ktoré je využité na implementáciu funkcií spracujúce dáta. Softvér bežiaci na serveri je potom zodpovedný len za funkcie ovládajúce rozhranie.

NetFPGA 2.1 [22] obsahuje veľké *Xilinx Virtex-II Pro FPGA*, ktoré je programovateľné z malého *Xilinx Spartan FPGA*, ktoré zabezpečuje *PCI* rozhranie s hostiteľským počítačom. Platforma disponuje s dvoma statickými *RAM* pamäťami a dvoma pamäťami druhej generácie *DDR2*. Komunikáciu zabezpečujú štyri 1 Gbps rozhrania. Dve *SATA* rozhrania umožňujú komunikáciu medzi viacerými *NetFPGA* v rámci jedného systému. Rozhrania *NetFPGA 2.1* sú parametrizovateľné, takže dovoľujú vyvíjať modulárny hardvér, ktorý pracuje s rôznymi dĺžkami slov. Ovládače umožňujú softvéru priamo čítať aj zapisovať do všetkých registrov v *FPGA*. Takisto pristupovať k všetkej pamäti *NetFPGA* prostredníctvom rýchlej *PCI* zbernice.

Všetky spomenuté vlastnosti povyšujú *NetFPGA* na veľmi efektívnu platformu. Bolo na nej vyvinutých už viac ako 1000 systémov s rôznymi funkciami: *NIC*, smerovač, prepínač, generátor paketov, *NetFlow* sonda a mnohé iné.

3.2 Platforma COMBO

Rodina *COMBO* [23] kariet bola vyvinutá pod záštitou firmy *CESNET* a jej partnermi. Ich hlavnou úlohou je dať vývojárom možnosť používať ich obdobne ako open-source softvér. Základom *COMBO* kariet je jedno alebo viac programovateľných hradlových polí *FPGA*, pamäte a ostatné nevyhnutné komponenty ako napájanie, vstupno-výstupné čipy, konektory atď. Vďaka flexibilite *FPGA* čipov je možné funkciu karty jednoducho meniť nahratím nového dizajnu. Týmto spôsobom je možné karty využiť na rôzne účely v oblasti výskumu a vývoja.

Existuje väčšie množstvo prevedení *COMBO* kariet. Samotná *COMBO* karta neobsahuje sieťové rozhrania, ale aj tak môže byť efektívnym koprocesorom. Nasadením rozširujúcej karty sa mení na akcelerovanú kartu s rozhraním *PCI*. Rôzne prevedenia sa líšia rýchlosťou sieťových rozhraní alebo vybavením a výkonom potrebných súčastí.

3.2.1 NetCOPE

Hlavným cieľom projektu *NetCOPE* [24] je vytvorenie konfigurovateľnej platformy pre zjednodušenie a urýchlenie vývoja sieťových aplikácií akcelerovaných pomocou *FPGA* čipov. Vlastné urýchlenie je založené na využití abstraktnej vrstvy nad *FPGA*, ktorá odtieni nízkoúrovňové hardvérovo závislé vlastnosti karty a poskytne jednoduchý prístup k jej zdrojom.

Táto vrstva umožní návrhárovi aplikácie koncentrovať sa primárne na vývoj vlastnej aplikácie a ušetriť tak značné úsilie, ktoré by bolo inak nutné venovať implementácii jednotiek pre obsluhu periférií karty. Spojenie so softvérovou časťou sieťovej aplikácie sprostredkujú pripravené ovládače a softvérové nástroje pre obsluhu karty. V hardvérovej časti platformy *NetCOPE* je integrovaný prepojovací systém, ktorý umožňuje rýchle *DMA* prenosy z karty do hostiteľského počítača.

Využitím týchto vlastností je tak platforma vhodná pre realizáciu aplikácií ako: sieťová karta s hardvérovou filtráciou, aktívny/pasívny sieťový monitoring, smerovač, generátor paketov alebo ako zariadenie pre funkcie šifrovania, zakódovania/dekódovania videa a iné.

4 Analýza sieťových nástrojov

4.1 Operácie a priepustnosť nástrojov

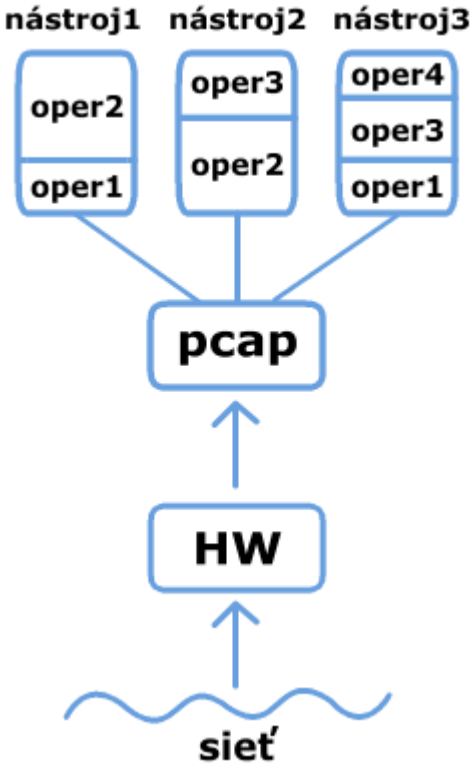
Z prehľadu kapitoly 2.2 vyplýva, že sieťové nástroje sa skladajú vždy z niekoľkých operácií, ktoré môžu byť rôzne výpočtovo náročné. V závislosti od rýchlosti sieťovej premávky a náročnosti funkcií nástrojov dochádza k zaťaženiu jadra operačného systému. To môže ďalej spôsobiť straty paketov a tým nedokonalé monitorovanie siete.

Každá operácia je závislá na vybraných vlastnostiach. Extrakcia paketov do systému závisí od zvoleného rozhrania. Rýchlosť filtrovanie prevádzky závisí na vybranom algoritme, ale môže aj na náročnosti vyhľadávaného výrazu. Šifrovanie je jednoznačne ovplyvnené výberom kryptografického algoritmu. Preto by bolo vhodné, tieto rôzne vlastnosti bližšie zanalyzovať.

Celkový výkon nástroja závisí od všetkých dielčich operácií. Často však určitá operácia svojou náročnosťou vysoko prevyšuje ostatné a tým rapídne znižuje *priepustnosť* nástroja. Priepustnosť je podľa [25] definovaná ako maximálna rýchlosť spracovania sieťovej premávky, kedy systém spracuje všetky pakety bez akýchkoľvek strát.

Aby bolo možné lokalizovať najkritickejšie operácie, je vhodná analýza nástroja ako celku. To umožní odhad zvýšenia výkonnosti v prípade ak bude daná funkcia akcelerovaná. Potom by bolo vhodné preskúmať operácie aj jednotlivo, s nastavením rôznych vlastností a pozorovať závislosť na ich priepustnosti.

Cieľom práce bude vytvoriť automatizované merania, ktoré tieto operácie analyzujú z pohľadu zaťaženia na systém a rýchlosti priepustnosti. K tomu bude potrebné rozsiahle softvérové vybavenie začínajúc od vybraných sieťových nástrojov, až po upravené a pripravené programy realizujúce operáciu samostatne, kvôli izolovanému preskúmaniu. Vyhodnotenie meraní by malo priniesť návrhy pre budúce HW akcelerácie.



Obr. 5: Architektúra nástrojov

4.2 Príprava meraní

4.2.1 Softvérové prostriedky

Na meranie popísaných operácií bude potrebné množstvo sieťových nástrojov, knižníc a ďalších aplikácií. Táto kapitola slúži ako ich prehľad a v ďalších častiach práce sa na ne bude odkazovať. Všetky nástroje sú voľne stiahnuteľné z internetu a použité verzie sú prístupné z priloženého CD, viď. zoznam príloh.

Mnohé nástroje budú preložené viackrát, s rôznymi knižnicami, kvôli porovnaniu efektivity algoritmov. Aby bolo možné určiť zaťaženie na systém jednotlivými funkciami aplikácie, základom niektorých meraní bude profilovanie nástrojov. Na to bude potrebné preložiť aplikácie aj s profilovacími údajmi a použitie profilovacích nástrojov.

Sieťové nástroje:

- tcpdump 4.0.0
- snort 2.8.3.2
- ngrep 1.45
- wireshark 1.0.7

Knižnice:

- libpcap-0.9.8, libpcap-1.0.0
- libpcap-0.9.7-ring (*PF_RING*)
- pcre-7.8
- openssl-0.9.7

Profilovacie nástroje:

- oprofile 0.9.4
- gprof 2.15.92.0.2

4.2.2 Hardvérové prostriedky

Táto kapitola by mala zhrnúť používaný hardvér, na ktorý sa budú merania ďalej odvolávať. Bude tu zoznam počítačov s technickými parametrami, na ktorých budú bežať merania. Niektoré experimenty budú bežať na prevádzke reálnej sieti, ale mnohé merania využijú hardvérové generátory paketov, ktoré tu budú tiež spomenuté.

PC, paket generátor:

- *mourverde:* Xeon(R) CPU E5410, 2.3 GHz (8 CPU), 4 GB RAM, sieťová karta Myricom 10GE, Linux 2.6.26.3
- *vouvray:* Xeon(R) CPU E5320 @ 1.86 GHz (4 CPU), 4 GB, Linux 2.6.26.3
- *veneto*: Xeon(R) CPU E5420 @ 2.50GHz, 2 GB RAM, sieťová karta Intel Pro/1000 MT
- generátor paketov *Spirent AX/4000:* 6x 1GbE port, 2x 10GbE port

4.3 Merania

4.3.1 Meranie výkonu zachytávania paketov

Zachytávanie paketov predstavuje základ viacerých sieťových nástrojov, takže rýchlosť tejto operácie je možné namerať na nich. Aby však ostatné funkcie nástroja neskreslili účinnosť tejto operácie, je vhodné vytvoriť osobitý nástroj, ktorý by pakety len zachytil a ďalej s nimi nepracoval.

Nástroj *packetplus* bol naprogramovaný podľa návodu [26] a jeho funkcia je načúvanie na vybranom porte, odchytávať pakety, počítať ich počet a potom ich zahodiť (ignorovať). Inkrementovanie počítadla je zanedbateľná operácia a výpis počtu paketov sa deje len v malom intervale, takže výsledky meraní by nemali byť nijak značne skreslené. Aby bolo možné porovnať efektivitu *libpcap* a vylepšenia *PF_RING*, nástroj je preložený dvakrát, dvoma rôznymi knižnicami a má teda verzie: *packetplus* a *packetplus-pf_ring*.

Meranie 1: Účinnosť zachytávania paketov vzhľadom k rýchlosti sieťovej prevádzky:

Prvým meraní sa zistí, ako ovplyvní rýchlosť sieťovej prevádzky zaťaženie nástroja na procesor a tiež množstvo zachytených paketov z celkového počtu odoslaných. Rýchlosť prevádzky je možné určiť ako množstvo dát prenesených za čas (*Gbps*) alebo ako počet paketov za jednotku času (*paket/s* inak *pps*). Prepočet závisí na dĺžke paketov, ktorý je v prvom meraní pevne daný. Vplyv dĺžky paketu na účinnosť nástroja bude skúšaný v druhom meraní. Využitím generátora paketov *Spirent AX/4000* sa vygeneruje prevádzka na rozhranie počítača, kde beží raz *packetplus* a druhýkrát *packetplus-pf_ring*.

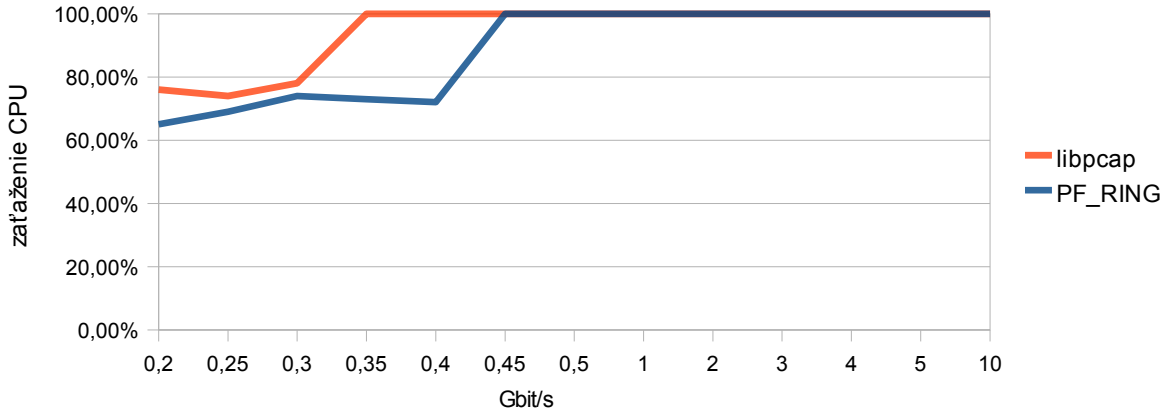
Nastavenia merania 1:

- *Spirent AX/4000:* dĺžka paketov: 64 B, rýchlosť premávky: 200 Mbps … 10 Gbps
- pre každú rýchlosť prevádzky 10 miliónov paketov
- stroj: *mourverde:* nástroje *packetplus*, *packetplus-pf_ring*
- meria sa: zaťaženie na procesor (*CPU*), percentuálne množstvo spracovaných paketov

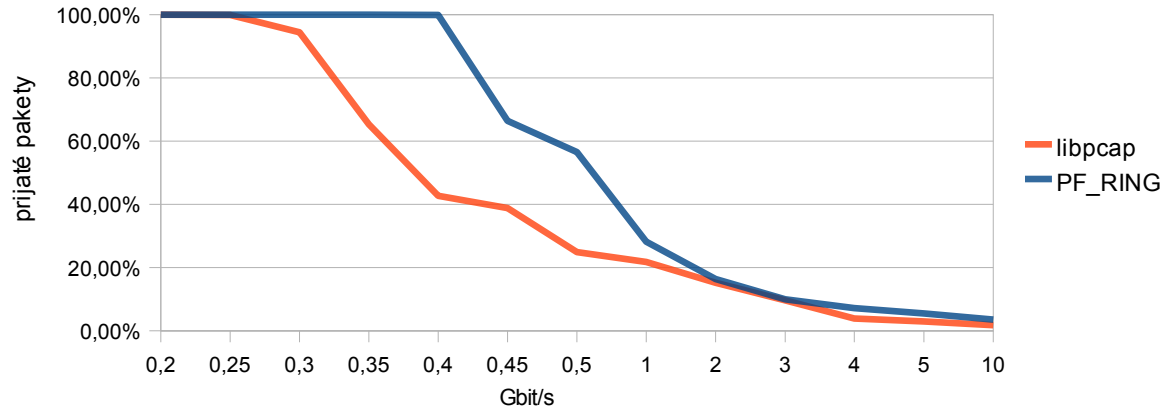
Výsledky merania 1:

Rýchlosť		libpcap		PF_RING	
Gbps	pps	CPU	prijaté	CPU	prijaté
0,20	304 611	76%	100,00%	65%	100,00%
0,25	380 764	74%	99,90%	69%	100,00%
0,30	456 917	78%	94,40%	74%	100,00%
0,35	533 070	100%	65,30%	73%	100,00%
0,40	609 223	100%	42,70%	72%	99,90%
0,45	685 375	100%	38,80%	100%	66,40%
0,50	761 528	100%	24,90%	100%	56,50%
1,00	1 523 057	100%	21,80%	100%	28,20%
2,00	3 046 113	100%	15,20%	100%	16,40%
3,00	4 569 170	100%	9,60%	100%	10,00%
4,00	6 092 227	100%	3,90%	100%	7,20%
5,00	7 615 283	100%	3,00%	100%	5,50%
10,00	15 230 566	100%	1,80%	100%	3,60%

Tab. 1: Účinnosť zachytávania paketov vzhľadom k rýchlosti premávky



Obr. 6: Zaťaženie procesora vzhľadom k rýchlosti premávky



Obr. 7: Účinnosť zachytávania paketov vzhľadom k rýchlosti premávky

Vyhodnotenie merania 1:

Ako vidno z Obr. 6, pri zvyšovaní rýchlosti prevádzky rastie záťaž nástroja na procesor. Platí to pre oba testované nástroje, s rozdielom, že použitie *PF_RING* zaťaží procesor na 100 % približne pri rýchlosti 450 Mbps a štandardný *libpcap* už pri 350 Mbps.

Zaťaženie spôsobuje, že pri týchto rýchlostiach začína rapídny pokles účinnosti spracovania paketov, čo vidno na Obr. 7. Opäť sa ukazuje, že je *PF_RING* mierne výkonnejší. Oba nástroje však pri rýchlosti 3 Gbps zvládli spracovať menej ako 10 % prevádzky.

Z nameraných hodnôt je možné vyčítať približnú priepustnosť nástrojov – t.j. najväčšiu rýchlosť, kedy sa spracujú všetky pakety. Pri dĺžke paketov 64 B sa priepustnosť pre *libpcap* ukazuje 250 Mbps (380 764 pps) a pre *PF_RING* asi 400 Mbps (609 223 pps). Dôkladnejšie sa priepustnosťou zachytávania paketov zaoberá práca [27].

Meranie 2: Účinnosť zachytávania prevádzky vzhľadom k dĺžke paketov:

Ďalší experiment má poukázať na závislosť účinnosti nástroja od dĺžky paketov, ktoré spracováva. Pri zvolenej pevnej rýchlosti paketov za sekundu sa bude opäť sledovať zaťaženie procesora a množstvo zachytených paketov. Použitým nástrojom je tentoraz len efektívnejší *packetplus-pf_ring*.

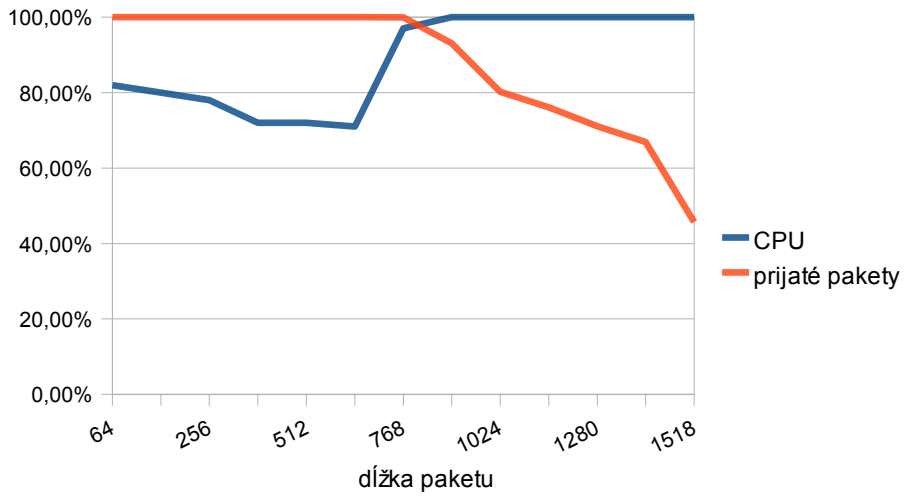
Nastavenia merania 2:

- *Spirent AX/4000:* rýchlosť premávky: 500 000 pps, dĺžky paketov: 64 – 1518 B
- pre každú dĺžku sa generuje 10 miliónov paketov
- stroj: *mourverde:* nástroj *packetplus-pf_ring*
- meria sa: zaťaženie na procesor (*CPU*), percentuálne množstvo spracovaných paketov

Výsledky merania 2:

Dĺžka Paketu [B]	CPU	Prijaté Pakety
64	82%	100,00%
128	80%	100,00%
256	78%	100,00%
384	72%	100,00%
512	72%	100,00%
640	71%	100,00%
768	97%	99,90%
896	100%	93,10%
1 024	100%	80,20%
1 152	100%	76,10%
1 280	100%	71,10%
1 408	100%	66,90%
1 518	100%	45,70%

Tab. 2: Účinnosť zachytávania paketov vzhľadom k ich dĺžke



Obr. 8: Účinnosť zachytávania paketov vzhľadom k ich dĺžke

Vyhodnotenie merania 2:

Pri paketoch s dĺžkou 64 až 256 Bytov sa zaťaženie procesora držalo okolo hodnoty 80 %. K zaujímavej zmene došlo v rozmedzí dĺžok 384 – 640 B, kedy využitie procesora kleslo rádovo na 70 %. Najväčšiu záťaž operácie zachytávania paketov spôsobujú najkratšie pakety, pretože časové intervaly medzi jednotlivými paketmi sú najkratšie. Za ten čas musí nástroj prichádzajúci paket odobrať a spracovať. Takže dlhšie pakety spôsobujú menšiu záťaž.

Ďalšia výrazná zmena nastala pri dĺžke 768 B, kedy sa CPU držalo na úrovni 100 % a nastali prvé straty paketov. Dôvodom je zrejme preplnenie zásobníka s prichádzajúcimi paketmi. Pri konštantnej rýchlosti paketov za sekundu a narastajúcej dĺžke paketov, vzrastá rýchlosť prevádzky. Veľké množstvo dát za jednotku času zaplní zásobník, preto pri prichádzajúcich paketoch nastávajú straty. Konkrétne v prípade *PF_RING*, nové pakety prepíšu staršie, ktorých spracovanie sa nemuselo stihnúť.

4.3.2 Meranie dekódovania paketov

Wireshark je veľmi výkonný program na komplexné dekódovanie paketov. Aby sa zamedzilo zaťažovaniu procesora vykresľovaním grafického rozhrania, na meranie sa použije konzolová verzia *tshark*. Vstupom meraní bude živá sieťová prevádzka, avšak uložená v binárnej forme, aby operácia zachytávanie paketov neskreslila výsledok merania. Reálna premávka je výhodná najmä kvôli tomu, že pakety majú skutočný dekódovateľný obsah, narozdiel od paketov generovaných *Spirentom*.

Nástroj *tshark* je možné spustiť s rôznymi možnosťami dekódovania paketov. Líšia sa v údajoch vypisovaných na výstup. Výstup sa nebude ukladať ani na disk, ani na obrazovku, aby sa dosiahol maximálny výkon. Z rovnakého dôvodu bude vypnutý aj preklad adries a portov.

Meranie 3: Priepustnosť operácie dekódovanie paketov

Značná časť výkonu procesoru sa spotrebuje na načítanie dát z disku. Preto je vhodné mať dáta načítané už v pamäti. To sa vyrieši jednoduchým spôsobom a to spustením testovanej aplikácie dvakrát. Druhé spustenie má už sieťovú prevádzku uloženú vopred v pamäti.

Nastavenia merania 3:

- stroj: *vouvray:* nástroj *tshark*
- vypnutý preklad adries aj všetky výpisy
- zdroj dát: zaznamenaná reálna sieťová prevádzka, predčítaná v pamäti
- meria sa: zaťaženie na procesor (*CPU*), priepustnosť operácie

Výsledky merania 3:

Meranie	Čas dekódovania [s]	CPU [%]
1.	1,367	100
2.	1,426	100
3.	1,550	100
4.	1,403	100
5.	1,632	100

Tab. 3: Čas dekódovania sieťovej prevádzky

Vyhodnotenie merania 3:

Priemerná doba potrebná na dekódovanie sieťovej prevádzky o veľkosti 764 MB vychádza 1,476 s. Z toho vychádza priepustnosť tejto operácie približne 4 Gbps. Dekódovanie je však relatívny pojem a závisí od množstva a druhu údajov, ktoré je nutné z prevádzky dostať.

Výsledná rýchlosť operácie je veľmi ovplyvnená spôsobom interpretovania výsledných hodnôt. Za samotné dekódovanie sa dá považovať operácia, kedy sa zistí adresa potrebnej položky. Táto operácia pozostáva z jednoduchých úkonov (pre procesor) – vetvenie podľa analyzovaných protokolov a výpočet *offsetu,* t.j. posunutia, na ktorej adrese sa hodnota nachádza.

Interpretovať sa zistené hodnoty dajú rôznym spôsobom a to je často najväčšia záťaž na celkový výkon nástrojov. Aplikácia *tshark* podporuje rôzne druhy výpisov: *pdml* (*XML* formát s

detailnými informáciami), *psml* (*XML* formát so zhrňujúcimi informáciami) alebo ako jednoducho čitateľný text. Ďalšie meranie ukazuje rozdiely zaťaženia pri rôznych formátoch dekódovania.

Meranie 4: Vplyv rôznych formátov dekódovania na výkon nástroja

Tento experiment bude volať *tshark* s rôznymi parametrami pre výpis. Výstup sa však nebude nikam ukladať, ani zobrazovať, ale hneď po pripravení zahadzovať. To zamedzí náročnej operácii ako ukladanie, skresliť výsledky. Program *tshark* bude preložený s profilovacími údajmi a analýza prebehne využitím *gprof*. Odskúčané budú 4 formy dekódovania: stručný, podrobný textový formát a spomínané *pdml*, *psml*.

Výsledky meranie 4:

% time	cumulative seconds	self seconds	calls	self us/call	total us/call	name
90.62	0.29	0.29	100000	2.90	3.10	process_packet
6.25	0.31	0.02	1200000	0.02	0.02	get_line_buf
3.12	0.32	0.01				main
0.00	0.32	0.00	1800077	0.00	0.00	data_start
0.00	0.32	0.00	100000	0.00	0.00	print_line
0.00	0.32	0.00	100000	0.00	0.00	print_line_text

Obr. 9: Výstup profilovania tshark pre stručný textový formát

Na výstupe profilovania *tshark* (Obr. 9) vidno, že pre tento formát nástroj využil 90 % času na spracovanie paketu, asi 6 % času bolo treba na alokovanie pomocnej pamäti na ukladanie výstupného textu, 0 % času na výpis (vďaka zahadzovaniu výstupu). 3 % času nástroj minul na inicializovanie operácií.

textový formát – stručný		psml	
process_packet	90,62%	print_escaped_xml	52,38%
get_line_buf	6,25%	process_packet	28,57%
main	3,12%	main	14,29%
		proto_tree_write_psml	4,76%
textový formát – podrobný		**pdml**	
proto_tree_print_node	43,10%	print_escaped_xml	46,51%
print_hex_data_buffer	24,27%	proto_tree_write_node_pdml	43,02%
print_line_text	21,34%	write_pdml_field_hex_value	8,92%
print_line	7,53%	get_field_data	0,97%
process_packet	1,67%	process_packet	0,22%
get_field_data	0,84%	proto_tree_write_pdml	0,22%
main	0,84%	main	0,15%
proto_tree_print	0,42%		

Tab. 4: Rozloženie záťaže nástroja pre rôzny formáty dekódovania

Ako vidno na Tab. 4, pri ďalších formátoch dekódovania bolo potrebné viac výpočetného času na prípravu všetkých potrebných dát na výstup. Aj napriek tomu, že sa generované dekódované údaje zahadzovali, pri podrobných výpisoch nástroj strávil takmer 99 % času s formátovaním dát.

Meranie prebehlo na zaznamenanej reálnej prevádzke (načítanej v pamäti) o množstve 100 tisíc paketov. Celkový čas potrebný na prípravu výstupu dekódovaných údajov pri stručných formátoch bol rádovo 2 sekundy, pri podrobnom textovom formáte 25 sekúnd a pri *pdml* dokonca až 77 sekúnd.

4.3.3 Meranie operácie filtrovania

Výkon filtrovanie sieťovej premávky sa bude merať na základnom nástroji *tcpdump* preloženom s profilovacími údajmi, aby bolo možné zistiť zaťaženie jednotlivých častí programu. Keďže funkcia filtrovania prebieha na úrovni jadra operačného systému, je nutné profilovať aj jadro. Preto sa použije profilovací nástroj *OProfile* [28].

Sieťová prevádzka bude generovaná nástrojom *Spirent*. Filter bude volený tak, aby ním prešli všetky vygenerované pakety. Najdlhšia cesta filtrom je totiž práve tá, kedy nastáva zhoda pravidiel.

Meranie 5: Výkon filtrovania pri rôznych plavidlách filtra

Tento experiment by mal odskúšať, ako vplýva nastavenia rôznych pravidiel filtra, na výkon nástroja. Mal by poukázať na zaťaženie systému a rýchlosť spracovania paketov. Aby sa zamedzilo skresleniu výsledkov, program *tcpdump* bude zahadzovať zaznamenanú prevádzku, nebude prekladať adresy, porty a ani vypočítavať časové značky. Merania sa budú robiť na rôznych rýchlostiach prevádzky a počet požadovaných vyfiltrovaných paketov sa bude prispôsobovať tejto rýchlosti.

Nastavenia merania 5:

- stroj: *mourverde*
- zdroj prevádzky: *Spirent AX/4000:* pakety dĺžky 64 – 1518 B, rýchlosť: 1 – 5 Gbps
- nástroj *tcpdump* s *libpcap-1.0.0:* vypnutý preklad adries, portov aj všetkých výpisov
- merania: bez filtra, jednoduchý filter (1 porovnanie), rozsiahly filter (vyše 20 porovnaní)
- meria sa: zaťaženie na procesor (*CPU*), účinnosť nástroja, čas
- automatizovaný skript *test-filter.sh,* ktorý okrem merania generuje aj profilovacie údaje

Výsledky merania 5:

Rýchlosť [Gbps]	Počet paketov	Filter 0			Filter 1			Filter 2		
		CPU	Čas [s]	Účinnosť	CPU	Čas [s]	Účinnosť	CPU	Čas [s]	Účinnosť
1,00	5 mil.	16,9%	32,475	99,99%	18,6%	32,472	99,99%	18,9%	32,474	99,99%
2,00	10 mil.	44,5%	32,497	99,99%	43,3%	32,499	99,99%	40,3%	32,478	99,99%
3,00	15 mil.	54,2%	32,509	99,97%	55,5%	32,477	99,99%	59,6%	32,504	99,95%
4,00	20 mil.	98,2%	46,363	69,97%	97,8%	49,286	65,83%	91,7%	102,585	31,58%

Tab. 5: Výkon filtrovania pri rôznych pravidlách filtra

Vyhodnotenie merania 5:

Ak sa porovná zaťaženie nástroja na procesor pri rôznych filtroch, nie je badať žiadne výraznejšie zmeny pri rýchlostiach do 3Gbps. Meranie bez filtra a s rozsiahlym filtrom majú rádovo podobné zaťaženie na systém. Odchýlky, ktoré vznikajú sú spôsobené aj vzorkovaním merania zaťaženia procesora.

Pri vysokých rýchlostiach (už pri 4 Gbps) dochádza k nadmernému zaťaženiu procesora, vplyvom veľkého množstva prichádzajúcich paketov. Zaťaženie samotným nástrojom je síce menšie, ale veľkú réžiu spôsobuje obsluhovanie množstva prerušení. Pri rýchlosti 5 Gbps už boli merania veľmi skreslené, pretože zachytávanie paketov spôsobilo nadmernú záťaž.

Pri týchto veľkých rýchlostiach sa však už prejavuje aj zaťaženie filtrom. Pri rozsiahlych pravidlách filtrovania vznikajú omnoho väčšie straty (68,42 %) ako pri meraní bez filtra (30,03 %). Aby bolo možné bližšie zistiť, ako filter zaťažuje procesor, je nutné analyzovať podrobne beh nástroja pomocou profilovania.

Meranie 6: Profilovanie systémového jadra a *tcpdump* s rôznymi filtrami

%	image name	app name	symbol name
15.6964	myri10ge	myri10ge	/myri10ge
11.4457	oprofiled	oprofiled	odb_update_node
5.0275	vmlinux	vmlinux	fn_hash_lookup
4.5323	vmlinux	vmlinux	ata_output_data
3.5454	vmlinux	vmlinux	tpacket_rcv
3.2910	vmlinux	vmlinux	netif_receive_skb
2.8008	vmlinux	vmlinux	ip_route_input_slow
2.4177	tcpdump-pg	tcpdump-pg	_IO_new_file_xsputn
1.6802	vmlinux	vmlinux	sock_def_readable
1.6344	vmlinux	vmlinux	skb_copy_bits
1.5681	tcpdump-pg	tcpdump-pg	mempcpy
1.4132	tcpdump-pg	tcpdump-pg	fwrite
1.1910	vmlinux	vmlinux	ip_rcv
1.1176	vmlinux	vmlinux	native_read_tsc
1.1040	vmlinux	vmlinux	__alloc_skb
0.4329	vmlinux	vmlinux	sk_run_filter

Obr. 10: Profilovanie systémového jadra a tcpdump – bez filtra

%	image name	app name	symbol name
14.5995	myri10ge	myri10ge	/myri10ge
12.3580	oprofiled	oprofiled	odb_update_node
6.6334	vmlinux	vmlinux	sk_run_filter
4.7551	vmlinux	vmlinux	fn_hash_lookup
3.2204	vmlinux	vmlinux	ata_output_data
3.1645	vmlinux	vmlinux	tpacket_rcv
3.0588	vmlinux	vmlinux	netif_receive_skb
2.6098	vmlinux	vmlinux	ip_route_input_slow
1.6393	vmlinux	vmlinux	sock_def_readable
1.4907	vmlinux	vmlinux	skb_copy_bits
1.3719	tcpdump-pg	tcpdump-pg	_IO_new_file_xsputn
1.3520	tcpdump-pg	tcpdump-pg	mempcpy
1.1422	vmlinux	vmlinux	ip_rcv
1.0805	vmlinux	vmlinux	__alloc_skb

Obr. 11: Profilovanie systémového jadra a tcpdump – s rozsiahlym filtrom

Vyhodnotenie merania 6:

Vo výsledkoch, značné percentá zaťaženia zabrali samotné funkcie profilovacieho nástroja. V oboch zobrazených prípadoch je to približne 12 %. Ak odpočítame čas strávený v týchto funkciách, na funkciu *sk_run_filter* vychádza v prípade bez filtra 0.5 % času a s rozsiahlym filtrom 7.5 % času. Zvyšok času bol strávený vo funkciách obsluhujúcich zachytávanie paketu.

Z merania vyplýva, že filtrovanie *BPF* filtrom je vďaka optimalizácii výkonné a v nástrojoch nespôsobuje nadmernú záťaž. Aj napriek použitiu výkonnej knižnice *libpcap-1.0.0*, rádovo väčšiu záťaž spôsobila operácia zachytávanie paketov.

%	image name	app name	symbol name
25.0708	tcpdump-pg	tcpdump-pg	memcpy
21.4999	vmlinux	tcpdump-pg	div_s64_rem
4.9276	tcpdump-pg	tcpdump-pg	bpf_filter
3.8122	tcpdump-pg	tcpdump-pg	mempcpy
3.3877	vmlinux	tcpdump-pg	irq_entries_start
2.9799	tcpdump-pg	tcpdump-pg	pcap_offline_read
2.8467	tcpdump-pg	tcpdump-pg	fread
2.6969	vmlinux	tcpdump-pg	do_generic_file_read
2.5637	tcpdump-pg	tcpdump-pg	read

Obr. 12: Profilovanie systémového jadra a tcpdump – offline filter

Profilovanie, ktorého výsledok je na Obr. 12 zobrazuje pokus, kedy sa pakety filtrovali zo zaznamenanej sieťovej prevádzky predčítanej v pamäti. V tomto meraní funkcia *bpf_filter* zabrala 5 % času. Na každých 764 MB uloženej sieťovej prevádzky nástroj spotreboval čas rádovo 1 s (27 s bez predčítania). Hrubým odhadom by mohla mať samostatná operácia filtrovania priepustnosť 15 Gbps, pri maximálnom zaťažení procesora.

4.3.4 Meranie operácií regulárnych výrazov

Pri práci s regulárnymi výrazmi existujú dva rôzne prístupy (kapitola 2.2.4), ktoré sa líšia v možnostiach vyhľadávania. V určitých prípadoch sa ale výrazne líšia aj vo výkone. Klasickým zástupcom pracujúci s rozšírenými regulárnymi výrazmi je knižnica *PCRE*. Umožňuje funkcionalitu rozšírených *RV,* využíva algoritmus *backtracking*. Pôvodne mala byť využitá na meranie tejto operácie spolu s nástrojom *ngrep*. Avšak ten predvolene pracuje s knižnicou *GNU regex,* ktorá je pre prácu so základnými *RV* veľmi výkonná. V prípade potreby, kedy je na vstupe rozšírený *RV* (obsahujúci napríklad spätnú referenciu), dokáže prepnúť do režimu, kedy pracuje aj s nimi.

Meranie 7: Vplyv rôznych regulárnych výrazov na výkon nástroja *ngrep*

Prvý experiment by mal odmerať, ako vplýva druh a dĺžka regulárneho výrazu na rýchlosť spracovávania sieťovej prevádzky nástrojom *ngrep*. Meranie bude prebiehať na zaznamenanej sieťovej prevádzke, predčítanej v pamäti, aby výsledky neboli nadmerne skreslené ostatnými funkciami.

Nástroj *ngrep* bude upravený tak, aby na výstup nevypisoval žiadne údaje. Bude preložený s profilovacími údajmi, aby bolo možné odsledovať zaťaženie jednotlivými funkciami.

Nastavenia terania 7:

- stroj: *vouvray:* nástroj *ngrep* (z programu odstránené všetky výpisy)
- zdroj dát: zaznamenaná reálna sieťová prevádzka, predčítaná v pamäti
- vyhľadávanie rôznych regulárnych výrazov
- meria sa: zaťaženie na procesor (*CPU*), čas potrebný na vyfiltrovanie

Výsledky merania 7:

Regulárny výraz filtra	Čas	CPU
(bez výrazu)	2,897 s	100 %
`x`	4,180 s	100 %
`xyz`	6,404 s	100 %
`val[0-9]{1,3}`	6,475 s	100 %
`<a>[^<]*</a>`	6,468 s	100 %
`([A-Z]{2}).*\1`	19,912 s	100 %
`([A-Z]{2})aa|bb\1`	31,732 s	100 %
`([A-Z]{2})aa|bb|cc\1`	41,872 s	100 %
`.*(xx).*\1`	> 2 hod	100 %

Tab. 6: Závislosť času spracovania na regulárnom výraze filtra

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 48.46     23.89    23.89                             re_match_2
 28.57     37.97    14.08                             re_search_2
  5.23     40.55     2.58                             group_match_null_string_p
  2.90     41.98     1.43                             vfprintf
  2.88     43.40     1.42                             op_match_null_string_p
  2.58     44.66     1.27                             read
  1.30     47.23     0.64                             memcpy
  0.30     47.73     0.15  1357013   110.54   132.64  process
```

Obr. 13: Zaťaženie upraveného nástroja ngrep pri filtrovaní regulárnym výrazom

Vyhodnotenie merania 7:

Sieťová premávka o veľkosti 764 MB predčítaná v pamäti zabrala programu *ngrep* veľmi variabilný čas, závislý od nastaveného filtrovacieho regulárneho výrazu. Systémové jadro bolo zaťažené na 100 % a z toho podľa výsledkov profilácie (Obr. 13), 95 % zabrali funkcie regulárnych výrazov.

Pri najjednoduchších filtroch trvalo spracovanie rádovo 6 sekúnd, čo znamená priepustnosť niečo viac ako 100 Mbps. Avšak použitím funkcií rozšírených regulárnych výrazov sa priepustnosť rapídne znižuje. Preto je rozhodujúcim faktorom algoritmus, ktorý vykonáva funkcie *RV*. Ďalšie meranie by mal porovnať dva základné prístupy.

Meranie 8: Porovnanie *egrep* a *pcregrep*

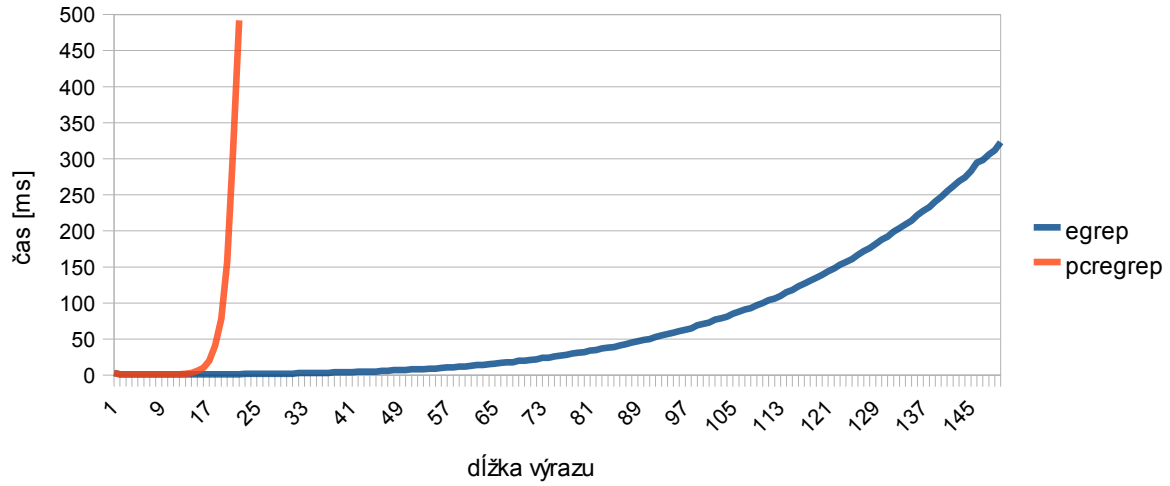
Toto meranie by malo odskúšať najextrémnejšie prípady vyhľadávania zhody *RV*. Takým prípadom je výraz, kedy algoritmus musí otestovať všetky možné pozície umiestnenia a až posledný označí zhodu. Porovná programy využívajúce dva rozdielne algoritmy:

- *pcregrep* – využíva *backtracking*
- *egrep* – využíva princíp konečných automatov

Nastavenia merania 8:

- stroj: *mourverde,* nástroje *pcregrep*, *egrep*
- automatizovaný skript *test-re.sh* generuje vyhľadávanie regulárneho výrazu `'(x?){i}x{i}'` vo výraze `'x{i}'` postupne pre dĺžky $i = 1 \ldots 150$ a predáva ich spomínaným nástrojom
- meria sa: čas potrebný na vyhľadanie zhody

Výsledky merania 8:



Obr. 14: Závislosť času spracovania od dĺžky regulárneho výrazu

Vyhodnotenie merania 8:

Na nameraných hodnotách (Obr 14.) vidno, že pri nástroji *pcregrep* s dĺžkou výrazu rastie čas potrebný na vyhľadanie regulárneho výrazu exponenciálne, kým pri *egrep* kvadraticky. Program *pcregrep* by teda pri dĺžke 50 potreboval na nájdenie zhody 5 rokov. Preto má naprogramované zabezpečenie, ktoré nástroj ukončí neúspechom v prípade, že presiahne určitú dobu výpočtov. Aj napriek neefektivite, *backtracking* je zatiaľ jediný spôsob, ako sa dá realizovať funkcia spätnej referencie.

4.3.5 Meranie operácie šifrovania

Meranie prebehne využitím nástroja *scp,* ktorý slúži na zabezpečené kopírovanie súborov zo vzdialeného stroja. Pôvodne mal experiment prebehnúť tak, že sa *scp* pripojí na lokálny stroj a bude kopírovať odtiaľ, aby sa zamedzilo réžii spojenej s prácou sieťovej karty. Avšak sa ukázalo, že týmto spôsobom je systém zaťažovaný ešte viac, pretože musí na jednom stroji prebiehať zároveň šifrovanie aj dešifrovanie.

Tento pokus aspoň poukázal na zaťaženie oboma týmito funkciami a ukazuje sa, že spôsobujú rádovo podobné zaťaženie, s rozdielom asi 1 – 2 % (dešifrovanie náročnejšie).

Meranie 8: Zaťaženie operácie šifrovania využitím rôznych šifrovacích algoritmov

Meranie by malo odmerať zaťaženie týchto operácií na systém a zistiť rozdiely medzi rôznymi šifrovacími algoritmami. Ďalej by mal vypočítať priepustnosť operácie šifrovania pri použití rôznych algoritmov. Základom bude šifrovaný prenos dát medzi 2 počítačmi cez sieťový spoj 1 Gbps.

Nastavenia merania 9:

- stroj: *mourverde,* stroj odkiaľ prebehne kopírovanie dát: *veneto*
- nástroj *scp,* prenášané dáta: 700 MB binárnych dát
- automatizovaný skript *test-ssl.sh* sa napojí na vzdialený stroj, využitím nástroja *scp* a rôznych šifrovacích algoritmov (*3des, aes128, aes256, arcfour, blowfish, cast128*) kopíruje súbor na lokálny stroj, avšak ho neukladá, aby nedochádzalo spomaleniu vplyvom zápisu na disk
- meranie vykonáva profiláciu nástroja a jadra a meria rýchlosť prenosu dát

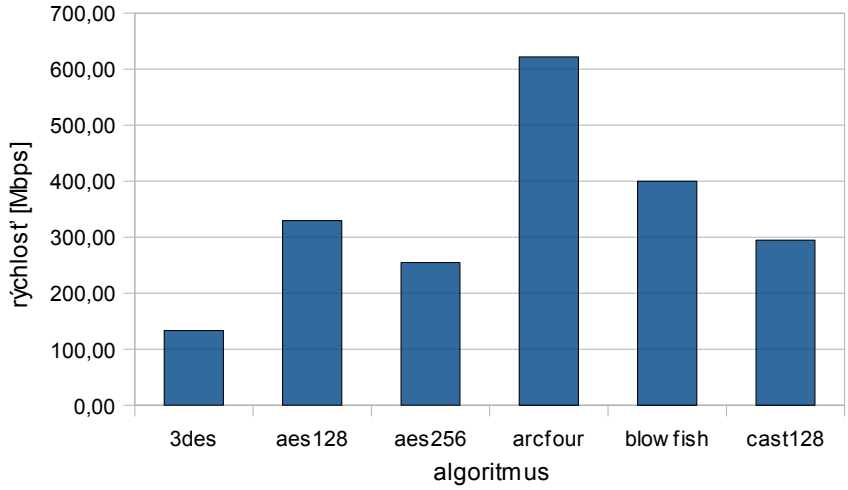
Výsledky merania 9:

Algoritmus	Rýchlosť [Mbps]	Zaťaženie
3des	133,33	90,03%
aes128	329,41	80,83%
aes256	254,55	83,26%
arcfour	622,22	56,49%
blowfish	400,00	73,72%
cast128	294,74	81,36%

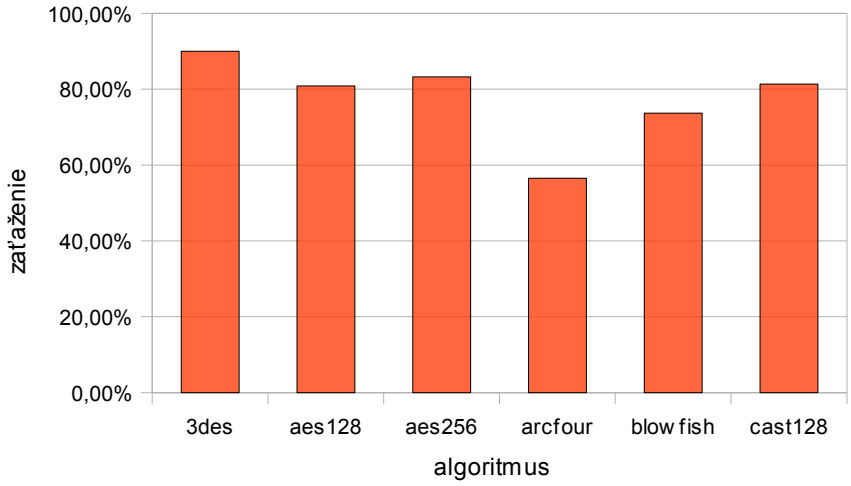
Tab. 7: Rýchlosť prenosu a miera zaťaženia šifrovaním při rôznych algoritmoch

```
samples  %        app name                 symbol name
888283   90.0383  libcrypto.so.0.9.7a      /lib/libcrypto.so.0.9.7a
15828     1.6044  vmlinux                  ata_output_data
8026      0.8135  libc-2.3.4.so            memcpy
6878      0.6972  vmlinux                  __copy_user_intel
```

Obr. 15: Výstup profilovania nástroja scp pri algoritme 3DES



Obr. 16: Závislosť rýchlosti prenosu na šifrovacom algoritme



Obr. 17: Miera zaťaženia jadra v závislosti od šif. algoritmu

Vyhodnotenie merania 9:

Na Tab. 7 je znázornená priemerná dosiahnutá rýchlosť prenosu pri rôznych šifrovacích algorimoch. Najpomalší sa ukázal algoritmus *3des* so 133 Mbps a najrýchlejší *arcfour* (*RC4*) s 622 Mbps.

Tento výsledok potvrdzuje aj miera zaťaženia operácie šifrovania na jadro systému. Najväčšiu záťaž dosiahol algoritmus *3des* – až 90 %. Práve preto, že algoritmus je náročný na výpočet, je aj celková rýchlosť prenosu nižšia. Ako naznačuje Obr. 15, zvyšných 10 % zaberá réžia zachytávania paketov.

Pri všetkých zvolených šifrovaných algoritmoch bolo zaťaženie systému vplyvom nástroja *scp* 100 %. Na Obr. 17 vidno, že vo všetkých prípadoch spôsobovala najväčšiu záťaž knižnica pre prácu so šifrovaním a to v rozmedzí od 56 % do 90 %.

5 Návrh HW akcelerácie

Cieľom tejto kapitoly je vyhodnotenie predchádzajúcich meraní a na základe nich identifikovať operácie vhodné na hardvérovú akceleráciu. Mala by načrtnúť možnosti, ako akceleráciu realizovať a tiež odhadnúť možné zrýchlenie výsledných sieťových nástrojov.

Zachytávanie paketov

Táto operácia je na počiatku každého sieťového nástroja. Miera zaťaženia touto operáciou je pri rôznych nástrojoch rozdielna a závisí od rýchlosti spracovávanej premávky (Meranie 1). Ako ukazuje Meranie 2, dĺžka paketov (teda aj rýchlosť paket/s) veľmi výrazne vplýva na efektivitu tejto operácie. Pri analyzátoroch sieťovej prevádzky je kritická a vyžaduje teda istý spôsob akcelerácie. Túto funkciu vykonáva vždy určitá knižnica (v *Unixe* najčastejšie *libpcap*), takže je to spoločný prvok pre množstvo nástrojov. Práve preto je to vhodné miesto na transparentnú akceleráciu. To využíva vylepšenie *PF_RING* alebo aj rozhranie *szedata2* pre *NetCOPE,* ktoré zvláda plný 1 Gbps sieťovej prevádzky bez nadmerného zaťaženia systému [27]. Bez dostatočnej rýchlosti tejto operácie, nie je možné akcelerovať ostatné funkcie, postavené nad ňou.

Dekódovanie paketov

Rýchlosť priepustnosti dekódovania paketov vyšlo v Meraní 3 rádovo 4 Gbps vtedy, ak sa všetok výstup okamžite zahadzoval. Meranie 4 však ukázalo, že táto operácia je veľmi závislá od toho, čo očakávame na výstupe. Dekódovanie je pojem, ktorý môže označovať funkciu procesora, ktorý na základe vetvenia podľa načítaných hodnôt, pristupuje na rôzne adresy kde sa nachádzajú údaje. Táto operácia je pre procesor počítača veľmi elementárna a *nevyžaduje* akceleráciu. Omnoho väčšiu záťaž na systém vyvíja vyčítanie údajov a formátovanie do vhodného výstupného formátu.

Filtrovanie

Filtrovanie sieťovej prevádzky je vo väčšine nástrojov súčasnosti realizované *BPF* filtrom v jadre unixových systémov. Ako ukazuje Meranie 6, zaťaženie touto funkciou je pri najrozsiahlejších pravidlách filtrovania asi 7,5 % popri ostatných funkciách (najmä zachytávanie paketov). Ukazuje sa, že evolúcia *BPF* filtra priniesla efektívny algoritmus, ktorý zvláda nároky na rýchlosť bez výrazného zaťaženia na systém. Avšak pri ďalších rastoch rýchlosti prevádzky by zaťaženie na systém rástlo. Preto je *vhodné* filtrovanie akcelerovať a presunúť všetku jeho funkcionalitu do zvlášť zariadenia, ktoré by filtráciu vykonávalo transparentne a bez zaťaženia na systém. To je do určitej miery možné použitím špeciálnych algoritmov [12] (popísaných v kapitole 2.2.3) a použitím *FPGA*. Je obtiažne odhadnúť výsledné akcelerovanie sieťového nástoja, pretože závisí od viacerých kritérií. Filtrovanie by prebiehalo transparentne, ešte pred tým, ako pakety dorazia k systému, takže nežiadané

pakety vôbec nezaťažujú systém ani operáciou zachytávania paketov. Vhodne zvolenými pravidlami filtrovania by systém zachytil len potrebné pakety, bez strát kvôli operácii zachytávania.

Regulárne výrazy

Ako ukazuje Meranie 7, zaťaženie filtrovaním regulárnymi výrazmi závisí od zvoleného výrazu a už pri jednoduchých regulárnych výrazoch je miera zaťaženia systému týmito funkciami až 95 %. To ukazuje na silnú *potrebu* akcelerovania tejto funkcie. Meranie 8 porovnáva výkon dvoch rozdielnych algoritmov na prácu s *RV*. Ukazuje sa, že prístup s konečnými automatmi je omnoho rýchlejší. Jeho nevýhoda je však, že nedokáže pracovať s rozšírenými regulárnymi výrazmi (so spätnou referenciou). Algoritmus *backtracking* je ale pre akceleráciu neprípustný, kvôli exponenciálnej časovej zložitosti od dĺžky výrazu. Práca [29] sa zaoberá akcelerovaním funkcií regulárnych výrazov založeným na konečných automatoch a snaží sa riešiť ďalší problém – veľkú pamäťovú náročnosť tohto prístupu. Vďaka rôznym optimalizáciám je možné zmenšiť pamäťové nároky, avšak za cenu spomalenia výkonu.

Pomocou *Amdahlovho zákona* [30] je možné vypočítať maximálny teoretický nárast výkonnosti pôvodnej sekvenčnej aplikácie po jej paralelizácii. Označuje ho vzťah:

$$N_V = \frac{T_S + T_P}{T_S + \frac{T_P}{n}}$$

- N_V je koeficient maximálneho nárastu výkonnosti aplikácie
- T_S je čas spracovania sekvenčných algoritmov
- T_P je čas spracovania paralelných algoritmov
- n je počet jadier procesora

Ak sa bude počet jadier zvyšovať, hodnota výrazu $\frac{T_P}{n}$ sa bude zmenšovať a pre $n \to 0$ bude limitne rovný nule. Ak to zoberieme do úvahy a $T = T_S + T_P$ označíme ako najlepší exekučný čas sekvenčnej aplikácie, dostaneme upravený *Amdahlov zákon:*

$$N_V = \frac{T}{T_S}$$

Keď tento vzťah chceme použiť na výpočet teoretickej maximálnej možnej akcelerácie sieťových nástrojov, tak akcelerovanú operáciu budeme považovať za maximálne zrýchlenú. Zvyšné operácie budú sekvenčné algoritmy. Takže ak prenesieme výraz na výpočet maximálneho možného zrýchlenia nástroja *ngrep*, ktorý na prácu s regulárnymi výrazmi využíva 95 % času, dostaneme:

$$N_V = \frac{100\,\% T}{5\,\% T} = 20$$

Ak by sa funkcie spojené s vyhľadávaním regulárnych výrazov hardvérovo akcelerovali na rýchlosť linky, tak by tým nástroj *ngrep* mohol byť zrýchlený až **20-násobne**.

Šifrovanie:

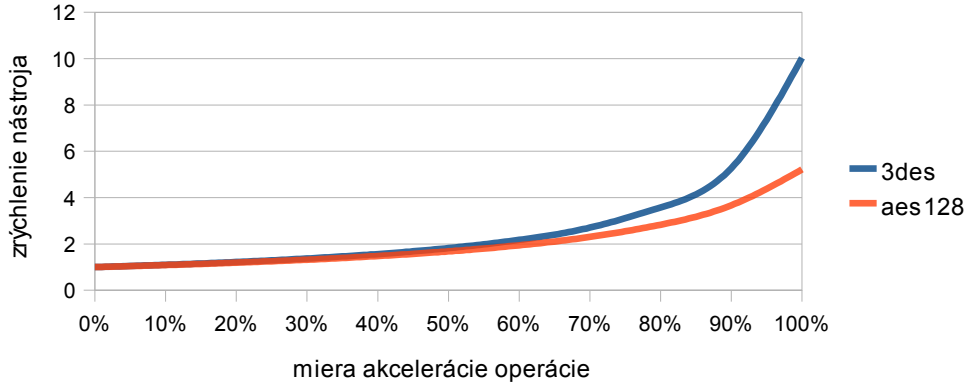
Výsledok merania 9 ukázal, že operácie šifrovania výrazne zaťažujú systém a znižujú priepustnosť nástrojov. Pri všetkých šifrovacích algoritmoch bolo využitie systémových prostriedkov maximálne a nameralo sa, že jednotlivé algoritmy predstavovali rôznu mieru zaťažovania v rozmedzí 56 – 90 %. Ukazuje sa, že akceleráciou týchto operácií, by sa výrazne zvýšil výkon nástrojov využívajúcich šifrovanie. Keď pomocou *Amdahlovho zákona* vypočítame maximálne teoretické možné zrýchlenie pre každý šifrovací algoritmus a rozšírime Tab. 7 o nové údaje, dostaneme:

Algoritmus	Rýchlosť [Mbps]	Zaťaženie	Max Zrýchlenie	Max rýchlosť [Mbps]
3des	133,33	90,03%	10,03	1337,35
aes128	329,41	80,83%	5,22	1718,37
aes256	254,55	83,26%	5,97	1520,58
arcfour	622,22	56,49%	2,30	1430,07
blowfish	400,00	73,72%	3,81	1522,07
cast128	294,74	81,36%	5,36	1581,21

Tab 8: Maximálne zrýchlenie nástroja scp akceleráciou operácie šifrovania

Pre každý algoritmus vychádza rôzny koeficient maximálneho zrýchlenia, ale výsledná priepustnosť nástroja *scp* sa po akcelerácii ukazuje rádovo podobná pre všetky prípady. Tá je ovplyvnená ostatnými charakteristikami nástroja, ako napríklad dĺžka používaných paketov a z toho vyplývajúca záťaž pri zachytávaní paketov.

Ak by sme brali do úvahy, že akcelerovanie šifrovania nebude možné maximálne, ale len do určitého stupňa, tak výsledné zrýchlenie nástroja bude závislé od miery akcelerácie operácie.



Obr. 18: Závislosť zrýchlenia nástroja scp od miery akcelerácie operácie

Šifrovanie je operácia, ktorú využíva množstvo nástrojov, protokolov a v budúcnosti sa môže počítať ešte s väčším nárastom. Preto je veľmi vhodné ju akcelerovať. Práca [31] predstavuje *FPGA* architektúru, ktorá umožňuje hardvérovo realizovať šifrovaciu jednotku podporujúcu *DES/3DES* algoritmus. Jedna jednotka má priepustnosť 200 Mbps. Do jednej zostavy je možné zapojiť viaceré jednotky a dosiahnuť tak priepustnosť viac ako 1 Gbps, avšak len pre viaceré sieťové prúdy zároveň.

6 Záver

Cieľom tejto práce bola hlboká analýza sieťových nástrojov. Najprv predstavila zástupcov kategórií sieťových aplikácií, potom identifikovala ich jednotlivé operácie za účelom hľadania vhodných miest pre hardvérovú akceleráciu. V spojitosti s tým boli predstavené aj platformy, ktoré programovanie hardvéru umožňujú.

Výsledkom meraní sa niektoré operácie ukázali veľmi vhodné na HW akceleráciu, pretože nadmerne zaťažujú systém a tým znižujú aj výkon samotného sieťového nástroja. Ak by boli hardvérovo akcelerované, umožnilo by to niektorým nástrojom pracovať 10 až 20-krát rýchlejšie a účinnejšie. Vlastná hardvérová akcelerácia by mala byť obsahom ďalšej práce.

Od pôvodného zadania bakalárskej práce sa na základe dohody s vedúcim upustilo a zvolila sa podrobnejšia analýza pred realizovaním hardvérovo akcelerovaného nástroja. Ciele tohto zadania som splnil a výsledky zhrnul v predchádzajúcej kapitole.

Merania výkonu jednotlivých operácií boli vykonané množstvom softvérových aj hardvérových nástrojov. Boli pripravené pomocné aplikácie a automatické meracie skripty, ktoré sú obsahom prílohy.

Celkový prínos práce by sa dal označiť ako podrobný prieskum možností a obmedzení súčasných sieťových nástrojov. Upozornila na ich nedostatky a odhadla ich vylepšenia využitím nových moderných technológií.

A Literatúra

[1] KRETCHMAR, J. M.: *Administrace a diagnostika sítí: pomocí OpenSource utilit a nástrojů*. Brno: Computer Press, 2004. 216 s. ISBN 80-251-0345-5.

[2] LYON, G.: *Insecure.Org: Top 100 Network Security Tools (online)*. 2006. Online: <http://sectools.org/>

[3] SOFTPEDIA: *Network Tools (online)*. [cit. 13.5.2009] Online: <http://www.softpedia.com/get/Network-Tools/>

[4] COAR, K: *The Open Source Definition*. Open Source Initiative. 2006. Online: <http://www.opensource.org/>

[5] CASE, J. D.; FEDOR, M.; DAVIN, J. R.: *RFC1157 - Simple Network Management Protocol*. Network Working Group. 1990. Online: <http://www.faqs.org/rfcs/rfc1157.html>

[6] WIKIPEDIA: *Paket*. [cit. 12.5.2009]. Online: <http://cs.wikipedia.org/wiki/Paket>

[7] CLAISE, B.: *Cisco Systems NetFlow Services Export Version 9*. Network Working Group. 2004. Online: <http://netflow.caligare.com/rfc_3954.txt>

[8] DERI, L.: *Improving Passive Packet Capture: Beyond Device Polling*. NETikos S.p.A. [cit. 13.5.2009] Online: <http://luca.ntop.org/Ring.pdf>

[9] SLABY, J.: Rapid Data Transfers on COMBO Platform, Diploma Thesis, Brno: Masarykova univerzita, 2008. Online: <http://is.muni.cz/th/98734/fi_m/>

[10] JACOBSON, V.; LERES, C.; MCCANNE, S.: *Tcpdump – dump traffic on network (Manual)*. Lawrence Berkeley National Laboratory. 2008. Online: <http://www.tcpdump.org/tcpdump_man.html>

[11] MCCANNE, S.; JACOBSON, V.: *The BSD Packet Filter: A New Architecture for User-level Packet Capture*. Lawrence Berkeley Laboratory, 1992. Online: <http://www.tcpdump.org/papers/bpf-usenix93.pdf>

[12] PUŠ, V.: *Klasifikace paketů s využitím technologie FPGA*. VUT Brno, Brno. 2008.

[13] JOSEY, A. ET AL: *Regular Expressions: Specification, Version 2*. Austin Group, The Open Group Consorium, IEEE. 2004. Online: <http://www.opengroup.org/onlinepubs/009695399/>

[14] HAZEL, P.: *PCRE - Perl Compatible Regular Expressions*. University Computing Service, Cambridge, England. 2008. Online: <http://www.pcre.org/>

[15] HAMPTON, M.; DONAHUE, S., CASTANHO R. P.: *Segmentation and Reassembly of Packets*. 2001.

[16] PETERSON, W. W.; BROWN, D. T.: *Cyclic Codes for Error Detection*. Proceedings of the IRE 49. 1961. Online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04066263>

[17] JACOBSON, V.; LERES, C.; MCCANNE, S.: *libpcap*, Lawrence Berkeley National Labs. 2003. Online: <http://www.tcpdump.org/>

[18] DEGIOANNI, L. ET AL: *Profiling and Optimization of Software-Based Network Analysis Applications.* Proceedings of the 15th IEEE SBACPAD Symposium, Brazil, 2003. Online: <http://winpcap.polito.it/docs/>

[19] GUOCHENG, L.; LU, C.; LING, L.; SHUNYU, Z.: *A FPGA Based Platform for 40G Network Research*. Communication Technology Lab, Intel Corporation. 2007.

[20] LING, L. ET AL: High-performance, Energy-efficient Platforms using In-socket FPGA Accelerators. FPGA 2009. 2009.

[21] WATSON, G.; MCKEOWN, N.; CASADO, M.: *NetFPGA: A Tool for Network Research and Education*. 2nd Workshop on Architecture Research using FPGA Platforms (WARFP). 2006. Online: <http://yuba.stanford.edu/~casado/watson-stanford.pdf>

[22] LOCKWOOD, J. W. ET AL: *NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing*. IEEE International Conference on Microelectronic Systems Education. 2007. Online: <http://netfpga.org/documents/NetFPGA-MSE-2007.pdf>

[23] LIBEROUTER: *Description of COMBO cards.* [cit. 13.5.2009] Online: <http://www.liberouter.org/hardware.php>

[24] KOŘENEK, J.; NOVOTNÝ J.: *NetCOPE, FPGA Platform for High Speed Networking*. Xilinx Academic Forum. 2009. Online: <http://www.liberouter.org/docs/>

[25] BRADNER, S.: *Benchmarking Terminology for Network Interconnection Devices.* Network Working Group. 1991. Online: <http://www.ietf.org/rfc/rfc1242.txt>

[26] CARSTENS, T.: *Programming with pcap*. 2002. Online: <http://www.tcpdump.org/pcap.htm>

[27] MRÁZEK, T.; VYKOPAL J.: *Packet Capture Benchmark on 1 GE*. CESNET technical report 22/08. 2008. Online: <http://www.cesnet.cz/doc/techzpravy/2008/packet-capture-benchmark>

[28] LEVON, J.: *OProfile manual*. Victoria University of Manchester. 2000. Online: <http://oprofile.sourceforge.net/doc/>

[29] KUMAR, S. ET AL.: *Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection*. Washington University in St. Louis. 2006. Online: <http://www.arl.wustl.edu/~sarang/>

[30] HANÁK, J.: Základy paralelného programovania v jazyku C# 3.0. *Amdahlov zákon*, 78 – 83 s. Artax. 2009. 201 s. ISBN 978-80-87017-03-6.

[31] WEE, C. M.; SUTTON, P. R.; BERGMANN, N. W.: *An FPGA network architecture for accelerating 3DES-CBC*. Field Programmable Logic and Applications 2005, 654 – 657s. 2005. ISBN 0-7803-9362-7. Online: <http://www.cs.uq.edu.au/~peters/papers/>

B Zoznam príloh

Príloha 1: Prehľad softvérového vybavenia práce
Príloha 2: CD obsahujúce softvérové vybavenie práce

Príloha 1: Prehľad softvérového vybavenia práce

Adresárová štruktúra:

archives	Zdrojové súbory aktuálnych verzií použítých programov a knižníc: `libpcap-0.9.8.tar.gz; openssl-0.9.7a.tar.gz; pcre-7.8.tar.gz; ssh-1.2.12.tar.gz; wireshark-1.0.7.tar.bz2; ngrep-1.45.tar; oprofile-0.9.4.tar.gz; snort-2.8.3.2.tar.gz; tcpdump-4.0.0.tar.gz;`
test-app	Nástoje preložné s profilovacími informáciami. Program `packetplus` s `Makefile`, ktorý vytvorí dve verzie: `packetplus a packetplus-pf_ring` Upravený `ngrep`
test-scripts	Automatické meracie skripty: `test-decode.sh; test-filter.sh; test-re.sh; test-regex-offline.sh; test-regex.sh; test-ssl.sh;`

Popis skriptov a nástrojov:

Meranie výkonu zachytávania paketov:	
`packetplus`	`./packetplus dev [n]` zachytáva sieťovú prevádzku v promiskuitnom režime na rozhraní `dev`, počíta pakety a vypisuje údaje každých `n` paketov.
Meranie dekódovania paketov	
`test-decode.sh`	Skript meria zaťaženie procesora nástrojom *tshark* a čas potrebný na dekódovanie sieťovej prevádzky zo súbora zadaného prvým parametrom.
Meranie operácie filtrovania	
`test-filter.sh`	1. parameter: meno testu, 2. parameter: počet paketov. Pre rôzne filtre meria čas potrebný na zachytenie určitého počtu vyhovujúcich paketov programom *tcpdump*. Meria zaťaženie na systém, generuje štatistiky, výstup profilácie programu a systémového jadra. Na určenom porte očakáva sieťovú prevádzku.
Meranie operácií reguárnych výrazov	
`test-regex.sh`	1. parameter: meno testu, 2. parameter: počet paketov na odchytenie. Pre rôzne regulárne výrazy meria čas potrebný na zachytenie určitého počtu paketov vyhovujúcich regulárnemu výrazu programom *ngrep*. Meria zaťaženie, generuje štatistiky, výstup profilácie programu a systémového jadra. Skript očakáva na určenom porte sieťovú prevádzku generovanú *Spirentom*.
`test-regex-offline.sh`	1. parameter: regulárny výraz, 2. parameter: cesta k súboru *pcap*. Skript spustí program *ngrep* nad súborom s uloženou sieťovou premávkou a vyfiltruje ju na základe určeného regulárneho výrazu. Meria čas, generuje výsledky profilácie.
`test-re.sh`	Skript porovnáva výkon *pcregrep* a *egrep*. Automaticky vytvára regulárny výraz, rozširuje ho a pre každý odmeria čas potrebný na spracovanie.
Meranie šifrovania	
`test-ssl.sh`	1. parameter: meno testu. Pre viaceré algoritmy meria a porovnáva čas potrebný na prenesenie súboru cez šifrovanú komunikáciu pomocou programu *scp*. Meria zaťaženie na systém, generuje štatistiky, výstup profilácie programu a systémového jadra. Je nutné nastaviť cestu k vzdialenému stroju, súboru a nastaviť veľkosť prenášaného súboru