INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Tato práce vznikla za podpory projektu TeamIt, jenž je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky prostřednictvím grantu ESF CZ.1.07/2.3.00/09.0067.

Budování konkurenceschopných výzkumných týmů pro IT

Výzkumná skupina ANT at FIT:

Sledování síových toků

Technická zpráva

20.srpna 2011

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ VUT V BRNĚ
Božetěchova 2, 612 66 Brno
tel.: +420 541 141 144, +420 541 212 219, fax: +420 541 141 170, 270
http://www.fit.vutbr.cz, http://teamit.fit.vutbr.cz

Obsah

1 Úvod

Sledování síťového provozu je nepostradatelnou součástí při správě, plánování a ochraně počítačové sítě. Dnešní sítě mají složitou a rozsáhlou topologii (fyzickou i logickou), ve které je ověření správného nastavení síťových prvků nebo řešení incidentů jako je zvýšení chybovosti linky, špatné směrování a dalších patologií, bez sledovací infrastruktury jen obtížně proveditelné. Navíc spektrum současných aplikací jako jsou IP telefonie, video konference, burzovní a bankovní transakce, kladou vysoké nároky na spolehlivost, kvalitu a bezpečnost komunikace, což jen podtrhuje význam sledování sítě. Od založení Internetu a jeho masového rozšíření se vyvíjely nejen vzory síťové komunikace, spektrum a počet uživatelů nebo rychlosti linek, ale nevyhnutelně se musely vyvíjet i formy sledování síťového provozu tak, aby odpovídaly aktuálním potřebám uživatelů a správců sítě a byly schopny podporovat například rozhodnutí o navýšení propustnosti linek nebo rozpoznání nežádoucího provozu. Proto dnes v síti najdeme nejen prvky schopné sledovat ty nejzákladnější statistiky o počtu přijatých/odeslaných paketů, ale i mnohem důmyslnější zařízení, která dokáží vzorkovat a ukládat vybrané pakety, vyhledávat dané řetězce znaků v obsahu paketu nebo sbírat statistiky o tocích, tj. o paketech, které si dvě komunikující strany mezi sebou vyměnily.

Právě sledování provozu na úrovni toků se ukazuje jako vhodný typ měření, jenž bude možné využít i v budoucnu. Jeho výhodou je především nezávislost na obsahu paketu a je tudíž imunní o pokusy obejít měření za pomoci šifrování, které je dnes u většiny moderních aplikací běžné. Navíc neporušuje soukromí komunikace uživatelů, tedy neprohledává obsah komunikace, ale vystačí si pouze se záhlavím paketu. V neposlední řadě jsou naměřené výsledky široce využitelné a lze je uplatnit i ve velké většině případů spojených se správou, plánováním a ochranou sítě. Jako příklad lze uvést měření kvality spojení s využitím statistik o mezi-paketových intervalech, kdy u VoIP spojení lze očekávat, že interval mezi dvěma následujícími pakety nepřesáhne určitou dobu (obvykle 20 ms). Toto měření poskytuje informaci o zdroji a cíli komunikace, o komunikující aplikaci, o času a délce trvání komunikace, o množství přenesených dat a podobných statistikách.

Samotné sledování několika toků je jednoduchá a nenáročná činnost, nicméně pro sledování celé sítě a správu páteřních linek je nutné sledovat statisíce současně probíhajících spojení. Sledování toků na linkách s vyšší rychlostí (více jak 1 Gb/s) představuje výzvu současným technologiím, neboť samotná doba mezi dvěma přijatými pakety je kratší než doba přístupu ke statistikám do paměti. Kromě rychlosti linek vrůstá počet uživatelů a aplikací, tím se zvyšuje i počet toků. Daná paměť proto musí být nejen rychlá, ale zároveň pojmout i velké množství dat. Škálovatelnost měření toků se tak jeví jako špatná. To bylo důvodem pro zkoumání přístupů jak tento problém řešit, lze je rozdělit do tří směrů. První z nich je založen na vzorkování příchozích paketů a extrapolaci naměřených dat pro získání původních a nezkreslených statistik. Tato extrapolace má ovšem své limity a bylo ukázáno, že existuje obdoba Nyquistova vzorkovacího teorému i pro síťový provoz, tedy při nižších vzorkovacích frekvencích není možné získat nezkreslené původní hodnoty. Další směr se proto zabýval identifikací a vzorkováním jen takových paketů, které náleží význačným tokům (význačným z pohledu správce). Většinou byl výzkum zaměřen na identifikaci velkých toků, tedy takových, které přenáší velký objem dat (tzv. elephants) a ignorování malých toků (tzv. mice). To umožňuje využít malou paměť a přesně měřit pouze velké toky, zatímco malé zanedbávat. Takové měření lze využít pro specifický druh aplikací, ale pro jiné, například detekci anomálií, jsou takové výsledky nepoužitelné. Posledním směrem, jak snížit náročnost sledování všech toků, je vyšší míra agregace. Například všechny toky, které mají shodnou cílovou IP adresu, je možné sdružit do jednoho toku. Stejně jako u předchozích případů může být tato metoda pro některé aplikace nepřijatelná.

Cílem této práce je shrnout dosavadní stav v oblasti sledování toků. Konkrétně se práce zaměří na popis doposud známých vlasností síťového provozu. Na základě těchto vlastností je práce dále zaměřena na diskuzi o efektivní správě paměti toků pomocí identifikace význačných toků a dále na diskuzi o efektivní indexovaci této paměti. Tato práce je členěna následovně. První část se zabývá popisem stávající infrastruktury pro sledování toků a problémy při spojených s jejich měřením. Následující kapitola pojednává o vlastnostech síťového provozu. Ve třetí části jsou rozebrány metody pro identifikaci intenzivních toků. Poslední kapitola se věnuje postupům, jak optimalizovat přístup ke kontextové informaci náležící jednotlivým tokům.

2 Sledování toků

V prvé řadě je vhodné si ujednotit definici pojmu tok, která se v různých publikacích uvádí odlišně. Nejnovější standard pro sledování a přenos toků IPFIX (IP Flow Information Export) [15] definuje tok jako sekvenci paketů, která za určitý časový úsek protekla kolem místa pozorování. Pakety daného toku mají společnou vlastnost, která je funkcí položek v záhlaví paketu, či charakteristikou paketu samotného, např. jeho délka. Jeden tok si tak lze představit jako všechny pakety v provozu, ale i jako jediný paket. Tato definice je příliš obecná a pokud se mluví v síťové komunitě o toku, pak se má na mysli množina paketů se shodnou pěticí údajů (zdrojová a cílová IP adresa, zdrojový a cílový port, protokol), u kterého interval mezi následujícími pakety nepřesáhne stanovený práh. Pokud nebude brán v úvahu směr komunikace, tedy pokud není rozlišen zdroj a cíl komunikace, pak vznikne obousměrný tok, který nazveme *spojení*.

Sledování toku znamená sběr informací o toku (většinou se jedná o statistické údaje). Tato činnost v sobě zahrnuje několik částí přijem paketů na fyzickém rozhraní, výpočet funkce náležitosti paketu k toku, vyhledání záznamu o tomto toku a uložení/agregace informace o paketu do záznamu. Následně je nutné rozpoznat ukončení toku a získané informace odeslat na kolektor.

Popíšeme nyní sběr informace o toku formálněji. Síťový provoz v určitém intervalu můžeme chápat jako množinu paketů P. Tuto množinu lze rozdělit do vzájemně disjunktních podmnožin $F_1, F_2, \ldots, F_N$ na základě příslušnosti k tokům (N je počet toků v P):

$$\forall i, j, 1 < i < N, 1 < j < N, i \neq j : F_i \cap F_j = \emptyset$$

Záznam r_i o toku i vytvoříme aplikací fce ag na všechny pakety p_j v toku F_i:

$$F_i : r_i = ag(p_j), j = 1, 2, \ldots, \mid F_i \mid$$

Je vidět, že pro sledování toku F_i není potřeba informace (pakety) o toku jiném. Z toho vyplývá, že pakety různých toků můžeme vzájemně libovolně promíchat, aniž bychom ovlivnili výsledek.

Bohužel většina agregačních funkcí je závislá na pořadí příchodů paketů v rámci toku, pak je nutné považovat tok za uspořádanou množinu, kde binární relace uspořádání je definována na základě pořadí příchodů paketů, tedy "přišel dříve než"a opět platí, že pakety různých toků můžeme libovolně promíchat, nesmí se ovšem změnit pořadí paketů v toku.

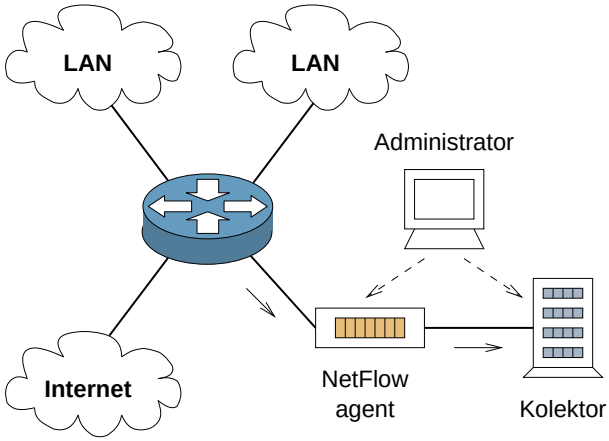
2.1 Infrastruktura pro sledování toků

Mezi široce používané technologie pro sledování toků v síti patří NetFlow [1], která byla představena firmou Cisco koncem devadesátých let. NetFlow je defacto synonymem pro sledování IP toků. Popularita NetFlow je způsobena vhodnou mírou abstrakce, kdy celkový provoz je rozdělen na toky, a ke každému toku je sledována nejen pětice klíčových údajů (zdrojová a cílová IP adresa, zdrojový a cílový port, číslo protokolu), ale i další statistiky jako je počet paketů a bytů, čas počátku a konce toku, nastavené TCP příznaky a další. Získaná data umožňují daleko lépe dohledávat incidenty na síti nebo ukázat vytížení sítě, ladit nastavení QoS (Quality of Service), atd.

Infrastruktura NetFlow (obr. 1) je postavena na dvou typech prvků. Prvním typem jsou prvky (nazvěme je exportéry) schopné sbírat statistiky (záznamy o tocích) a odesílat je pomocí NetFlow protokolu. Druhým typem prvku je kolektor, který přijímá naměřené statistiky a ukládá je pro další analýzu. Exportéry mohou být implementováni v routerech nebo autonomních sondách, které jsou umístěny do důležitých míst sítě. Například do místa připojení lokální sítě do Internetu, do uzlů kampusových sítí nebo do datových center. Kolektor může být umístěn kdekoliv v síti a sbírat informace z několika exportérů zároveň. Administrátor sítě přistupuje k datům přes webové rozhraní nebo přes terminál.

Díky své jednoduché struktuře má NetFlow v5 širokou podporu jak na straně exportérů, tak i kolektorů. NetFlow v5 formát záznamu o toku obsahuje pouze zdrojovou a cílovou IPv4 adresu, zdrojový a cílový port, číslo protokolu, čas začátku a konce toku, počet přenesených paketů a byte, TCP příznaky, pole ToS/DiffServ a číslo síťového rozhraní, na kterém byl tok měřen.

V dnešní době se tento formát jeví jako limitující a přechází se na flexibilní záznam definovaný protokolem NetFlow v9 [14]. Nejdůležitější rozdíl mezi v5 a v9 je v zavedení šablon. Šablony slouží pro definici vlastní struktury a položek záznamu. Uživatel si tak definuje, které hodnoty si přeje exportovat a jak velký jim přiřadí prostor. Kolektor tyto šablony zpracuje a na jejich základě interpretuje příchozí záznamy.



Obrázek 1: Infrastruktura sledování toků

Organizace IETF (Internet Engeneering Task Force) začala pracovat na obecné, daleko obsáhlejší definici protokolu pro přenos záznamů o tocích z názvem IPFIX [15] (Internet Protocol for Flow Information Export). Očekává se, že IPFIX nahradí NetFlow a v dnešní době již existuje několik prototypových implementací exportérů a kolektorů schopných práce s IPFIX protokolem.

2.2 Problémy spojené se sledováním toků

Novou specifikací protokolu pro export dat IPFIX byl odstraněn nedostatek spojený s omezenou množinou reportovaných položek. Flexibilita tohoto protokolu umožňuje vybírat položky záznamu ze široké palety definovaných položek anebo definovat nové se specifickým významem. Lze tak podpořit větší množinu aplikací využívajících data o tocích. Z pohledu těchto aplikací je důležité pracovat s nezkreslenými statistikami, v opačném případě nelze data pro aplikaci využít nebo dochází k častým selháním.

Sběr informací o tocích vyžaduje několik úkonů: příjem paketů na fyzickém rozhraní, nalezení relevantních položek v záhlaví paketu, výpočet funkce náležitosti paketu k toku, vyhledání záznamu o tomto toku a uložení/agregace informace o paketu do záznamu. Následně je nutné rozpoznat ukončení toku a získané informace odeslat na kolektor.

Příjem paketů musí být bezeztrátový, tedy na rychlosti linky. Zároveň je svázán s přiřazením přesné časové značky. Tato značka musí mít dostatečné rozlišení tak, aby dvěma po sobě následujícím paketům byla přiřazena unikátní značka. V případě, kdy uživatelská aplikace pracuje s daty pouze z jedné sondy, přesnost této značky v čase nemusí být velká. Naopak v situacích, kdy je nutné porovnávat data z více sond, je nezbytná vysoká přesnost časové značky, aby bylo možné párovat interakce v síťovém provozu. Běžné síťové karty takovou operaci nepodporují a časová značka je paketu přidělena až při přenosu do paměti počítače. V situacích, kdy je nutná přesná časová značka, se využívá specializovaných síťových karet napojených na GPS modul.

Samotný příjem paketů na fyzické vrstvě nepředstavuje problém u žádného typu síťové karty, neboť je řešen v podobě specializovaného čipu. Problém nastává v okamžiku přenosu veškerého síťového provozu do paměti počítače, kdy úzkým hrdlem může být koncepce komunikace síťové karty a jejího driveru. Proto specializované síťové karty využívají alternativní druh komunikace, např. techniku ,,polling" [19] na místo využívání přerušení.

Experimenty bylo zjištěno, že optimalizované programové vybavení běžící na tradiční architektuře personálního počítače se zapnutou podporou efektivní komunikace síťové karty umožňuje měřit linky s rychlostí až 1 Gbps [19]. Pro vyšší rychlosti je nutné využít hardwarové akcelerace v podobě specializované síťové karty, která umožní jednu z následujících operací:

- distribuovat síťové toky mezi více procesorů s vlastní pamětí [18] nebo
- implementovat měřící proces přímo ve hardwarovém vybavení karty [62].

V obou případech může docházet k omezení výkonnosti v důsledku úzkého místa v následných fázích sledování toků. To je způsobeno omezenou dobou, kterou má celý systém k dispozici pro zpracování jednoho paketu. Tab. 1 zobrazuje intervaly pro zpracování nejkratšího paketu pro několik rychlostí síťových linek.

Tabulka 1: Interval mezi příchody nejkratších paketů

Rychlost [Gbps]	10	40	100
Max. počet paketů za sekundu	$1,49.10^7$	$5,9.10^7$	$1,49.10^8$
Doba pro zpracování paketu [ns]	67	17	7

První z potencialně kritických operací je extrakce relevantních údajů ze záhlaví paketu. Tato operace spočívá v nalezení a duplikaci hodnot polí ze záhlaví protokolů na síťové, transportní a aplikační vrstvě modelu TCP/IP do unifikované struktury. Na vyšších rychlostech je nutné extrakci dat z paketu paralelizovat, nejjednoduššeji duplikací extrakčních procesů. Nejen pro účely sledování toků je nutné při extrakci dodržet pořadí paketů v toku tak, jak byly přijaty (viz výše). Tento fakt vyžaduje distribuci paketů mezi paralelně běžící extrakční procesy na základě příslušnosti paketu k toku, v opačném případě by mohlo dojít k záměně pořadí paketů v toku, pokud by zpracování dříve příchozího paketu trvalo déle než následujícího. Bohužel při distribuci paketů nemůže být příslušnost paketu k toku spočítána, neboť tato informace je k dispozici až po extrakci dat z paketu. Obecně je proto nutné připustit záměnu pořadí a následně tyto záměny řešit pomocí seřazovací paměti. Z toho vyplývá, že extrakce dat z paketu je dobře paralelizovatelná za přispění relativně malého množství paměti. Pokud jsou tedy k dispozici příslušné hardwarové prostředky, extrakce dat z paketu není úzkým místem při sledování toků.

Po extrakci dat je spočítána funkce příslušnosti paketu k toku, jejímž výstupem je adresa toku v paměti. V reálném systému tato operace znamená vytváření indexové struktury v paměti, jejímž prohledáním zjistíme adresu příslušného toku. Kromě triviálního přístupu, kdy je funkce příslušnosti vypočtena přímo transformací položek klíče, je nezbytné využít paměť. Tato paměť by měla pojmout celou indexovou strukturu a zároveň doba pro vyhledání adresy toku by neměla překročit dobu pro zpracování jednoho paketu. Efektivní indexace záznamů je klíčová pro zmenšení paměťových nároků, respektive zvýšení zaplněnosti paměti, doby vyhledání a míry kolizí záznamů. Tato fáze se jeví jako kritická a je jí věnována kapitola 5.

Časově kritickou fázi měření toku představuje aktualizace záznamu. Tato operace se skládá ze získání záznamu z paměti, aktualizace statistik v záznamu a jeho opětovné uložení do paměti. Celková doba tohoto úkonu přímo ovlivňuje propustnost sondy. Navíc tato operace musí být v rámci daného toku atomická, tzn. není možné zpracovávat daný záznam v několika paralelních jednotkách. Je nezbytně nutné tuto operaci maximálně optimalizovat. Nabízí se využití hierarchie pamětí různých typů, které většina platforem nabízí. Typicky se jedná o kombinaci pomalé-velké a rychlé-malé paměti. Nicméně vlastnosti síťového provozu se liší od klasických dat, se kterými běžně programy pracují, kdy přímočaré schéma využití cache selhává. Proto tato práce shrnuje vlastnosti síťového provozu v kapitole 3. Na ní navazuje kapitola 4 popisující přístupy, které by mohly pomoci efektivně využít cache i v síťové oblasti.

Sondy dnes exportují pakety s celými záznamy. Uvážíme-li, že míra agregace příchozího síťového provozu vůči počtu ochozích záznamů je na gigabitových sítích 1:10 vůči paketům a 1:200 vzhledem k počtu bytů, pak při monitorování deseti-gigabitové linky může export vytvořit až 50 Mbps proud dat, který může být problém přenést a uložit na kolektoru. Proto se jeví jako vhodné provádět předzpracování záznamů přímo na sondě a snížit tak odchozí tok, resp. distribuovat výpočetní zátěž kolektoru na sondy. Přestože je tato oblast velice zajímavá je mimo rozsah této práce.

3 Vlastnosti síťového provozu

Návrh a konstrukce robustního síťového zařízení vyžaduje kvalitní rozbor síťového provozu, tedy dat, nad kterými bude zařízení pracovat. Analýza vstupních dat umožňuje předvídat a eliminovat, pokud je to možné, úzká místa v navrhovaném systému.

3.1 Soběpodobnost a heay-tail rozložení

Představa o síťovém provozu se v průběhu minulých let vyvíjela. Zpočátku panovalo přesvědčení, že síťový provoz je podobný klasickému telefonnímu a tudíž, že většina jeho vlastností se řídí Poissonovým rozložením [52]. Empirické studie reálných vzorků dat ale ukázaly, že síťový provoz je tzv. sobě podobný ve velkém rozsahu časových měřítek, což odporuje teorii o Poissonovu rozložení. Představíme-li si některou vlastnost síťového provozu (např. intervaly mezi příchody paketů) jako řadu hodnot a začneme přes tuto řadu počítat klouzavý průměr s různě velkým oknem, pak u Poissonova rozložení se očekává, že čím většího okna se použije, tím více bude výsledná řada hodnot hladší, tedy bude se snižovat rozptyl dat. To ale u síťového provozu neplatí, což vedlo k závěru, že většina vlastností síťového provozu se řídí tzv. heavy-tail rozložením [42]. Tedy takovým, kdy tvar rozložení pravděpodobnosti se asymptoticky blíží hyperbole.

$$Pr[X > x] \sim x^{-a}, x \to \infty, 0 < \alpha < 2.$$

To znamená, že převážná většina vzorků dat bude mít nízkou hodnotu, ale zaroveň existuje malá skupina vzorků s velmi vysokou hodnotou, která tvoří významnou část celkové sumy hodnot vzorků. Existují specifické případy heavy-tail rozložení, například v ekonomii se využívá Pareto rozložení k popisu rozložení bohatství mezi populaci. Odtud pochází i známá poučka, že 80% bohatství je vlastněno 20% populace. Pareto rozložení bylo identifikováno v mnoha dalších oblastech, například v rozsahu lesních požárů, velikosti částic v písku nebo velikosti souborů na pevném disku. V souvislosti se síťovým provozem bylo heavy-tail rozložení pozorováno u různých typů provozu, jako je webový provoz, FTP, TELNET, ale i na nižších úrovních modelu ISO/OSI jako je TCP a Ethernet.

Usuzuje se, že příčina heavy-tail rozložení vlastností síťového provozu je pravděpodobně způsobena rozložením velikosti souborů [46] , chováním uživatelů či aplikací [16,63] a dynamikou Ethernet provozu [34, 40,51,67]. Heavy-tail rozložení charakteristik tak představuje zcela nový problém při analýze a zpracování síťového provozu, kdy nelze využít dříve používaných předpokladů, a vzniká tak zcela nová sada problémů.

3.2 Složení provozu a sledované charakteristiky

Nejdůležitější sledované charakteristiky pro jeden tok síťového provozu jsou dostupná propustnost/intenzita, počet paketů za daný časový úsek, velikosti paketů, ztrátovost paketů, doba oběhu (Round Trip Time – RTT). Pří sledování celkového obrazu se tyto charakteristiky sledují pro každý tok zvlášť a navíc se sledují statistiky o počtu nově vzniklých toků za daný časový úsek či rozložení toků mezi aplikace. Právě aplikace ovlivňují nejvíce složení síťového provozu.

V průběhu mnoha let vývoje Internetu nastaly tři význačné éry [58]. První z nich lze nazvat érou textových aplikací, kdy většina provozu náležela emailové komunikaci, přenosům souborů a USENET skupinám. V tomto období bylo 80% provozu přenášeno přes TCP (většinou SMTP a FTP) a zbylých 20% patřilo UDP (DNS, Telnet) [9, 10]. Následovala éra hyperlinková, kdy podíl HTTP provozu vzrostl s malého procenta až na 75% celkového provozu a stejně tak TCP dosáhlo svého současného podílu, tedy 95% provozu, zbytek náleží UDP [9]. Poslední éra přetrvávající dodnes se nazývá multimediální, neboť provozu dominují multimediální aplikace (hovory, videokonference, sdílení multimediálních souborů) používající P2P paradigmatu. V současné době je podíl P2P aplikací v provozu vyšší, asi 55% bytů, u HTTP zhruba 40% nicméně, HTTP provoz je odpovědný za větší množství toků, 38% oproti pouhým 4% u P2P [23].

Přestože známe nejčastěji používané aplikace, není možné přímo odvodit výsledné vlastnosti provozu, které jsou ovlivněny dalšími faktory, jako je topologie sítě, chování uživatelů a další. Existují však představy podpořené měřením provozu na reálné síti. Mezi nejvýznamnější vlastnosti, které jsou relevantní při sledování toků, patří velikost toků, jejich doba trvání, intenzita, u všech těchto charakteristik se přepokládá heavy-tail rozložení. To bylo potvrzeno i několika pracemi, které navíc zavedly určitou klasifikaci toků založenou na jejich chování.

3.3 Třídy toků a jejich vlastnosti

První z prací, která se věnovala vlastnostem toků, byla publikována již v roce 1986 [34] a autoři zjistili, že pakety toku přicházejí ve shlucích, kdy po období dlouhých intervalů mezi pakety, může následovat

období velice krátkých. Tento proces může být pozorován ve více časových úrovních, tedy první důkaz soběpodobnosti síťového provozu. Další práce se zabývali distribucí velikosti toků ve smyslu přenesených dat/paketů, intenzitou nebo trváním toku, všechny se závěrem, že se tyto vlastnosti řídí heavy-tail rozložením. Význačné toky, tedy z obou konců rozložení, dostaly i jednotlivá pojmenování. Podle množství dat v toku byly rozděleny na slony a myši, kde slony tvoří jen malé množství toků, jenž přenáší většinu dat a naopak myši tvoří velké množství toků, které přenáší výrazně méně provozu. Kromě množství dat byly toky pojmenovány i dalších dimenzích. Z pohledu délky trvání byly rozděleny na krátké (vážky) a dlouho-žijící (želvy), intenzity na rychlé (gepardy) a pomalé (šneky) a poslední podle shlukovosti na toky s velkou shlukovostí (dikobrazy) a malou (rejnoky).

Definice jednotlivých tříd se v literatuře liší a nicméně využijme systematickou definici těchto toků publikovanou v [39]. V tomto článku Heidemann definoval význačné toky podle dvou prahových hodnot, které vypočítal jako průměr charakteristiky plus/mínus třikrát její směrodatnou odchylku. Například, pokud velikost toku s převyšovala horní hranici, pak byl označen jako slon, pokud byla velikost toku menší než dolní hranice, pak byl tok označen jako myš. Stejně postupoval pro délku trvání d i intenzitu r. Měření posledního sledovaného parametru – shlukovosti, Heidemann definoval několika způsoby, z nichž vybral následující. Shlukovost je definována jako sekvence paketů v toku, mezi nimiž je interval menší než práh t. Velikost shluku je počet bytů přenesených ve shluku. Délka trvání shluku je doba mezi příchodem prvního a posledního paketu shluku. Intezita shluku je podíl velikosti a délky shluku, vyjma jedno-paketových shluků. Samotná shlukovost je pak definována jako součin průměrné intenzity shluků a průměrem intervalů mezi shluky.

V několika publikacích [5, 26, 60] byly studováni sloni a myši. Jejich výsledky potvrdili, že převážná většina provozu ve smyslu bytů je přenesena malým počtem toků, zatímco existuje mnoho toků, které tvoří jen malé procento celkového provozu. Například v [58] autoři využili několikaletý vzorek provozu mezi Japonskem a západním pobřežím Spojených států, aby demonstrovali tuto skutečnost. Obr. 2 ukazuje poměr největších deseti toků na celkovém provozu. Tento poměr v průběhu let klesal s rostoucím počtem toků na síti.

Obrázek 2: Zastoupení deseti největších toků v celkovém oběmu dat (převzato z [45])

Existence velkého množství malých toků byla jasně prokázána například v článku [13], který analyzuje síťový vzorek osahující až 40% toků s jediným paketem.

Následně článek [8] věnovaný analýze délce trvání toků reportuje, že 45% toků trvá méně než 2 s, 53% trvá mezi 2 s a 15 minutami a 12% trvá déle. Studie síťového provozu aktuálnějšího vzorku dat ze sítě MAWI ukázala, že až 70% toků je kratších než 10 s. Heidemann [39] se věnuje měření propustnosti toku a zjišťuje, že 80% toků jsou šneci s propustností menší než 10 kB/s, a jen 2% toků gepardi s více jak 100 kB/s.

Savrotham [56] ukázal, že shlukové chování je způsobeno jen několika toky s velkým objemem dat a je daleko pravděpodobnější, že tyto toky jsou odpovědné za blokování linky. Jeho výsledky jsou konzistentní i s výsledky v [39], přestože oba autoři definovali shlukovost odlišně.

Jednotlivé rozdělení toků do tříd vzniklo v různých publikacích, podle toho, kterou vlastností se daná publikace věnovala. Ucelený obraz o vlastnostech toků, jejich vzájemné korelaci podal až Heideman [39]. Zároveň podává i vysvětlení, co danou vlastnost a její korelaci k ostatním vlastnostem způsobuje, tedy jak aplikace a uživatelé ovlivňují chování síťového provozu.

Tabulka 2: Vzájemný vztah mezi slony, želvami, gepardy a dikobrazi (převzato z [39])

height	**Sloni**	**Želvy**	**Gepardi**	**Dikobrazi**
Sloni	-	6%	3%	68%
Želvy	20%	-	0.007%	8%
Gepardi	7%	0.004%	-	3%
Dikobrazi	19%	1%	4%	-

Tab. 2 ukazuje vzájemný vztah mezi různými třídami toků na dvou vzorcích síťového provozu. Mezi

významné výsledky patří zjištění, že převážné procento dikobrazů (70%) jsou sloni, tedy existuje silná korelace mezi velkými a shlukovými toky. Výsledky změřené na obou vzorcích se výrazně liší pro korelaci dikobrazů a gepardů, což autoři vysvětlují slabým DNS provozem v prvním vzorku dat a vyvozují konstatují, že druhý vzorek dat obsahuje shlukové DNS toky, které tvoří 60% gepardů.

Tabulka 3: Zastoupení prvních čtyř aplikací ve význačných tocích (převzato z [39])

Sloni	**Želvy**	**Gepardi**	**Dikobrazi**
web(67%)	DNS(51%)	web(53%)	web(71%)
kazaa(5%	web(15%	DNS(28%)	SMTP(10%)
telnet(3,5%)	telnet(9,1%)	ftp(5%)	ftp(6%)
gnutella(2%)	ftp(5%)	smtp(3,3%)	nntp(2,1%)

Tab. 3.3 zobrazuje pět nejvýznamnějších aplikací pro každou třídu. Konkrétní aplikace je určena pouze na základě čísla cílového portu přiřazeného organizací IANA. To může vést ke zkreslení výsledků zejména pro P2P aplikace, jenž používají čísla portů jiných aplikací (snaží se tak maskovat svůj provoz) nebo dynamicky přidělované. Je vidět, že HTTP a P2P provoz tvoří většinu provozu, což je potvrzení výsledků prezentovaných v [55]. Heidemann navíc uvádí, že webový provoz je odpovědný za většinu rychlého a shlukového přenosu. Dále, že více jak 50% dlouho trvajících toků je DNS provoz, jehož toky se navíc chovají jako gepardi a dikobrazi zároveň.

Tab. 3.3 přehledně shrnuje vlastnosti význačných toků a jejich chování, kdy byla odvozena z mnoha pozorování.

Tabulka 4: Chování význačných toků (převzato z [39])

Kategorie	**velikost**	**doba**	**propustnost**	**shlukovost**
Sloni	A	A	N	N
Želvy	N	A	N	N
Gepardi	N	N	A	A
Dikobrazi	A	N	A	A

Například více jak 95% gepardů jsou krátké toky trvající méně než jednu sekundu. Asi 50% sloních toků trvá přes dvě minuty, což naznačuje, že sloni žijí dlouho, nicméně pouze 6% sloních toků jsou želvy. Nízká korelace želv a slonů je pravděpodobně způsobena rozhodnutím uživatelů nestahovat velké objemy dat přes nízko kapacitní linky. Velké procento dikobrazů (65%) žije méně než deset sekund a 95% žije méně než dvě minuty. Vzhledem k tomu, že dikobrazi a sloni jsou poměrně korelováni, naznačuje to, že převážná část dikobrazů je tvořena sloními toky, kterou jsou kratší jak 2 minuty. Na základě tohoto pozorování Heideman usuzuje, že většina dikobrazích toků je tedy způsobena přenosy velkých souborů po rychlých linkách, což je konzistentní výsledek s [56]. U gepardů je možné pozorovat, že jsou krátké a malé a dobře korelují s dikobrazi, kde tvoří doplněk ke sloním tokům, které nejsou tak rychlé.

3.4 Diskuze o provozu

Výše uvedené vlastnosti například velikost toku, byly vždy vztaženy k objemu přenesených dat, tedy k bytům. Předpokládá se, že stejná zjištění o vlastnostech toků platí i pokud měrnou jednotkou jsou pakety. Samotné rozložení délek paketů v síťovém provozu není ovšem rozhodně uniformní, ale je bimodální [55, 60], kdy většina paketů má buď délku do 100 bytů (60%) nebo 1280 až 1500 Bytů (20% paketů). Je možné tedy očekávat mírné zkreslení. Většina výše citovaných prací zabývajících se zkoumáním slonů pracovala s různými stupni agregace toků, nicméně heavy-tail rozložení bylo pozorováno ve všech vzorcích.

Podle různých studií se nároky na propustnost linek zvyšují o 40–60% ročně [2, 48], kdežto počet uživatelů Internetu roste průměrně každý rok o 10% [3], hlavní podíl tak na růstu objemu přenesených dat mají nově vznikající aplikace (např. YouTube, VoIP, IPTV, a další). Předpovědi růstu Internetového

provozu neočekávají, že by se rostoucí trend měl v příštích letech zastavit. Lze navíc očekávat vznik nových protokolů, podmíněný zastaralostí některých protokolů pro moderní sítě a aplikace.

4 Identifikace význačných toků

Jak již uvedla předchozí kapitola, většina vlastností síťového provozu se řídí heavy-tail rozložením. Z toho vyplývá, že lokalita toků, ve smyslu pravděpodobnosti, že se v N následujících paketech bude vyskytovat paket stejného toku, bude velice špatná. Samozřejmě stále platí, že existuje několik význačných toků, u kterých lokalita bude dobrá, díky jejich velikosti a intenzitě. Myšlenka sledování pouze význačných toků je velice lákavá, neboť těchto toků je málo, zároveň zahrnují většinu provozu a pro jejich monitorování teoreticky vystačíme s malým množstvím paměti. Nicméně díky heavy-tail rozložení toků může být problémem takové toky rozpoznat a uchovat v paměti dostatečně dlouho. Vznikly proto různé postupy zabývající se identifikací význačných toků (především slonů), kterým je věnována tato kapitola.

4.1 Prediktory

Prediktory pro určení propustnosti toku jsou dvojího druhu: založené na vzorci a založené na historii. Oba typy prediktorů jsou zaměřeny především na odhad propusnosti TCP toků.

Prediktory založené na vzorci zakládájí své předpovědi znalosti na matematického modelu sítě, kdy predikci vyjadřují jako funkci vlastností dané cesty (např. doba oběhu, ztrátovost paketů). Výhoda tohoto typu prediktoru spočívá v tom, že nepotřebuje znalost o předchozích přenosech. Navíc výpočet predikce je poměrně nenáročný a neintruzivní, probíhá zjištěním doby oběhu a ztrátovosti cesty a následným výpočtem očekávané propusnosti. Běžně se tyto parametry měří a priori před začátkem samotného toku, což ale může vnést chyby do odhadu. Nicméně v literatuře se uvádí výsledky pouze pro tyto situace, neboť se předpokládá, že prediktor má sloužit k nastavení správné rychlosti generování paketů TCP toku [31]. Tedy není možné ověřit parametry cesty až po začátku spojení. Můžeme spekulovat, že takový prediktor lze využít pro identifikaci význačných (sloních) toků, kdy dobu oběhu i ztrátovost paketů lze měřit přímo na samotném toku.

Mezi nejznámnější prediktor tohoto typu je založen na následující formuli [50]:

$$E[R] = min\left(\frac{M}{T\sqrt{\frac{2bp}{3}} + T_0 min(1, \sqrt{\frac{3bp}{8}}p(1+32p^2))}, \frac{W}{T}\right),$$

kde $E[R]$ je odhadovaná propusnost, T_0 je doba pro znovu zaslání (retransmit) nepotvrzeného paketu, W je velikost odesílacího okna, b je počet potvzených paketů jením ACK paketem, p a T jsou průměrná ztrátovost paketů a doba oběhu.

Druhý typ využívá pozorování předchozích přenosů pro predikci propusnosti aktuálního toku. Tento typ využívá klasických postupů pro predikci hodnot časových řad pro určení přenosu propustnosti aktuálního toku. Mezi mnoha prediktory se vhodným pro určení propusnosti již běžícího přenosu jeví využití tzv. růstové strategie [44]. Ta využívá celkový počet přenesených bytů a strmost růstu propusnosti v předchozích intervalech pro určení sloních toků. Algoritmus se snaží pomocí lineární extrapolace předpovědět počet přenesených bytů m, tedy:

$$m = S + gT,$$

kde S je počet doposud přenesených bytů, g je míra růstu a T je měřící časové okno. Hodnotu g rekurentně vypočteme s příchodem každého paketu toku takto:

$$g_{i+1} = \alpha g_i + (1-\alpha)b,$$

kde b je velikost paketu a α je zvolená konstanta, pro níž platí $0 \geq \alpha \geq 1$. Doporučuje se zvolit $T = 1/(1-\alpha)$. Pak se výraz pro odhadovaný počet bytů zjednodušší na

$$m = S + \frac{g}{1-\alpha}.$$

Pokud $m >$ práh, pak se tok považuje za význačný. Bohužel analýzu pro volbu konstant α a prahu Molina [44] neuvádí.

U uvedených prediktorů chybí analýza, jak rychle, tedy za jak dlouho od počátku toku, jsou schopny správně předpovědět propusnost toku. V současné době se pro určování sloních toků využívá spíše heuristik, kterým se věnuje následující kapitola.

4.2 Heuristiky

V mnoha případech je výhodnější využít heuristik spíše než prediktorů, neboť jejich výpočetní náročnost může být nižší a výsledky přesnější (jednou z příčin může být například dynamika provozu). Mnohdy může být také problematické zjistit vstupní parametry pro prediktory, které heuristika nepotřebuje.

Nejjednodušším způsobem identifikace sloních toků je využít vzorkování bytů. Například bude-li nastavené vzorkování vstupního toku na poměr 1:10000 (vzorkuje se paket obsahující 10000-ící byte), pak jednopaketové toky mají jen malou pravděpobodnost, že jejich paket bude navzorkován. Naproti tomu tokům se vzrůstající velikostí roste šance na navzorkování alespoň jednoho z paketů.

Na vzorkování je založena i metoda Sample-and-Hold publikovaná v [24]. Oproti klasickému vzorkování přidává následující modifikaci přidáním paměti.

- Pokud existuje záznam pro paket v paměti, pak tento paket vždy navzorkujeme.
- Pokud neexistuje záznam pro paket v paměti, pak ho navzorkujeme s pravděpodobností p. V případě, že byl paket navzorkován, založíme záznam v paměti.

Analýza tohoto algoritmu uvedená v [24] ukazuje jeho sílu. Představme si situaci, kdy si přejeme sledovat toky, které zabírají více než 1% kapacity přenosového pásma C v daném intervalu. Těchto toků nemůže být více než 100, proto paměť se 100 položkami by měla stačit na jejich uložení, nicméně dovolíme využít paměť větší například s 10000 položkami. Pravděpodobnost p navzorkování paketu/záznamu zvolíme tak, aby bylo těchto 10000 položek zaplněno $p = 10000/C$. Uvažujme tok F zabírající více než 1% kapacity, pak tento tok posílá více než $C/100$ bytů. Protože je každý byte je vzorkován s pravděpodobností $p = 10000/C$, pak pravděpodobnost, že by tok nebyl v paměti je $(1-10000/C)^{C/100}$, což je blízko e^{-100}. Podobně, pravděpodobnost, že tok F bude v paměti již po 5% svého provozu, je $1 - e^{-5}$, což je více než 99%.

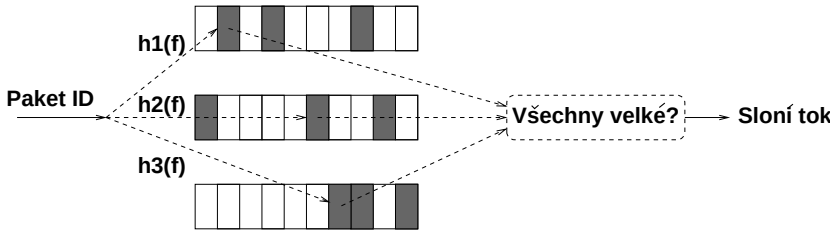
Nevýhodou je nutnost předimenzovat paměť tak, aby metoda byla úspěšná. Tento nedostatek se snaží eliminovat algoritmus publikovaný v [41]. Základní myšlenkou je náhodně zvolit tok a sbírat jeho velikost co do počtu bytů to čítače. Zároveň je tento čítač dekrementován v pravidelných intervalech o část kapacity linky C, například pro sledování toků s více jak 1% provozu je to $C/100$. Pokud je velikost toku je větší než 1% kapacity linky, pak se hodnota v čítači bude zvětšovat až překročí určitý práh a tok bude identifikován jako sloní a vložen do speciálního listu, v opačném případě je ignorován a pokračuje se sledováním dalšího náhodně zvoleného toku. Použitím několika takových čítačů, tedy paralelním sledováním několika toků současně, se sníží procento toků, které by nebyly analyzovány. Autoři této metody tvrdí, že její výsledky jsou srovnatelné s výsledky prezentovanými pro Sample-and-Hold a zároveň vystačí s mnohem menším množstvím paměti. Otázkou zůstává, zda tyto výsledky nebyly závislé na provozu, který byl použit. Zároveň chybí analýza rychlosti, se kterou tato metoda je schopna identifkovat velké toky.

Další heuristika, která je v dnešní době široce využívána, je založena na pravděpodobnostní struktuře o několika stupních, odtud její název Multistage Filters [24].

Základní myšlenka tohoto algoritmu je znázorněna na obr. 3. Paket náležící toku F je namapován do několika paralelních polí pomocí několika různých hash funkcí. Každá položka pole obsahuje čítač, který je inkrementován délkou příchozího paketu. Pokud jsou všechny adresované čítače daného toku nad určeným prahem, pak se jedná o sloní tok.

Samozřejmě nastává situace, kdy se malý tok namapuje na některé čítače velkého toku. Přidáváním dalších paralelních polí se exponenciálně snižuje pravděpodobnost, že malý tok bude identifikán jako velký, neboť všechny čítače musí překročit daný práh.

Analýza uvedená v [24] podává důkaz o síle tohoto algoritmu na konkrétním příkladu. Uvažujme 100 Mbps linku se 100000 toky, kdy je cílem je identifikovat toky s více než 1% provozu. Analyzujme



Obrázek 3: Schéma Multistage Filters

pravděpodobnost, že malý tok se 100 Kbps bude označen chybně. To by musel být označen všemi paralelními poli jako velký. Předpokládejme, že každé pole má 1000 čítačů. Aby byl 100 Kbitový tok označen jako velký, musí ostatní toky přispět do stejného čítače 900 Kbity. Takových polí je nanejvýš $99900/900 = 111$ v jednom poli, pravděpodobnost je tedy 11%. S dalšími poli se tato pravděpodobnost násobí, tedy pro čtyři pole je pravděpodobnost špatné identifikace $1,52.10^{-4}$.

Výsledky metod Sample-and-Hold a Multi-stage filter na reálném provozu jsou uvedeny v tab. 4.2, kde nejlepším algoritmem se jeví Multi-stage filters. Nicméně pro všechny výše uvedené metody chybí porovnání, co se týče rychlosti identifikace.

Tabulka 5: Porovnání výsledků pro různé heuristiky (převzato z [24])

	Neidentifikováno / Průměrná chyba		
Flow Size	**Sample-and-Hold**	**Multistage Filters**	**Vzorkování**
$> 0{,}1\%$	0% / 0,000008%	0% / 0,000007%	0% / 4,877%
$0{,}1 \ldots 0{,}01\%$	0% / 0,0015%	0% / 0,0014%	0,002% / 15,28%
$0{,}01 \ldots 0{,}001\%$	0,000016% / 0,1647%	0% / 0,1444%	5,717% / 39,87%

5 Indexace

Většina síťových zařízení potřebuje uchovávat informace potřebné pro zpracování příchozích paketů, jako jsou informace o směrování, klasifikaci, filtraci a statistikách. Způsoby, jak tyto data organizovat a jak v nich vyhledávat se mohou lišit, podle typu dat, aplikace, platformy a dalších parametrů návrhu. Například data, která se neustále mění (přidáváním nových položek, odebíráním starých položek) vyžadují odlišnou strukturu uložení než data, u kterých tyto operace nejsou vyžadovány vůbec nebo sporadicky. Data, na jejichž základě vyhledávání probíhá, se nazývají klíč. V síťové oblasti bývá klíčem například cílová IP adresa anebo může klíč tvořit i vektor více dat, tzv. klíčových položek (např. zdrojová a cílová IP adresa, zdrojový a cílový port, číslo protokolu). Při sledování toků se vyžaduje, aby zvolený postup indexování dovoloval dynamicky přidávat a odebírat záznamy o tocích a zároveň redukoval počet přístupů do paměti.

5.1 Asociativní paměť

Asociativní paměť (TCAM – Ternary Content Addressable Memory) umožňuje přístup k datům na základě obsahu dat v klíči. Asociativní paměť je možné implementovat na několika úrovních, od ASIC implementace přes FPGA až po v programovém vybavení počítače (viz C++ STL knihovna), kde ale neexistuje nativní podpora tohoto typu adresování. Naproti tomu specializovaný TCAM čip bude mít implementováno pole komparátorů, pro každý řádek jeden komparátor o délce klíče. Porovnání klíče a dat je provedeno paralelně ve všech řádcích současně, vyhledání je tak velice rychlé. Nicméně v současné době se při vývoji síťových zařízení začíná od těchto pamětí upouštět nebo se využívají v kombinaci s dalšími technikami. Důvodem je jejich poměrně vysoká cena za bit takové paměti (až desetkrát dražší než u SDRAM) a vysoká spotřeba. Přesto TCAM byla a je používána v mnoha aplikacích, například vyhledávání řetězců [65], směrování [33] nebo filtrování provozu .

5.2 Indexování se stromy

Při tvorbě indexu se lze inspirovat postupy využívaných při směrování paketů, které vyžaduje efektivní uložení směrovací tabulky tak, aby v ní bylo možné rychle vyhledávat cílový port na základě IP adresy. V těchto případech se řeší problém vyhledání nejdelšího shodného prefixu – LPM (Longest Prefix Match), tedy pro vstupní IP adresu hledáme v množině IP prefixů takový, který bude s IP adresou sdílet nejdelší prefix.

Základním algoritmem pro vyhledávání nejdelšího prefixu je trie [28], z anglického retrieval. Jeho základem je tzv. prefixový strom. Každý uzel obsahuje data a dva ukazatele na dva následníky. Při vyhledávání postupujeme od kořene stromu, kdy se podle nejvýznammějšího bitu rozhodneme, který z ukazatelů použít. S přechodem do dalšího uzlu zahodíme i nevýznamnější bit a považujeme tento uzel za nový kořen. Takto postupujeme ve stromu dále do hloubky. Pokud není možné přejít do dalšího uzlu (neexistuje), pak jsme nalezli nejdelší uložený prefix odpovídající naší IP adrese.

Existuje řada modifikací trie, například Tree Bitmap [22] se zvýšenou aritou uzlů a bitmapou v uzlu poskytující informace o podstromech. Z dalších stromově orientovaných indexů je vhodné zmínit hardwarovou realizaci uspořádaného B-stromu v podobě ASIC [47], která poskytuje dostatek výkonu pro zpracování až 60 milionů dotazů za vteřinu.

Přestože trie poskytuje vhodnou formu indexace pro vyhledávání na základě IP adresy, v kombinaci s dalšími položkami vzniká již příliš rozsáhlá struktura vyžadující mnoho přístupů do paměti. Navíc některé vlastnosti trie nejsou při indexaci toků využity, například vyhledání podle nejdelšího prefixu.

Věnujme se proto stromovým algoritmům [30, 38], které na tvorbu indexu nahlíží obecněji jako na geometrický problém. Jejich základem je rozhodovací strom, který v každém uzlu dělí prostor podle různých dimenzí (položek) klíče na několik intervalů, dokud v uzlech nezůstane rozumné množství položek, které je možné sekvenčně prohledat. Zde může vzniknout problém, kdy se v jednom koncovém uzlu hromadí více položek než v ostatních v důsledku již známého pravidla o heavy-tail rozložení převážné většiny vlastností toků a je nutné takový strom balancovat. Navíc si povšimněme, že strom opět sdružuje podobné klíče, což je při udržování toků na rozdíl od směrování paketů naprosto nevýhodné. Proto se většinou upřednostňují jiné algoritmy a stromů se nanejvýš využívá jako pomocných struktur.

5.3 Hash tabulky s nepřímým adresováním

Hashovací tabulky jsou populární indexovací strukturou umožňující rychlé vyhledávání v datech v konstantním čase O(1). V současné době jsou nejvíce rozšířené ve své nejjednodušší formě tzv. naivní hashovací tabulky (NHT). Další tabulky se snaží zlepšit efektivitu vyhledávání za cenu průběžných změn v tabulce nebo přidáním pomocné struktury, např. Bloomova filtru [6].

Naivní hashovací tabulku si lze představit jako pole seznamů, kdy hledaná data jsou uložena v uzlech seznamu. Při vyhledávání dat se postupuje takto: vypočítání hash hodnoty klíče h(k), využití spočítané hash pro výběr seznamu, sekvenční průchod seznamu a porovnání každého prvku na shodnost s prvkem hledaným. Průměrnou délku seznamu je možné jednoduše odvodit ze zaplnění tabulky, schopné pojmout B seznamů a M prvků, tj. $\gamma = \frac{M}{B}$. Molina [44] rozebírá některé další vlastnosti NHT, mezi jinými uvádí rovnici pro výpočet pravděpodobnosti, že délka seznamu l překročí N prvků:

$$P(l > N) = \left(\frac{1}{\sum_{k=0}^{N} \frac{\gamma^{-k}}{k!}} \right) \cdot \frac{\gamma^{-N}}{N!}$$

Tato pravděpodobnost je velice důležitá v případech, kdy není možné dimenzovat délku seznamu neomezeně, například kvůli omezené velikosti hardware nebo neakceptovatelné době pro dohledání prvku.

Následnými rozšířeními konceptu NHT se zabýval Song [59]. Jeho cílem bylo snížit počet přístupů do paměti na minimum a zároveň také snížit maximální délku seznamu. K tomuto účelu využil počítající Bloomovy filtry [25], na jejichž základě vybíral seznam s nejmenším počtem položek, ve kterém se vyhledává prvek. Díky výběru více seznamů a za cenu přeuspořádání položek mezi těmito seznamy je možné dosáhnout rovnoměrnějšího rozložení délek seznamů, a tím snížit počet přístupů do paměti. Navíc Bloomův filtr poskytuje odpovědi na dotazy na členství do množiny, tedy pokud se daný prvek nevyskytuje v paměti, pak Bloomův filtr tuto skutečnost odhalí ještě před tím, než se vystaví požadavek na čtení z paměti. Nezbytným předpokladem je, že platforma obsahuje rychlou paměť, ve které je možné čítající

Bloomův filtr implementovat. Pro ilustraci jsou v tab. 6 uvedeny výsledky prezentované v [59], kde je možné porovnat přínos Bloomova filtru (sloupec basic) a přeskládávání (balancing).

Tabulka 6: Porovnání naivní a rozšířené hashovací tabulky, předpokládaný počet seznamů, jejichž délka bude $> j$, pro 8% zaplnění tabulky a 10 hash funkcí (převzato z [59])

j	**NHT**	**rozšířená**
1	740	19
2	28	$4,3.10^{-4}$
3	0,7	0

NHT lze také rozšířit o přeskládání uzlů v seznamu při prohledávání seznamu. V případě, že je uzel nalezen, pak je převázán na začátek seznamu. Tím jsou na začátku seznamu uchovány nejčastěji přistupované uzly a na konci seznamu se hromadí nejméně využívané, každý seznam je tedy seřazen podle frekvence použití uzlů. Toho lze využít ke dvěma účelům: (1) Rychlejší vyhledání uzlů, které jsou častěji využívané, (2) při nedostatku místa v tabulce nebo v seznamu je možné začít uvolňovat nejméně využívané uzly. Rozbor vlastností takto rozšířené tabulky ovšem chybí.

Doposud byly uvažovány pouze deterministické formy indexování, tedy takové, u kterých je zaručeno, že pokud se prvek v datové struktuře vyskytuje, pak bude vždy nalezen a zároveň nebude zaměněn s prvkem jiným. Pro některé aplikace může být výhodné obětovat determinismus, respektive tolerovat nedeterminismus v malém procentu případů, za efektivnější vyhledávání a uložení celé struktury. Například je možné upravit NHT tak, že místo konkrétních dat v uzlech uložit pouze ukazatel na tyto data a otisk klíče. Ten lze lépe porovnat při vyhledání dat a zabírá méně místa než celý klíč. Vzniká ale možnost, že více uzlů v seznamu bude mít shodný otisk. Pravděpodobnost, že tato situace nastane, je popsána tzv. ,,narozeninovým paradoxem" a je vyjádřena následující rovnicí:

$$p_f = 1 - \frac{m!}{m^n(m-n)!} = 1 - \frac{2^{(h+b)}!}{2^{(h+b)^n}(2^{(h+b)}-n)!}$$

$$\approx 1 - e^{-\frac{(n-1)n}{2\times 2^{(h+b)}}},$$

kde m je počet rozdílných výsledků hash funkce a n je aktuální počet vložených položek. Počet bitů výsledku hash hodnoty je roven sumě počtu bitů otisku b a počtu bitů h sloužících pro výběr seznamu. V tabulce 7 jsou shrnuty hodnoty pro často využívané konfigurace.

Tabulka 7: Pravděpodobnost kolize

Počet klíčů	**Délka hash hodnoty** $(h+b)$

Stejně tak u již zmíněných Bloomových filtrů, jenž se tradičně využívají pro dotazy náležitosti prvku do množiny, nastává situace, kdy již není možné přidat nový prvek do Bloomova filtru, neboť všechny adresované bity byly již nastaveny předchozími prvky. V takovém případě nastane falešně pozitivní odezva filtru značící, že daný prvek se v množině nachází, přestože je nový. Pravděpodobnost, že nastane taková kolize, je dána následující rovnicí:

$$p = (1 - (1 - \frac{1}{m})^{kn})^k$$

Další z významných poznatků o tvorbě indexu shrnuje [43], kde autor ukazuje sílu možnosti volby. Například pokud je zvoleno schéma se dvěma hash funkcemi a tedy možností si vybrat, na které ze dvou míst prvek uložíme, pak se výrazně zvýší kapacita takové tabulky. Naopak další zvýšení možnosti volby, tedy přidání dalších hash funkcí, již tak výrazný přínos mít nebude.

5.4 Hash tabulky s přímým adresováním

V předchozí kapitole byla popsána tvorba indexů za pomoci hash funkce a následného vyhledání položky. Použitá datová struktura tak musela mít formu dvourozměrného pole (neboli pole seznamů). Následující kapitola je věnována indexům, které vystačí s jednorozměrným polem, které adresují přímo hodnotou vypočítané hash funkce.

Základní algoritmus pro přístup do takového pole se nazývá *uniformní hash*. Princip vyhledávání je založen na iterativním prohledávání tabulky několika hash funkcemi $f_0, f_1, ..., f_i$, kde i je pořadí iterace:

1. Vypočteme hodnotu hash funkce f_i pro pro klíč k, $h_i = f(k)$.
2. Pokud adresovaná položka v tabulce $t[f_i(k)] = k$, pak vyhledání končí úspěchem, v opačném přecházíme do další iterace do bodu 1.

Pro vyhledání volné položky postupujeme obdobně, dokud nenalezneme volné místo v tabulce, tedy do doby než $t[f_i(k)] = \emptyset$. Bylo ukázáno, že takový způsob indexování je optimální [64] pro dosažení největšího zaplnění tabulky. Vyžaduje však mnoho přístupů do paměti, jak pro operaci založení nového klíče, tak pro jeho vyhledání, což je v řadě případů neakceptovatelné.

Zajímavých výsledků, především vyhledání v omezeném a konstantním čase, dosáhlo adresování založené na tzv. *kukačkové hash tabulce* [20]. Samotná tabulka je organizována jako prosté jednorozměrné pole s $(1 + \epsilon)n$ prvky, kde n je počet prvků, které si přejeme uložit, a ϵ je konstanta s nízkou hodnotou umožňující algoritmu dosáhnout lepších výsledků (v článku se udává $\epsilon = 0,03$). Hlavní trik kukačkové hash funkce spočívá ve využití dvou hash funkcí pro vyhledání či vložení klíče. Pokud klíč způsobí při vkládání kolizi na obou adresovaných místech, pak nový klíč nahradí jeden z původních, pro který se okamžitě vypočítají jeho dvě hash hodnoty a hledá se nové místo pro tento klíč stejným způsobem. Rekurzivně tak algoritmus pokračuje dokud nenajde volné místo pro aktuálně přesouvaný klíč. Pro kukačkovou hash se dvěma funkcemi platí, že zaplněnost tabulky musí zůstat pod 49%, v opačném případě přestane platit horní omezení $\Omega(ln(\frac{1}{\epsilon}))$ pro délku vložení nového klíče. Algoritmus byl rozšířen o využití na d-ární kukačkovou hash, pro kterou je možné dosáhnout téměř plné zaplnění tabulky. Analýza a experimenty prezentované v [27] uvádějí dosažení zaplněnosti 97% při využití čtyřech hash funkcí. Pro vložení nového prvku se vyžaduje průměrně 20 přístupů do tabulky při zaplněnosti 90%, což je jen dvakrát více než při uniformním hashování.

Pokud je množina klíčů předem známa, lze nalézt bezkolizní hashovací funkci, tzv. *perfektní hash*. Tento způsob hashování je možné použít u aplikací, kde množina klíčů je předem známa a přidávání klíčů, které nebyly v době generování hash funkce známé, je velmi řídké a dává dostatek času vypočítat novou hash funkci. Pro zajímavost je uveden jeden ze způsobů, jak takovou funkci vypočítat [17]. Principem je sestrojení neorientovaného grafu, kde každý uzel odpovídá výsledkům dvou různých hashovacích funkcí a každá hrana odpovídá závislosti těchto uzlů a zároveň výsledku perfektního hashování.

1. Mějme množinu klíčů K, $\forall k \in K$ existuje unikátní číslo $h(k)$, které je výsledkem hledané perfektní hashovací funkce.
2. Vytvoříme graf $G = (V, H)$ s $\mid V \mid = c\dot{\mid}K \mid$ vrcholy, kde $c > 1$.
3. Zvolme dvě různé hashovací funkce f_1 a f_2 s obory hodnot $< 0, N)$.
4. $\forall k \in K$ vytvoříme hranu $(f_1(k), f_2(k))$ s ohodnocením $o(f_1(k), f_2(k)) = h(k)$.
5. Pokud graf G obsahuje cyklus, pak zvětšíme c a vrátíme se na krok 2.
6. Provedeme ohodnocení vrcholů tak, aby platilo, že pokud $(v_1, v_2) \in H \Rightarrow (o(v_1) + o(v_2)) = o(v_1, v_2)$.

Pro samotný výpočet hodnoty perfektní hash funkce klíče vystačíme pouze s uzly $h(k) = o(f_1(k)) + o(f_2(k))$. Jak již bylo zmíněno v neýhodách PHF, pokud přijde klíč, který nebyl zahrnut do konstrukce PHF, pak dojde ke kolizi. Tuto kolizi je nutné detekovat, například porovnáním klíče požadavku a klíče adresovaných dat a vybudovat PHF znovu.

U všech index algoritmů se předpokládá uniformní rozložení výsledků hash funkcí, tu poskytují například [11]. Pro kombinaci síťového provozu a obvodovou realizaci a se ukázala jako vhodná třída funkcí založená na dělení dat polynomem [53]. Jedním z příkladů takové hash funkce je CRC (Cyclic Redundancy Check) s navrženou možností paralelizace v hardware [7].

6 Správa paměti

6.1 Jednoduché správy paměti

Random neboli náhodná správa paměti, kdy při plné cache dojde k uvolnění náhodně vybrané položky. Lze ji vnímat jako aproximaci správy paměti FIFO (popsána níže). Čím déle je položka v paměti, tím větší je pravděpodobnost, že bude uvolněna. Výhodou je, že tato správa paměti nezabírá další paměťové místo. Nicméně v některých systémech není k dispozici kvalitní generátor náhodných (pseudonáhodných) čísel, jehož perioda je shodná s velikostí spravované cache. V opačném případě by nedocházelo k uvolnění, již nepoužívaných položek.

First In First Out (FIFO) je jedním z nejjednodušších přístupů pro správu paměti cache, kdy položky v paměti cache jsou uloženy ve frontě. Nové položky položky jsou vkládány na začátek fronty a při nedostatku místa v paměti jsou položky odebírány z konce fronty. Jednoduchost a zároveň naivita spočívá v tom, že samotné využití položek nemá žádný vliv na pozici položek ve frontě. Položka je do fronty vložena při svém prvním použití a následně je odsouvána dalšími nově vkládanými položkami. Doba, po kterou vydrží položka v cache je dána pouze počtem nově příchozích položek. Pokud od vložení položky přišlo tolik položek, kolik je kapacita cache, pak je položka z cache odsunuta. Dá se říci, že všechny položky dostanou možnost využít cache rovným dílem. Jediným ukazatelem pro odsun položek je čas prvního využití položky. Intuitivně bude tato strategie dobře pracovat s daty, která vykazují určitý typ využití položek v paměti. Konkrétně, ihned po prvním využití položky následuje posloupnost dalších využití po omezený interval, který je dán velikostí pamětí cache. Po skončení tohoto intervalu se položka již více nepoužije. Takový typ využití položek je ovšem velmi ojedinělý, naopak většina datových sad na Internetu vykazuje odlišné chování, kdy délky *života* položek a množství přístupů do paměti se značně liší.

Nicméně strategii FIFO lze implementovat jednosměrně vázaným seznamem, kdy nové položky jsou vkládány na konec seznamu a při nedostatku místa v paměti jsou položky odebírány ze začátku seznamu.

6.2 Správa paměti využívající nedávnou historii přístupů

Least Recently Used (LRU) je široce rozšířená technika správy paměti cache využívána v různých oblastech, např. pro správu překladu stránek paměti, databáze nebo diskové vyrovnávací paměti. Oblíbenost LRU plyne z efektivity při správě paměti na běžných datových sadách a z jednoduché implementace. LRU pracuje následujícím způsobem. Když je paměť plná, tato strategie odsouvá z paměti položku, která byla nevyužita nejdelší dobu. Konkrétní implementace může být například realizována obousměrně vázaným seznamem pamětových položek. Nové položky a rovněž i používané (přistupované) položky jsou přesouvány na začátek tohoto seznamu, zatímco nepoužívané položky zůstávají na konci tohoto seznamu a poslední s nich může být odsunuta z paměti. Tato strategie výměny položek v paměti cache předpokládá, že paměť bude pracovat s daty, která mají velmi dobrou časovou lokalitu, neboť nové a nedávno použité odsouvají již dříve použité položky. V případě špatné lokality, například výskytu postupného náčítání nových položek bude LRU neúčinná. Z tohoto důvodu vznikla vylepšení LRU.

Segmented LRU rozděluje seznam položek spravovaných pomocí LRU na dvě části, tzv. chráněnou a nechráněnou část. Bod rozdělení seznamu je určen místem, kam jsou vkládány nové položky. Chráněná část seznamu vede od začátku seznamu až po místo vkládání nových položek a od tohoto místa do konce seznamu se nachází nechráněná část. Určení vhodného místa vkládání nových položek, tedy velikost chráněné části je závislé na různých faktorech jako jsou datová sada, velikost cache a další. Správné nastavení je předmětem experimentů a konkrétního nasazení. Přesun místa vložení ze začátku do vnitřní části seznamu umožňuje z cache rychleji odstraňovat položky, které nemají žádnou lokalitu a jejich přítomnost v cache je nežádoucí, neboť zabírají místo pro položky s dobrou lokalitou. Předpokládá se,

že položky s dobrou lokalitou se dostanou do chráněné oblasti, kdežto ty se špatnou lokalitou zůstanou v nechráněné. O místo v chráněné oblasti tak budou soupeřit pouze položky s dobrou lokalitou.

LRU-2 (Least Recently Used Twice) je strategie, která využívá pro rozhodnutí o uvolnění položky z cache poslední dva přístupy k položce, na rozdíl od LRU, která uvažuje pouze poslední přístup k položce. Obecně lze rozšířit LRU-2 až na LRU-K [49] techniku, tedy uchovávání a rozhodování se podle posledních K přístupů k položce. Strategie LRU-K pracuje následovně. Při příchodu nové položky a plné cache se nahradí položka s nejdelším intervalem mezi K-tým přístupem do paměti a aktuálním časem. Položky, které nebyly alespoň K-krát využity mají interval nastaveny na nekonečno. Pokud tedy existuje více položek s menším počtem než K přístupů, pak se rozhoduje o uvolnění mezi těmito položkami. Sníží se K (K ¡- K-1) a postupujeme podle předchozího scénáře až na K=1, kdy aplikována již známá strategie LRU neboli LRU-1. K vyhledání položky s nejdelším intervalem navrhují autoři LRU-K využít vyhledávací strom. Na základě experimentů nad datovými sadami ukazují, že LRU-2 dosahuje dostatečně dobrých výsledků a implementace LRU-K, kde $K > 2$ přináší pouze malé zlepšení v porovnaní s nárůstem režie spojeného s udržováním dlouhodobé historie.

HLRU vyvinuto nezávisle na LRU-K, ale naprosto shodná metoda aplikována jako strategie pro uvolňování položek z Web cache.

Two Queues (2Q) [36]. Algoritmus LRU-2 má logaritmickou časovou složitost, neboť položky musí být indexovány pomocí vyhledávácího stromu. Správa paměti cache 2Q implementuje filozofii správy LRU-2 ale s konstantní časovou složitostí. K tomu využívá v nejjednodušší verzi dvě fronty A1 a Am. Fronta Am je spravována jako LRU seznam a uchovává často využívané položky. Fronta A1 je spravována jako FIFO a slouží pro identifikaci položek, které mohou být potenciálně často využity. Takto identifikované položky jsou přesunuty do Am. Tím tato technika připomíná Segmented LRU s tím rozdílem, že A1 je spravována pomocí FIFO strategie a dále, pokud je překročena velikost Am pak nejsou přebývající položky přesunuty do A1, nýbrž jsou rovnou uvolněny.

LIRS [35] Správa paměti LIRS je založena na udržování položek s malou vzdáleností mezi po sobě následujícími přístupy, ktere jsou udržovány v tzv. Low-level interreference recency sadě

Exponential Smoothing EXP1 [54]. Podobně jako LRU-K uvažuje posledních K přístupů k položce. Na rozdíl od LRU-K přikládá vyšší váhu nedávným přístupům. Konkrétně váha jednotlivých přístupů klesá exponenciálně se zvyšujícím se pořadím přístupů k položce. Tedy nejvíce vzdálený přístup má nejnižší váhu a aktuální přístup nejvyšší. Podstatou strategie je odhad frekvence budoucího využití položky na základě historie přístupů, tedy $W(t_i) = \frac{1}{\mu_i}$, kde W je odhad frekvence, t_i je interval od i-tého přístupu k položce a μ_i je odhad průměru intervalů mezi přístupy. Tento μ_i je počítán klouzavým průměrem s exponeciálním oknem jako $\mu_i = \alpha t_i + (1 - \alpha)\mu_{i-1}$ a $\mu 0 = \frac{1}{t_0}$, kde α je konstanta ovlivňující délku využívané historie, např. pokud $\alpha = 1$, pak se z EXP1 strategie stává LRU, neboť využívá k odhadu frekvence pouze dobu od posledního přístupu k položce. Autoři ovšem navrhují nastavit $\alpha = 0.1$, kdy dochází ke kombinaci nedávnosti a frekvenci přístupů. Konkrétní implementace této strategie autoři neuvádí, ale můžeme předpokládat, že se až na výpočet frekvence neliší od LRU-K, neboť položky je nutné řadit a vyhledávat ve vyhledávacím stromě na základě aktualní hodnoty μ_i.

6.3 Správa paměti využívající četnost přístupů

V prostředí, kde množina často přistupovaných položek zůstává stále stejná nebo se mění jen velmi pomalu, je možné spravovat položky dle jejich popularity. Předpokládá se, že aktuálně populární položky budou populární i v budoucnu. Dalším předpokladem je, že frekvence přístupů k populárním položkám bude vyšší než u nepopulárních. Úkolem správy paměti je v takových případech sledovat počet přístupů k položkám a dobu života položky a na základě těchto informací rozhodovat o tom, které položky se uvolní.

Least Frequently Used (LFU) se řadí mezi základní a zároveň nejznámější správu paměti, založené na sledování četnosti přístupů k položkám. Pokud je cache plná a je potřeba zajistit místo pro novou položku, pak LFU odstraní položku, která byla nejméně často používána. Tedy položku s nejmenším počtem přístupů za časový interval jejího života. Implementace této správy tedy vyžaduje spočítat frekvence přístupů ke všem položkám a řadit položky dle těchoto frekvencí. Následně je jednoduché identifikovat položku s nejnižší frekvencí přístupů a tuto položku uvolnit. Nedostatkem tohoto přístupu je zvýhodnění položek, které mají málo po sobě jdoucích přístupů v krátkém časovém úseku. Například, mějme dvě

položky, první s velkým množstvím přístupů a dlouhou dobou života, a druhou, která má pouze dva přístupy hned za sebou. Frekvence přístupů u obou položek je shodná, nicméně o popularitě druhé položky lze zatím pochybovat.

$LFU*$ [4] je lehkou modifikací LFU. LFU* povoluje, na rozdíl od LFU, odstranit z cache pouze položky pouze s jedním přístupem. Pokud v cache není místo a nejsou dostupné položky s jedním přístupem, pak se nová položka do cache nevloží. Arlitt a Wiliamson ve své práci [4] uvádějí, že tato správa paměti má na vzorku dat z webového serveru horší výsledky než samotná LFU, nicméně je základem pro další modifikaci LFU, konkrétně LFU*-Aging(popsána níže), která již dosahuje výrazně lepších výsledků než samotná LFU. Důvodem horších výsledků je, že jakmile se cache zaplní položkami s četností vyšší než jedna není možné do cache další položky přidávat. U většiny datových sad, ale dochází k obměně populárních položek s časem a tím pádem položky v cache nebudou

$LFU\text{-}Aging$ neboli LFU se stárnutím položek. LFU-Aging se snaží odstranit neschopnost LFU rozpoznat situaci, kdy položka získala vysokou míru popularity v minulosti, nicméně v současné době již není používána. Tyto položky pak zůstávají v cache zbytečně dlouho a zabírají tak místo novým položkám. LFU-Aging odstraňuje tento jev zavedením mezní frekvence, které může položka dosáhnout a tzv. stárnutí. Ke stárnutí dochází následovně. Pokud frekvence položky přesáhne mezní frekvenci, pak je čítač počtu přístupů snížen na polovinu. Mezní frekvence by měla být nastavena na základě analýzy vlastností prostředí, ve kterém cache bude použita.

$LFU*\text{-}Aging$ [61] kombinuje LFU* a LFU-Aging. Dle Arlitta dosahuje tato kombinace výrazně lepších výsledků oproti běžnému LFU, i když samostné LFU* a LFU-Aging dosahují horších výsledků.

$Window\text{-}LFU$ [32] je modifikací LFU, kdy měření frekvence přístupů k položkám v cache je limitováno na určitý počet přístupů do cache. To znamená, že frekvence přístupů k položce je počítána pouze z posledních několika přístupů, na rozdíl od LFU, kdy se počítá frekvence přístupů od vzniku položky.

$LFU\text{-}DA$ [21] neboli LFU with Dynamic Aging (LFU s dynamickým stárnutím). LFU-DA se snaží odstranit problém s nastavením parametrů u LFU-Aging. Nastavení parametrů je závislé na datové sadě, na kterou cache pracuje a je nutná analýza této datové sady pro správné nastavení parametru stárnutí a mezní frekvence. V mnoha případech se ovšem vlastnosti datové sady mění a není možné dopředu předpočítat zmíněné parametry. LFU-DA proto zavádí tzv. inflation factor nebo také faktor běžícího stáří L. Faktor L má na začátku hodnotu nula, a v průběhu uvolňování položek je aktualizován na $L = K_i$, kde K_i je prioritní klíč uvolněné položky i, který se spočítá jako:

$$K_i = F_i + L,$$

kde F_i je frekvence položky i. Rozhodnutí, které položky uvolnit je prováděno na základě prioritního klíče K_i a nikoliv pouze na základě frekvence F_i.

$Value\text{-}aging$ [66] odhaduje frekvenci přístupů k položce od jejího vzniku akumulací intervalů mezi přístupy, kdy poslední interval má největší váhu. Pro výpočet nové hodnoty V, dle které se rozhoduje o uvolnění dané položky, využívá následující rovnice:

$$V_{new} = V_{old} + C_t \sqrt{\frac{C_t + L_t}{2}},$$

kde C_t je aktuální čas a L_t je čas posledního přístupu k položce. Položka s nejnižší hodnotou V je uvolněna z paměti.

$Alpha\text{-}aging$ zavádí funkci $f(V)$, která má za úkol periodicky snižovat hodnotu V u všech položek. V tomto případě se nová hodnota V_{new} vypočítá při přístupu k položce jako

$$V_{new} = V_{old} + C_t$$

kde C_t je opět aktuální čas, a funkce $f(V)$ je definována jako:

$$f(V) = \alpha V; 0 \leq \alpha \geq 1$$

a je aplikována na hodnotu V v periodicky s definovaným intervalem. Zde si myslím, že rovnice je špatně v literatuře uvedena, neboť koeficient $\alpha \geq 1$ nedává smysl. V takovém případě by totiž položky zvyšovaly svou hodnotu V, i když nejsou například používány a zůstávaly by tak v cache. Koeficient α by tedy měl být omezena na $0 \leq \alpha \leq 1$

6.4 Adaptivní správa paměti

Nedávno bylo navrženo několik algoritmů správy paměti, které sledují chování bloků několika položek, tedy přístup k těmto položkám, co se týče časové a prostorové lokality. Na základě tohoto chování se snaží nové přístupy určit, který algoritmus použijí pro správu daného bloku, aby postihli specifické nároky různých programů a dosáhli tak lepšího využití cache.

SEQ [29]. Glass a Cao navrhli adaptivní výměnu stránek pro správu virtuální paměti. Navržený algoritmus se nazývá SEQ a detekuje vzory v sekvencích adresových referencí. Pokud jsou nalezeny dlouhé sekvence adres způsobující výpadký stránek, pak je aplikována správa paměti MRU. Ve všech ostatních případech je aplikována správa paměti LRU. Autoři tedy předpokládají, že LRU je dostatečně dobrá správa paměti, která selhává pouze v případech, kdy přístupy ke stránkám v paměti vykazují specifické vzory přístupů, např. sekvence přístupů k položkám se zvyšující se adresou. SEQ je založen právě na detekci takových sekvencí. To snižuje jeho univerzálnost, neboť existují i další vzory chování, u nichž může LRU selhat.

EELRU [57]. Autoři EELRU se snáží zobecnit detekci specifických sekvencí přístupů k položkám v paměti. Změna správy paměti je řízena na základě detekce mnoha odsunutých stránek, které byly nedávno vloženy do paměti cache. Implementace této myšlenky spočívá ve sběru informace o distribuci výpadků stránek a následném propočtu, v jakém bodu se vyplatí využít jinou správu paměti oproti LRU. Část položek je pak spravována LRU a část pomocí alternativní správy paměti. Nevýhodou EELRU je pomalá reakce na změnu přístupů ke stránkám. To je způsobeno tím, že velikost části paměti, která je spravována alternativní správou, je určena na základě celkové distribuce přístupů k položkám, což způsobí skrytí rychlých změn.

DEAR [12]. DEAR je rozšířením správy paměti SEQ. DEAR algoritmus je založený na klasifikaci řetězců přístupů do paměti. Pro každou běžící aplikaci DEAR určuje, zda se jedná o sekvenční, kruhový, shlukový nebo náhodný přístup. Na základě klasifikace DEAR použije pro každou aplikaci příslušnou správu paměti. Pro sekvenční a kruhový vzor chování aplikuje DEAR správu MRU, pro shlukový vzor LRU a pro náhodný vzor LFU.

UBM [37]. UBM je pokračovatelem DEAR. Oproti DEAR navíc automaticky určuje velikost částí cache přiřazené jednotlivým vzorům chování. Toto přidělování je řízeno očekávaným ziskem při alokaci dalšího místa dané správě paměti.

7 Závěr

Tato práce se zaměřila objasnění vlastností síťového provozu a následně na algoritmy, které jsou potřebné pro sledováním síťového provozu na úrovni toků. Studiem a zdokonalováním těchto algoritmů může být dosažena významná úspora paměťových a výpočetních prostředků v mnoha systémech nasazených na síti.

Reference

[1] Cisco web pages, http://www.cisco.com, 2006.

[2] Big growth for the internet ahead, cisco says, dostupné online: http://gigaom.com/2008/06/16/big-growth-for-internet-to-continue-cisco-predicts/, 2008.

[3] Internet growth statistics, http://www.internetworldstats.com/emarketing.htm, 2009.

[4] Martin F. Arlitt and Carey L. Williamson. Trace-driven simulation of document caching strategies for internet web servers. *Simulation Journal*, 68:23–33, 1996.

[5] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Pop-level and access-link-level traffic dynamics in a tier-1. In *In ACM Sigcomm Internet Measurement Workshop*, pages 39–54, 2001.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[7] M. Braun, J. Friedrich, T. Grun, and J. Lembert. Parallel crc computation in fpgas. In *FPL '96: Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 156–165, London, UK, 1996. Springer-Verlag.

[8] N. Brownlee. Understanding internet traffic streams: Dragonflies and tortoises. *IEEE Communications Magazine*, 40:110–117, 2002.

[9] R. Caceres. Measurements of wide-area internet traffic. Technical report, University of California, Berkeley, 1989.

[10] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area tcp/ip conversations. In *Proceedings of ACM SIGCOMM '91*, pages 101–112, 1991.

[11] J. L. Carter and M. N. Wegman. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM.

[12] Jongmoo Choi, Sam H. Noh, Sang Lyul Min, and Yookun Cho. An implementation study of a detection-based adaptive block replacement scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 1999. USENIX Association.

[13] K. C. Claffy, H. Braun, and G. C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications*, 13:1481–1494, 1995.

[14] B. Claise. Cisco systems netflow services export version 9. RFC(Informational) 3954, Internet Engineering Task Force, 2004.

[15] B. Claise. Ipfix protocol specification. Internet draft: draft-ietf-ipfix-protocol-21.txt, work in progress, Internet Engineering Task Force, April 2006.

[16] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.

[17] Z. J. Czech, G. Havas, and B. S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Inf. Process. Lett.*, 43(5):257–264, 1992.

[18] L. Degioanni and G. Varenni. Introducing scalability in network measurement: toward 10 gbps with commodity hardware. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 233–238, New York, NY, USA, 2004. ACM.

[19] L. Deri, S. P. A. Netikos, V. D. Brennero, and L. L. Figuretta. Improving passive packet capture:beyond device polling. In *Proceedings of SANE 2004*, 2004.

[20] L. Devroye and P. Morin. Cuckoo hashing: further analysis. *Inf. Process. Lett.*, 86(4):215–219, 2003.

[21] John Dilley, Martin Arlitt, and Stephan Perret. Enhancement and validation of squid's cache replacement policy. Technical report, HP.

[22] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, 2004.

[23] 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.

[24] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[25] L. Fan, J. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[26] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Global Telecommunications Conference*, Dec 1999.

[27] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. In *STACS '03: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, pages 271–282, London, UK, 2003. Springer-Verlag.

[28] E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960.

[29] Gideon Glass and Pei Cao. Adaptive page replacement based on memory reference behavior. In *Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '97, pages 115–126, New York, NY, USA, 1997. ACM.

[30] P. Gupta. *Algorithms for routing lookups and packet classification*. PhD thesis, Stanford University, Stanford, CA, USA, 2000. Adviser-Nicholas W. Mckeown.

[31] Q. He, C. Dovrolis, and M. Ammar. On the predictability of large transfer tcp throughput. *Comput. Netw.*, 51(14):3959–3977, 2007.

[32] Wen-Chi Hou and Suli Wang. Size-adjusted sliding window lfu - a new web caching scheme. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications*, DEXA '01, pages 567–576, London, UK, 2001. Springer-Verlag.

[33] D. Antoš J. Novotný, O. Fučík. Project of ipv6 router with fpga hardware accelerator. In *Book Series Lecture Notes in Computer Science*, volume 2778/2003, pages 964–967. Springer Berlin / Heidelberg, 2003.

[34] R. Jain and S. A. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4:986–995, 1986.

[35] Song Jiang and Xiaodong Zhang. Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *SIGMETRICS Perform. Eval. Rev.*, 30:31–42, June 2002.

[36] Theodore Johnson and Dennis Shasha. 2q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 439–450, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[37] Jong Min Kim, Jongmoo Choi, Jesung Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 9–9, Berkeley, CA, USA, 2000. USENIX Association.

[38] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, 28(4):203–214, 1998.

[39] K. Lan and J. Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46 – 62, 2006.

[40] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.

[41] J. Li, C. Hu, and B. Liu. Monitoring large flows in network.

[42] P. Loiseau, P. Goncalves, P. Primet, P. Borgnat, P. Abry, and G. Dewaele. Investigating self-similarity and heavy tailed distributions on a large scale experimental facility. Research Report RR-6472, INRIA, 2008.

[43] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[44] M. Molina, A. Chiosi, S. D'Antonio, and G.Ventre. Design principles and algorithms for effective high speed ip flow monitoring. Technical report, 2004.

[45] J. Murai, H. Kusumoto, S. Yamaguchi, and A. Kato. Construction of internet for japanese academic communities. In *Supercomputing '89: Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 737–746, New York, NY, USA, 1989. ACM.

[46] R. Nossenson and H. Attiya. The distribution of file transmission duration in the web: Research articles. *Int. J. Commun. Syst.*, 17(5):407–419, 2004.

[47] M. O'Connor and C. A. Gomez. The iflow address processor. *IEEE Micro*, 21(2):16–23, 2001.

[48] A. Odlyzko. Internet growth: Myth and reality, use and abuse. *Journal of Computer Resource Management*, 102:23–27, 2001.

[49] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The lru-k page replacement algorithm for database disk buffering. *SIGMOD Rec.*, 22:297–306, June 1993.

[50] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughpu: A simple model and its empirical validation. Technical report, Amherst, MA, USA, 1998.

[51] K. Park, G. T. Kim, and M. E. Crovella. The protocol stack and its modulation effect on self-similar traffic. In Kihong Park and Walter Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*. Wiley / Wiley Interscience, New York, 1999.

[52] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.

[53] M. V. Ramakrishna, E. Fu, and E. Bahcekapili. A performance study of hashing functions for hardware applications. In *Proc. of Int. Conf. on Computing and Information*, pages 1621–1636, 1994.

[54] Mike Reddy and Graham P. Fletcher. Intelligent web caching using document life histories: A comparison with existing cache management techniques. In *In 3rd International WWW Caching Workshop*, pages 35–50, 1998.

[55] K. C. Claffy S. McCreary. Trends in wide area ip traffic patterns a view from ames internet exchange, 2000.

[56] S. Sarvotham, R. Riedi, and R. Baraniuk. Connection-level analysis and modeling of network traffic. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 99–103, New York, NY, USA, 2001. ACM.

[57] Yannis Smaragdakis, Scott Kaplan, and Paul Wilson. Eelru: Simple and effective adaptive page replacement. Technical report, University of Texas at Austin, Austin, TX, USA, 1998.

[58] R. D. Smith. The Dynamics of Internet Traffic: Self-Similarity, Self-Organization, and Complex Phenomena. *ArXiv e-prints*, July 2008.

[59] H. Song, S. Dharmapurikar, J. Turner, and J. W. Lockwood. Fast hash table lookup using extended bloom filter: an aid to network processing. *SIGCOMM Comput. Commun. Rev.*, 35(4):181–192, 2005.

[60] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.

[61] U. Vallamsetty, P. Mohapatra, R. Iyer, and K. Kant. Improving cache performance of network intensive workloads. In *Proceedings of the International Conference on Parallel Processing*, pages 87–, Washington, DC, USA, 2001. IEEE Computer Society.

[62] M. Žádník, T. Pečenka, and J. Kořenek. Netflow probe intended for high-speed networks. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL05)*, pages 695–698. IEEE Computer Society, 2005.

[63] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. Netw.*, 5(1):71–86, 1997.

[64] A. C. Yao. Uniform hashing is optimal. *J. ACM*, 32(3):687–693, 1985.

[65] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit rate packet pattern-matching using tcam. In *ICNP '04: Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 174–183, Washington, DC, USA, 2004. IEEE Computer Society.

[66] Junbiao Zhang, Rauf Izmailov, Daniel Reininger, Maximilian Ott, and Nec U. S. A. Web caching framework: Analytical models and beyond. In *Proceedings of the 1999 IEEE Workshop on Internet Applications*, pages 132–, Washington, DC, USA, 1999. IEEE Computer Society.

[67] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 309–322, New York, NY, USA, 2002. ACM.