Vysokorychlostní vyhledávání regulárních výrazů

Jan Kaštil
Výpočetní Technika a Informatika, 3. ročník, prezenční studium
Supervisor: Zdeněk Kotásek

Faculty of Information Technology, Brno University of Technology
Božetěchova 2

ikastil@fit.vutbr.cz

Abstract. Práce popisuje architekturu jednotky pro rychlé vyhledávání regulárních výrazů v moderních počítačových sítích. Navržená jednotka je schopná pracovat na propustnostech do 10Gbps, v závislosti na množství dostupné paměti. Představená implementace využívá perfektní hashování pro efektivní využití dostupné paměti. Jednotka podporuje možnost zmenšení potřebné paměti zavedením malého množství chyb. Datovou strukturu jednotky je možno umístit přímo do paměti na čipu nebo do externí paměti. V případě využití externí paměti jednotka maskuje latenci paměti paralelním zpracováním většího množství síťových toků. Přepnutí mezi toky nezanáší do vyhledávání žádné zpoždění. Experimenty s hardwarovou jednotkou ukázaly, že využití logických zdrojů čipu a frekvence jednotky nejsou závislé na množství a struktuře vyhledávaných regulárních výrazů.

Keywords. Regulární výraz, Konečný automat, IDS, FPGA, Perfektní hashování, vysokorychlostní sítě

1 Úvod a současný stav výzkumu

Vyhledávaní regulárních výrazů je základní operací systémů ochrany počítačových sítí (IDS), jako jsou Snort [1] nebo Bro [2]. S rozvojem počítačových sítí vzrůstají také požadavky na tyto systémy. Vzhledem k masivnímu rozšíření internetu nelze při zajišťování bezpečnosti spoléhat na znalosti a schopnosti jednotlivých uživatelů. Proto se IDS systémy musí nasazovat již na páteřních linkách, jejichž propustnost dosahuje až desítek Gbps.

Softwarové implementace nemají dostatečný výkon, aby dokázali zpracovat veškerý provoz těchto rychlých páteřních linek a často zahazují pakety, které nedokáží zpracovat [3]. Tyto pakety pak vstupují do chráněné sítě bez sebemenší kontroly. [4] ukazuje, že vhodnou volbou zahazovaných paketů lze významně zvýšit bezpečnost sítě. Útočník obeznámený se strukturou sítě však může snadno sestavit takový provoz, který softwarové IDS zahltí a donutí je přeposílat do sítě i pakety obsahující útok [5].

Pro zajištění vysoké spolehlivosti a propustnosti se používá hardwarová akcelerace regulárních výrazů. Nejjednodušším přístupem je implementace hardwarového filtru [6]. Hardware pracující s vysokou propustností vyhledává pouze části regulárních výrazů a pokud není úspěšný, je jisté, že paket nemůže hledaný výraz obsahovat a není nutné jej přeposílat pomalému softwarovému systému.

Nevýhodou hardwarového filtru je náchylnost k DoS útokům, neboť útočník může sestavit provoz tak, aby obsahoval části hledané filtrem, ale ne celé regulární výrazy a tak opět celý systém zahltit a donutit propouštět do sítě neověřený provoz. Řešením je plnohodnotná hardwarová akcelerace vyhledávání regulárních výrazů.

Jedním z používaných způsobů je implementace nedeterministického konečného automatu (NFA) pro dané výrazy do FPGA [7][8]. Jednotlivé stavy konečného automatu se mapují na registry FPGA a každý přechod je pak representován vlastní logickou funkcí, což umožňuje zpracovávat jeden znak každý hodinový cyklus. Nevýhodou implementací založených na NFA je nutnost opětovné syntézy a následné rekonfigurace FPGA při každé změně pravidel. Změna pravidel tedy může trvat i několik hodin, což není v prostředí IDS akceptovatelné.

Metody založené na deterministických konečných automatech (DFA) mají přechodovou tabulku umístěnou v paměti, kterou je možné snadno přepsat daty odpovídajícími novým pravidlům. Základním problémem DFA je však jejich exponenciální velikost oproti NFA. Metody založené na DFA musí proto efektivně využívat dostupnou paměť. Některé metody [9][10] snižují velikost přechodové tabulky zavedením dodatečné paměti pro zápis a čtení, další se snaží využít podobnosti mezi jednotlivými přechody a uchovávat pouze část přechodové tabulky [11][12].

Metoda popsaná v této práce patří do skupiny metod založených na deterministickém konečném automatu. Zbývající část práce je členěna do kapitol následujícím způsobem. Kapitola 2 popisuje podmínky kladené na vyhledávací jednotku a tím i definuje cíle práce. Třetí kapitola popisuje návrh vyhledávací jednotky a čtvrtá kapitola představuje experimentální výsledky získané při implementaci jednotky nebo při její simulaci. Pátá kapitola stručně popisuje možnosti praktického využití výsledků práce. Závěrečná šestá kapitola shrnuje přínosy práce a nastiňuje možnosti jejího dalšího rozšiřování.

2 Definice Problému

V předchozí kapitole jsem ukázali nutnost hardwarové akcelerace hledání regulárních výrazů a stručně zmínili základní přístupy používané v praxi. Tato práce se zabývá návrhem nové vyhledávací metody vhodné pro implementaci pomocí FPGA nebo ASIC. Na základě intenzivního studia problematiky jsem stanovil čtyři základní podmínky, které musí vyhledávací jednotka splňovat, aby byla prakticky použitelná:

- **Podpora Regulárních výrazů** – Jazyky používané pro popis vyhledávaných vzorů jsou často silnější než regulární jazyky. Jejich nevýhodou je velká časová složitost. Velká většina prakticky používaných vzorů proto tyto složité konstrukce nepoužívá a dají se popsat regulárními výrazy.

- **Podpora 10Gbps sítí** – Páteřní linky velkých poskytovatelů internetu již běžně dosahují multigigabitových rychlostí. Pro detekci útoků na síti je nutné aby vyhledávací jednotka zvládala zpracovat veškerý provoz v reálném čase.

- **Přepínání kontextu** – Provoz na internetu je rozdělen do jednotlivých paketů. Související pakety tvoří síťový tok. Původní implementace IDS prováděly vyhledávání vzorů v každém paketu toku samostatně, což umožňuje útočníkovy, aby útok rozdělil do dvou po sobě jdoucích paketů a tak jej před IDS ukryl. Kompletní rekonstrukce TCP toku ale není v silách zařízení na páteřní síti. Proto se používá kompromis, kdy se pakety zpracovávají postupně a mezi jednotlivými pakety náležejícími stejnému toku se uchovává stav jednotky. Pro podporu tohoto řešení je však nutné, aby vyhledávací jednotka umožňovala okamžité přepnutí kontextu vyhledávání a paket nebyl zdržován čekáním na uvedení jednotky do správného stavu.

- **Rychlá změna vyhledávacích pravidel** – Útočníci vytvářejí stále nové a nové metody útoků. Samotné útoky jsou často automatizovány a probíhají pomocí robotů či červů. Červ Witty [13] dokázal napadnout většinu zranitelných počítačů do hodiny po začátku své aktivity.

3 Návrh řešení

Navrhované řešení je založeno na deterministickém konečném automatu, který umožňuje zpracovávat vstupní symbol každý krok výpočtu. Veškerá data potřebná pro vyhledávání jsou umístěna v paměti a lze je tedy měnit bez nutnosti rekonfigurace akcelerační jednotky.

V článku [14] jsem provedl analýzu pravidel používaných IDS systémy s pohledu deterministických konečných automatů. Ukázalo se, že pro velkou část výrazů lze vytvořit konečný automat s velmi řídce zaplněnou přechodovou tabulkou.

Deterministický konečný automat vyžaduje pro každý symbol vyhledání v přechodové tabulce. Aby byla zachována vysoká propustnost navrženého řešení je nutné realizovat vyhledání v konstantním čase. Na základě znalosti o nízkém zaplnění přechodové tabulky jsem se rozhodl implementovat automatu pomocí hashovací tabulky.

Hashovací tabulka má v průměru konstantní časovou složitost vyhledávání, ale složitost nejhoršího případu je lineární s počtem položek v tabulce. Použitím Perfektního hashování však lze zajistit konstantní časovou složitost i v nejhorším možném případě. Perfektní hashovací funkce (PHF) je taková hashovací funkce, která nemá pro danou množinu klíčů žádné kolize. Vzhledem k vysoké pravděpodobnosti kolizí je PHF poměrně málo a jejich hledání není triviální proces. Ve své práci využívám algoritmus [15], který je schopen nalézt PHF s lineární časovou složitostí v závislosti na počtu klíčů. PHF se skládá z několika universálních hashovacích funkcí odkazujících do paměti. Z údajů uložených do paměti při přípravě PHF se pak spočítá, která z hashovacích funkcí vrací pro daná klíč jedinečný výsledek. Velkou výhodou tohoto algoritmu je možnost rozšířit tabulku o další informace. V rámci jednoho přístupu do paměti tak lze získat nejen informace potřebné pro výpočet PHF, ale také záznam z hashovací tabulky.

Architekturu využívající perfektního hashování pro implementaci konečného automatu jsem publikoval na konferenci DSD [16] spolu s popisem postupu transformace množiny regulárních výrazů do podoby vhodné pro vyhledávací jednotku. Při použití hashovací tabulky pro ukládání přechodů je třeba uložit nejen následující stav, ale také klíč hashovací funkce, aby se dalo po vyhledávání rozpoznat, zda nedošlo ke kolizi. PHF neobsahují kolize mezi klíči, ale v případě, že se na vstupu objeví neznámá kombinace, tedy neexistující přechod, PHF vrátí index ukazující na jeden z existujících přechodů. Klíč hashovací funkce se do tabulky ukládá pouze proto, aby bylo možno rozhodnout, zda daný přechod patří do množiny přechodů automatu či nikoliv. Brodnik ukázal, že pro efektivní řešení problému příslušnosti do množiny v konstantním čase je vhodné použít PHF, pokud je množina řídce zaplněna [17]. Platí tedy, že publikovaná architektura je pro řídce zaplněné přechodové tabulky vysoce efektivní.

Snižování nároků na paměť je možné pouze při snižování požadavků na vyhledávací jednotku. Jednou z možností je snížit přesnost vyhledávání zavedením malého množství chyb za účelem uspoření paměti. Pokud místo k-bitového klíče hashovací funkce použijeme jen n-bitový hash můžeme ušetřit $k-n$ bitů u každého klíče cenu chyby automatu s pravděpodobností $\frac{1}{2^n}$. Jedinou chybu, kterou automat může udělat je akceptování přechodu, který by neměl být proveden. Pokud se však síťový provoz liší od vzoru ve více než jednom symbolu, automat se korektně resetuje v následujícím kroku. Pokud rozdíl mezi útokem a provozem na síti byl pouze jediný symbol ve kterém došlo k chybě, vyhledávací jednotka vrátí chybu. Další možností selhání je situace, kdy k chybě dojde těsně před začátkem útoku. V tomto případě automat útok nedetekuje, neboť na jeho počátku se nebude nacházet v počátečním stavu pro daný útok. Lze tedy odhadnout, že jednotka nedetekuje útok přibližně s pravděpodobností $\frac{1}{2^n}$. Nesmíme zapomenout, že současné vyhledávací metody pracují na úrovni paketů a tudíž přehlédnou každý útok, který je rozdělen mezi několik paketů, čímž ve výsledku dosahují nižší spolehlivosti pro útoky které se mohou nacházet kdekoliv v síťovém toku. Popsanou myšlenku jsem prezentoval v článku [18]

Další možností řešení nedostatku drahé paměti na čipu je využití externí paměti z řádově větší kapacitou. Nevýhodou externí paměti je vysoká latence, což vede ke zpomalení doby prohledávání jednoho toku. Vyhledávací jednotka založená na DFA může přepínat mezi několika toky pouhou změnou registru reprezentujícího aktivní stav automatu. Vysokorychlostní sítě se v současné době používají převážně k implementaci páteřních linek, kde jsou data již z principu tvořena velkým množstvím pomalejších toků. Pokrytí latence vyhledáváním ve více tocích tedy dosáhneme stejného výkonu jako využitím pamětí s minimální latencí. Obě navržené úpravy lze snadno kombinovat.

4 Experimentální výsledky

Základním prvkem navržené architektury je výpočet perfektní hashovací funkce [15]. Výsledná rychlost tohoto algoritmu přímo ovlivňuje propustnost celého vyhledávacího systému. Původní algoritmus byl navržen pro použití na moderních CPU. V rámci své práce jsem v algoritmu provedl několik jednoduchých úprav aby bylo jeho mapování do hardwaru jednodušší. Základní úprava spočívala v nahrazení operace sčítání a modulo jednodušší operací XOR.

Dalším zásadním rozdílem mezi implementací v FPGA a CPU je správa paměti. Při implementaci v softwaru máme k dispozici jednu paměť, ke které máme přístup z jakéhokoliv místa výpočtu a snažíme se do ní ukládat co nejmenší datové struktury. Naopak v FPGA připojujeme paměť přímo k jednotce a připojená paměť je pak rezervována pro tuto jednotku. V FPGA se tedy snažíme paměť minimalizovat s ohledem na nejhorší možný stav a její využití v běžném provozu již není tak kritické. Z tohoto důvodu byl algoritmus PHF upraven tak, aby vždy počítal s pamětí o velikosti mocniny dvou i když by bylo teoreticky možné využít méně paměti.

Velikost paměti potřebné pro vyhledávací jednotku je možné vyjádřit jako počet záznamů v přechodové tabulce násobeno velikostí záznamu. Počet záznamů je závislý na velikosti a složitosti vyhledávaných výrazů. Největší část záznamu v přechodové tabulce tvoří validační informace. První experiment ukazuje závislost rychlosti jednotky implementované v obvodu Virtex6 na velikosti záznamu v přechodové tabulce. Druhý experiment pak ukazuje využití zdrojů FPGA v závislosti na šířce záznamu.

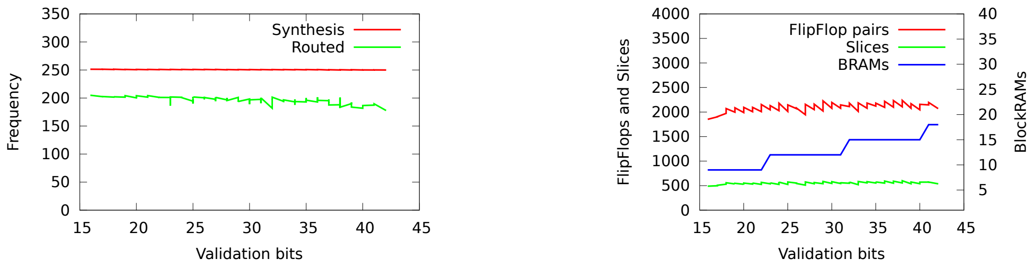

Figure 1: Využití zdrojů podle velikosti klíče

Z grafů je patrné, že frekvence je nezávislá na šířce reprezentace přechodu. Architektura je tedy rozšiřitelná pro vysoké propustnosti zpracováním více znaků současně, což se projeví nárůstem abecedy a tedy také nárůstem počtu bitů potřebných k reprezentaci přechodů. Vzhledem k omezenému počtu paměti, jsem provedl druhý experiment, při kterém jsem analyzoval vliv snížení spolehlivosti automatu na úsporu paměti.

Tučný průběh v grafu 2 ukazuje závislost pravděpodobnosti korektního vyhledání na počtu bitů použitých pro reprezentaci klíče a zelený ukazuje závislost velikosti přechodové tabulky na počtu bitů klíče. Modrý průběh ukazuje paměť potřebnou pro implementaci bezchybného vyhledávání. Rozdíl mezi zeleným a modrým průběhem tedy ukazuje míru úspory paměti a hodnota červeného průběhu ukazuje spolehlivost výsledné jednotky při zpracování toku o velikosti 10KB v procentech.

5 Praktická aplikace výsledků

Cílem práce je návrh a implementace architektury pro vysokorychlostní vyhledávaní regulárních výrazů. Výsledná architektura bude použita v rámci návrhu bezpečnostních prvků pro páteřní linky, což povede

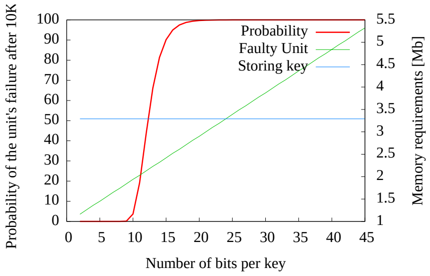

Figure 2: Spolehlivost jednotky podle počtu bitů klíče

k zvýšení bezpečnosti internetu. Realizačním výstupem bude IPcore umožňující vysokorychlostní vyhledávání regulárních výrazů s podporou vyhledávání v několika datových proudech současně.

6 Závěr

Ve své práci jsem navrhl a implementoval architekturu umožňující vyhledávaní regulárních výrazů v síťových tocích na moderních páteřních linkách. Implementovaná jednotka je schopna vyhledávat s propustností 5 a více Gbps na jeden tok [18]. Pro zvětšení množiny regulárních výrazů může být jednotka připojena k externí paměti. Při použití externí paměti jednotka zpracovává více toků paralelně, aby zakryla latenci paměti.

V další práci bych se chtěl zaměřit na efektivní implementaci zpracování více znaků v jednom taktu, což povede k vyšší propustnosti vyhledávací jednotky nebo k nižšímu využití pamětí.

Acknowledgment

Tato práce vznikla díky podpoře z projektu Pokročilé bezpečné, spolehlivé a adaptivní IT, VUT v Brně, FIT-S-11-1

References

[1] "Snort homepage." [Online]. Available: www.snort.org

[2] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*. Berkeley, CA, USA: USENIX Association, 1998, pp. 3–3. [Online]. Available: http://portal.acm.org/citation.cfm?id=1267549.1267552

[3] L. Schaelicke, T. Slabach, B. J. Moore, and C. Freeland, "Characterizing the performance of network intrusion detection sensors." in *RAID'03*, 2003, pp. 155–172.

[4] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Improving the accuracy of network intrusion detection systems under load using selective packet discarding," in *Proceedings of the Third European Workshop on System Security*, ser. EUROSEC '10. New York, NY, USA: ACM, 2010, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/1752046.1752049

[5] R. Smith, C. Estan, and S. Jha, "Backtracking algorithmic complexity attacks against a nids," in *Proceedings of the 22nd Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 89–98. [Online]. Available: http://portal.acm.org/citation.cfm?id=1191820.1191867

[6] M. Bando, N. S. Artan, R. Wei, X. Guo, and H. J. Chao, "Range hash for regular expression pre-filtering," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '10. New York, NY, USA: ACM, 2010, pp. 20:1–20:12. [Online]. Available: http://doi.acm.org/10.1145/1872007.1872032

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

[8] C. R. Clark and D. E. Schimmel, "Scalable Pattern Matching for High-Speed Networks," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, California, 2004, pp. 249–257.

[9] R. Smith, C. Estan, S. Jha, and I. Siahaan, "Fast signature matching using extended finite automaton (xfa)," in *ICISS '08: Proceedings of the 4th International Conference on Information Systems Security*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 158–172.

[10] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, ser. ANCS '07. New York, NY, USA: ACM, 2007, pp. 155–164. [Online]. Available: http://doi.acm.org/10.1145/1323548.1323574

[11] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 339–350.

[12] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. Di Pietro, "An improved dfa for fast regular expression matching," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 29–40, 2008.

[13] C. Shannon and D. Moore, "The Spread of the Witty Worm ," *IEEE SECURITY and PRIVACY*, vol. 2, no. 4, pp. 46–50, 2004.

[14] J. Kastil and J. Korenek, "Hardware accelerated pattern matching based on deterministic finite automata with perfect hashing," in *DDECS 2010*, 2010, pp. 149–152.

[15] F. C. Botelho, R. Pagh, and N. Ziviani, "Simple and Space-efficient Minimal Perfect Hash Functions," in *In Proc. of the 10th Intl. Workshop on Data Structures and Algorithms*. Springer LNCS, 2007, pp. 139–150.

[16] J. Kastil, J. Korenek, and O. Lengal, "Methodology for fast pattern matching by deterministic finite automaton with perfect hashing," in *DSD '09: Proceedings of the 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 823–829.

[17] A. Brodnik and J. I. Munro, "Membership in constant time and almost-minimum space," *SIAM J. Comput.*, vol. 28, no. 5, pp. 1627–1640, 1999.

[18] J. Kastil and J. Korenek, "High speed pattern matching algorithm based on deterministic finite automata with faulty transition table," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '10. New York, NY, USA: ACM, 2010, pp. 7:1–7:2. [Online]. Available: http://doi.acm.org/10.1145/1872007.1872017