Fast and Scalable Packet Classification Using Perfect Hash Functions

Viktor Puš *
CESNET z. s. p. o.
Zikova 4, 160 00 Prague, Czech Republic
pus@liberouter.org

Jan Kořenek †
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
korenek@fit.vutbr.cz

Klasifikace paketż je dżležitá operace pro aplikace jako smżrovaże a firewally. Bylo vytvożeno mnoho algoritmż, ale żádný se nemżże rovnat rychlosti TCAM. Navrhujeme nový hardwarový algoritmus klasifikace paketż. żežení je založeno na dekompozici problému a je urżeno pro vysokorychlostní sítż. Jedinežnou vlastností algoritmu je konstantní żasová složitost. Algoritmus provede pżesnż dva pżístupy do externí pamżti pro klasifikaci jednoho paketu. S využitím FPGA a jedné SRAM je možné dosáhnout propustnosti 150 milionż paketż za sekundu. To odpovídá 100 Gb/s na nejkratżích paketech. Dalżí zrychlování je možné s více nebo rychlejżími SRAM.

ABSTRACT

Packet classification is an important operation for applications such as routers, firewalls or intrusion detection systems. Many algorithms and hardware architectures for packet classification have been created, but none of them can compete with the speed of TCAMs in the worst case. We propose new hardware-based algorithm for packet classification. The solution is based on problem decomposition and is aimed at the highest network speeds. A unique property of the algorithm is the constant time complexity in terms of external memory accesses. The algorithm performs exactly two external memory accesses to classify a packet. Using FPGA and one commodity SRAM chip, a throughput of 150 million packets per second can be achieved. This makes throughput of 100 Gbps for the shortest packets. Further performance scaling is possible with more or faster SRAM chips.

*This research has been partially supported by the Research Plan No. MSM, 6383917201 – Optical National Research Network and its New Applications
†This research has been partially supported by the Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'09, February 22–24, 2009, Monterey, California, USA.
Copyright 2009 ACM 978-1-60558-410-2/09/02 ...$5.00.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays, Algorithms implemented in hardware*;
C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*

General Terms

Design, Performance, Security

Keywords

Packet Classification, FPGA, SRAM

1. INTRODUCTION

With the rapid development of computer networks, traffic filtering has become one of the first steps in securing any network or computer. Basic traffic filtering device is the firewall, which makes per-packet decision based on the given set of rules. As network speeds are increasing, the demand for the speed of packet classification algorithms is also growing. Software solutions for the packet classification problem are available [3, 4], but their performance is not sufficient for wirespeed processing in the highest speed networks. Existing hardware approaches also do not fulfill performance requirements, or they require excessive amount of memory.

A classification algorithm contains a set of rules ordered by priority. Each rule defines a condition for all significant packet header fields. These fields are typically: Source IP Address, Destination IP Address, Source Port, Destination Port, Protocol. A condition may be exact match, prefix match (usually for IP addresses), range match (for ports), or a wildcard (matching any value). The goal of a packet classification algorithm is to find the matching rule with the highest priority. The output of the algorithm is then the number of the matched rule.

The traditional method of classifying packets makes use of Ternary Content Addressable Memories (TCAMs). However, the TCAM is an expensive device with high power-consumption [2]. It also matches only words with fixed data width and can limit throughput for complex rules. Therefore, algorithmic solutions without the use of TCAMs has become a research subject. While many algorithms have been published [7, 15, 17], none of them can match TCAM speed, because all existing algorithms require non-constant number of memory accesses in the worst case. This must be compared to the performance of TCAM solution, which classifies packet in a single memory access and the throughput is

guaranteed. We propose a new packet classification method which uses SRAM to store necessary data, and FPGA to implement the algorithm. We will show that our solution is fully competitive to TCAM.

The rest of the paper is organized as follows: in the next section we discuss the related work and point out disadvantages of current solutions. Section 3 introduces a new packet classification algorithm. The most innovative part of the algorithm is described in detail in Section 4. Experimental results of our work are summed up in Section 5, and Section 6 concludes the paper. Finally, in Section 7 we discuss the possibilities of the future work in this area.

2. RELATED WORK

As the packet classification problem is inherently hard from a theoretical standpoint [7], a large number of hardware and software solutions [7, 15, 17] have been proposed. Solutions are based on exhaustive search, decision tree and grid-of-tries.

An interesting approach was introduced by Gupta in Hi-Cuts algorithm [14]. Hi-Cuts algorithm creates decision tree which cuts the packet space across one dimension at each level. The scheme was further improved by Hyper-Cuts [21] to cut the space across more dimensions at each level. In the Lucent bit vector scheme [17], range search is performed in each dimension, returning a vector with one bit for each rule. If one rule dimension is matched, its bit is set and a simple logical conjunction over all dimension vectors returns matching rules. Several improvements [8, 19, 22] were introduced later.

From the wide choice of available algorithms, we discuss only those which are related to our work. All of them belong to the family of decomposition-based methods. In decomposition methods, packet classification is divided into several steps. First step is the Longest Prefix Match (LPM) operation, which is performed independently in each dimension. From the given set of prefixes with various lengths, the LPM algorithm finds the one that best fits to the given full-length value. Range conditions (such as port ranges) in the ruleset are converted to prefixes, so that LPM may be performed in all dimensions.

LPM operation is performed in IP packet routing, so it is well studied topic. In fact, packet routing is a classification in one dimension only – the destination IP address. Basic algorithm for LPM is a trie, often modified to process more input bits in each step and to reduce memory consumption. Popular example of such algorithm is Tree Bitmap [12], but there are also many other solutions [10, 16, 18].

After LPM, all results must be combined together to get the resulting rule number. Basic Crossproduct algorithm [24] precomputes a crossproduct table, which contains resulting rule numbers for all possible combinations of prefixes. Because of the multiplicative nature of the crossproduct, this table may become extremely large. This table is implemented as a hash table, which yields issues with collisions. The whole crossproduct word must be stored in the table to detect a hash collision. In case a collision occurs, there must be a pointer to the next item. In this way, a linked list is created and performance may be reduced significantly.

Other method of combining LPM results together is the Distributed Crossproducting of Field Labels [25]. LPM is modified to return all valid prefixes (not only the longest one) for the given field value. What follows is the hierarchical structure of small crossproduct engines. Inputs of each engine are two sets of prefixes (or Labels, in general). Engine then performs set membership query for each possible pair of Labels. Result of the engine is another set of Labels. The result of the last engine is in fact a set of rules, from which the one with the highest priority is selected. Even when crossproducting is performed in a distributed way, it is still a weak point of the algorithm, because it is multiplicative in nature. If, for example, both input sets of the crossproducting engine have 10 items, then the engine has to perform $10 \times 10 = 100$ set membership queries.

Fast Packet Classification Using Bloom filters [11] brings further improvements to decomposition methods. The authors of this work replace crossproducts by *pseudorules*. To cover all valid combinations, certain rules are added to the ruleset. In fact, a pseudorule is always a special case of some rule. Example of pseudorules generation can be seen in Figure 1.

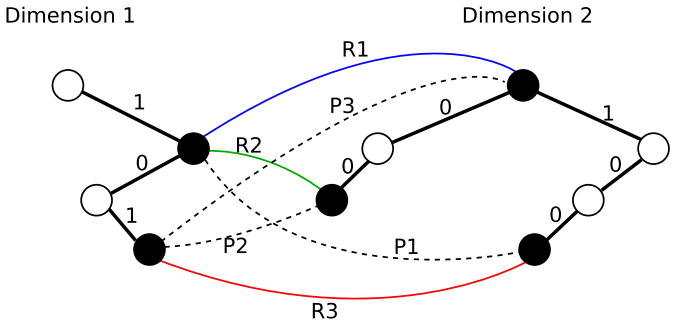

Figure 1: Three rules $R1, R2, R3$ and three added pseudorules.

Rule	Dimension 1	Dimension 2	Target rule
R1	1*	*	
R2	1*	00*	
R3	101	100	
P1	1*	100	R1
P2	101	00*	R2
P3	101	*	R1

Table 1: Rules and pseudorules.

We can see classification in two dimensions with three rules. For example, there is no rule for packet with header fields $(111, 100)$, but the correct result is rule $R1(1*, *)$[1]. Therefore pseudorule $P1(1*, 100)$ has to be added to cover this situation. Table 1 contains all rules and pseudorules together. *Target rule* in this table points to the correct classification result of pseudorule.

The generation of pseudorules has the character of crossproducting, and it may potentially expand the ruleset significantly, but not all possible combinations of prefixes need to be added. If the *universal rule* (a rule covering all possible packets) were in the ruleset, then all possible combinations would have to be added, but this rule can be removed from

[1]Symbol * denotes prefix or wildcard

the ruleset and returned only if no other rule matches the packet.

Because pseudorules expansion is similar to crossproduct, the article provides heuristics on how to break ruleset into several subsets, eliminating the majority of pseudorules. The paper also identifies rules that generate excessive amount of pseudorules. These rules are called *spoilers* and are removed to small on-chip TCAM. LPM operation is slightly modified to return result for each subset, because subsets may contain different prefixes. One Bloom filter is associated with each subset to perform set membership query. If the result is true, one Rule Table memory access is performed to retrieve resulting rule or pseudorule.

However, this scheme has several important disadvantages. Firstly, Rule Table is implemented using external SRAM, which imposes high requirements on SRAM throughput. If a rule format is very wide (for classification in more than five dimensions), the time required to read out one rule is also longer. We claim that Rule Table must be stored in an on-chip memory in order to achieve higher throughputs.

Secondly, an inherent property of the Bloom filter is nonzero probability of false positive errors. This may lead to a situation, when there exists a packet that causes false positives in several Bloom filters, resulting in several external memory accesses. If huge amount of such packets occurs in the network (e.g. during an attack), the classification algorithm slows down significantly.

Thirdly, Bloom filters are used only to reduce external memory accesses. Nevertheless, their implementation consumes on-chip resources which could be used in a more useful way.

Finally, the worst-case memory requirements are still exponential, even with the ruleset division into subsets and the use of TCAM for spoilers. But the algorithm does not try to reduce the size of one item – the whole rule or pseudorule has to be stored in an off-chip memory.

3. ALGORITHM

We propose a novel high-speed hardware-suited packet classification algorithm, which has a modular design and removes the drawbacks of Bloom filters, which were mentioned in Section 2. The algorithm consists of LPM for rule fields followed by a mechanism to search a rule. LPM and the rest of the classification algorithm have a very simple unidirectional interface and both parts may be changed separately, when a better solution is available. Recent research results for LPM operation have outstanding results even over 100 Gbps [18], therefore we do not propose any new architecture for LPM and focus on the rule searching mechanism.

With the bandwidth of the off-chip memory being the performance and scaling limitation for many existing solutions, we try to reduce amount of off-chip memory accesses for every incoming packet. Therefore, we propose to store the whole Rule Table in the on-chip memory using simple rule compression scheme and utilize the off-chip memory only to search the rule.

The primary goal is to find a solution with the constant packet rate and a good scalability with the size of the rule. Therefore, we propose to use a perfect hashing mechanism to provide the Rule Table search in a constant time and utilize the off-chip memory to store Perfect Hash Table.

The process of packet classification is divided into three basic steps (see Figure 2). The first step is the Longest Prefix Match operation, which is similar to approaches mentioned in Related work. The second step is mapping LPM results to the rule number, where we propose to use perfect hash function to perform fast searching. Even if the packet does not match any rule, the hash function will map the packet to some rule number. Because such invalid mapping can occur, it is necessary to include the third step, in which the packet is checked against the resulting rule. Therefore, the complete Rule Table has to be stored in the third step.

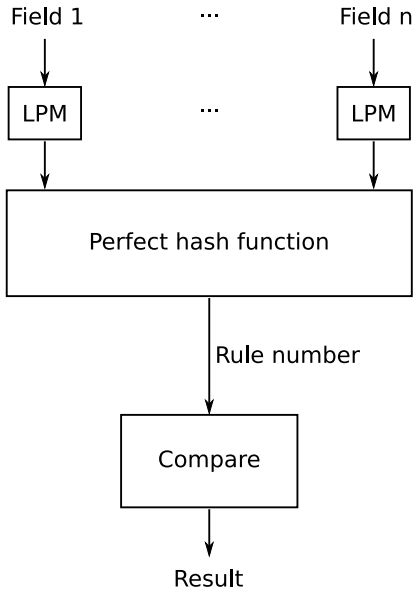

Figure 2: Three basic steps of the algorithm.

The perfect hash function must find a correct rule number for every packet. Thanks to LPM used in the first step, the packet state space is reduced significantly. The hash table is stored in the off-chip memory and its construction is described in the Section 4 of the paper.

The last part of the algorithm is the comparison of the rule to corresponding packet header fields. As high throughput memory is needed to read a rule, the Rule Table is stored in on-chip memories. The on-chip memory has a limited capacity, therefore all rules are compressed to save as many memory resources as possible. We propose simple prefix indexing scheme (see Figure 3) to reduce the Rule Table size significantly. The rule itself contains only several indexes to adjacent Prefix Tables, where all prefixes are stored. Port number is stored directly in the Rule Table, because it is a small field. This exploits the property of Rule Tables we and others [25] have observed: the number of unique prefixes in each dimension is usually quite small, therefore Prefix Tables will be also small. The experimental results for rulesets mentioned later in the text show that memory was reduced at least by one half, compared to simple direct storage of rules.

This compression makes use of hardware parallelism, because all Prefix Tables are accessed in the same time. Off-chip Rule Table implementation cannot be fast enough when using this scheme.

4. PERFECT HASH FUNCTION

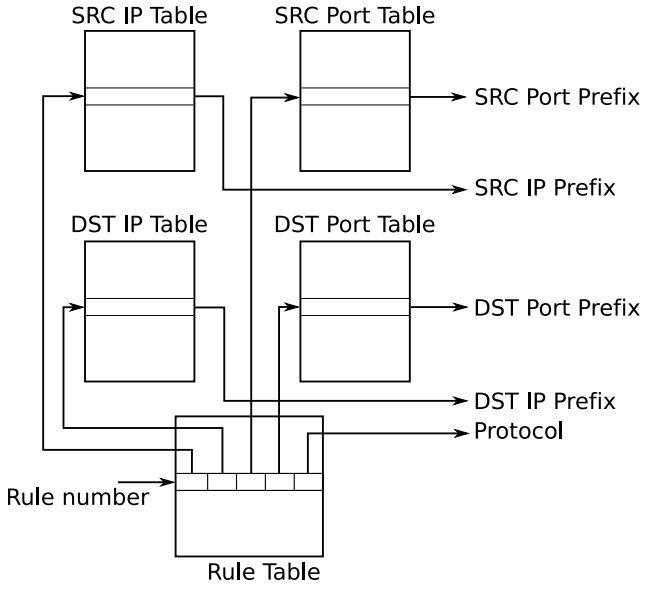

Figure 3: Prefix indexing scheme to reduce the Rule Table size.

The problem of designing the perfect hash function could be described as follows: each independent LPM returns one word for every packet. Each word represents one prefix and all prefixes together have a meaning of one (possible) pseudorule. Each pseudorule is associated with one rule. We seek a function mapping all valid pseudorules to their associated rules. Invalid pseudorules (for a packet that matches no rule) may be mapped to any rule, because this false positive is resolved later by simple comparison.

We chose static perfect hashing, because dynamic schemes have significantly greater overhead. Instead of dynamic rules insertion or removal, we can simply recompute the whole static perfect hash. We propose to use a perfect hash construction algorithm described in [9] to get a hash function, which for each pseudorule returns a number of its associated rule. From a wide choice of static perfect hashing schemes, we chose this one because of its simplicity and good results. Perfect hashing has been proposed to be used in networks applications before [20, 13, 23, 6], but according to our knowledge, the idea of using perfect hash functions is novel in the field of multidimensional packet classification.

The perfect hash construction algorithm creates acyclic graph, where edges are the keys, and vertices are results of two different hash functions. Vertices are then assigned values so that they sum up to the desired hash value. Detailed description can be found in [9]. The algorithm consists of seven basic steps:

1. Input: K keys, each associated with a number which it is to be hashed to.
2. Create graph with $N = cK$ vertices, where $c > 1$.
3. Pick any two different ordinary hash functions f_1, f_2 that output values $0 \ldots N-1$.
4. For each key, compute $h_1 = f_1(key), h_2 = f_2(key)$, draw an edge between vertices h_1 and h_2 of the graph and associate the desired hash value with that edge.
5. Check if the graph is acyclic. If not, increase c and go to step 2.
6. Associate values to each vertex such that for each edge you can add the values of both its vertices and get the desired value for the edge. This may be done by depth-first search algorithm, because the graph is acyclic.
7. f_1, f_2 and vertex values now make up the desired function.

In our algorithm, keys are rules and pseudorules in the form of concatenated LPM results, and associated numbers are numbers of the correct rule. This way, we get a function that hashes rule and all its associated pseudorules directly to the correct rule number. In fact, we introduce intended collisions of the hash function. The idea of intended hash collisions is a non-traditional usage of perfect hash functions. The important point is that none of pseudorules is stored in our scheme. Therefore, we save significant amount of memory.

Table 2 and Figures 4 and 5 show how a graph for the example in Table 1 could look like. When the graph is created, the hash function is simple. At first, two different hash functions are evaluated over the input word. Then two vertex values are read from the Vertex Table and added. For each vertex, only one integer is stored.

Input word	f1	f2
<1*, *>	0	7
<1*, 00*>	6	0
<101, 100>	5	4
<1*, 100>	0	4
<101, 00*>	1	3
<101, *>	3	2

Table 2: Two hypothetical hash functions' results for inputs in the form of encoded and concatenated LPM results

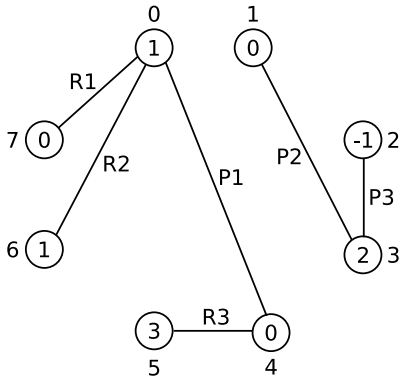

Figure 4: Example graph with 6 (pseudo)rules and 8 vertices.

By theory, acyclic graph with n edges must have at least $n+1$ vertices. This means that a table with more items than the number of rules and pseudorules is needed. The perfect hash algorithm usually needs greater overhead. Our

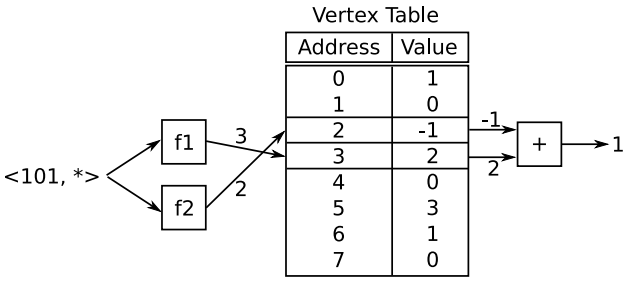

Figure 5: Example of computing perfect hash function.

experiments in Section 5 show that the table size must be approximately twice the theoretical minimum, which is good result among other perfect hash algorithms.

Similarly to [11], we use small on-chip TCAM to store spoilers and save significant amount of memory. Identifying the greatest spoilers is a complex task, which needs to be further investigated. For our experiments, we use semi-automatic method, and we intend to do more research in finding an automatic heuristic method with good results.

5. RESULTS

The proposed algorithm was implemented in FPGA and utilizes one external static memory. We have studied several rulesets to get information about typical properties of rules. Similarly to [25], we have found that the number of unique prefixes in each dimension is usually quite small (see Table 3). Therefore, the data structures for LPMs may be easily stored in small on-chip memories, either BlockRAMs or distributed memories. Also the Rule Table itself is not greater than a few kilobytes; therefore, we do not need an external memory for its storage.

The table of graph vertices may become large, therefore, we propose to use an external memory to store the Perfect Hash Table. On-chip memories could be used only in case of small rulesets. For example, Xilinx FPGA Virtex5 LX110 [5] contains 4 608 kb of BlockRAM memory, which gives us 262 144 vertices (suppose 18 bits for one vertex).

Ruleset	Rules	SRC Addr	DST Addr	SRC Port	DST Port	Protocol
fw1	32	13	2	10	22	4
fw2	58	26	24	4	1	2
fw3	103	28	48	36	1	4
fw4	171	84	84	1	6	3
synth1	40	12	19	10	22	5
synth2	49	35	41	8	22	3
synth3	49	26	14	14	1	4
synth4	70	27	62	1	48	3
synth5	82	20	37	3	3	4
synth6	100	73	85	1	54	4

Table 3: Numbers of unique prefixes in each dimension.

The proposed solution significantly reduces the bottleneck caused by the speed of external memory. Only two 18-bit words need to be read for every packet, which is many times less than one whole rule (or even worse, several rules) [11]. Moreover, performance of our algorithm is not affected by the complexity of rules. Other fields (i.e. MAC addresses, TCP flags, etc.) can be added to rules and the throughput remains the same, only on-chip Rule Table size increases linearly. It means that our solution scales well with complexity of rules.

5.1 Performance

Similarly to [11], we suppose 300 MHz DDR memory with the burst length of two words. The throughput of our solution is compared to the Crossproduct algorithm and Bloom filters in Table 4. We do not take into account the speed of LPM operation, because we consider it is fast enough [18]. It can be seen that our solution has a constant throughput and does not require very wide external memory data bus. The time to recompute all necessary data structures was always below 4 seconds. We use two Jenkins hash functions [1] with various seeds to implement the perfrect hash function.

Data Width	Crossproduct Based	Bloom Filter-Based 4	6	8	Perfect Hash
9	37.5	9.375	6.25	4.6875	150
18	75	18.75	12.5	9.375	150
36	150	37.5	25	18.75	150
72	300	75	50	37.5	150

Table 4: Throughput (in millions packets per second) for several data bus widths of 300 MHz DDR memory. Rule word width of 144 bits is considered. For the algorithm exploiting Bloom filters we consider three cases: match of four, six and eight rules for each packet.

5.2 Memory requirements

We performed pseudorules expansion and perfect hash function search for several rulesets from university campus network (fw) together with several synthetic ruleset generated by ClassBench [26] (synth) to determine off-chip memory requirements. Memory requirements are compared to Bloom filters-based and Crossproduct algorithm in Table 5.

As can be seen, numbers of graph vertices may become prohibitive for on-chip memories, but is acceptable for commodity SRAM chips. Each vertex is stored as one signed integer, actual range of vertex values determines number of bits required to represent it. If SRAM works with larger data width, words can be split into several parts to multiply the available table size.

Overall chip area is hard to compare to other solutions, because every implementation has many variable parameters (speed, number of stored prefixes, selection of classification dimensions). However, we provide informal comparison to [11]:

- Both schemes use LPM as the first step.
- In our solution, we need only two various on-chip hash functions to compute the perfect hash function, while [11] uses many hash functions to implement Bloom filters.

Ruleset	Rules	Crossprod.	Bloom F.	Perf. Hash
fw1	32	3 618	823	740
fw1	58	13 086	1 492	3 424
fw3	103	1 008 954	2 651	252 220
fw4	171	443 484	4 401	116 356
synth1	40	11 070	1 029	2 740
synth2	49	29 520	1 261	6 601
synth3	49	19 278	1 261	5 035
synth4	70	10 512	1 801	2 451
synth5	82	90 324	2 110	22 495
synth6	100	17 010	2 574	3 827

Table 5: Off-chip memory requirements (in Bytes) for several rulesets. For Bloom Filter-Based algorithm we assume that pseudorules expand the ruleset by the factor of 1.43 (average from the original paper [11]). For Crossproduct and Perfect Hash scheme we use on-chip TCAM for 16 spoilers.

- Our solution stores the ruleset in on-chip memories.
- In [11], small bit array is stored for each Bloom filter.
- Both schemes use external memory and other common blocks (packet recieve and transmit modules etc.).

To verify our results, we have implemented the described algorithm for the Virtex 5 LX110T FPGA. We used 125 MHz working frequency and set the throughput to two cycles per packet. This limitation of the particular implementation is induced by our current needs – we have connected two 10 Gbps network interfaces and one PCI-Express x8 bus to the FPGA.

We added four other classification dimensions: Source and Destination MAC address, TCP flags and Input interface number. We were able to load up to 1 000 rules into the device. Data structures (LPMs, Vertex Table, etc.) generation time was below 0.5 second on a PC with 2 GHz Intel Pentium processor. This is also the update delay if the ruleset changes.

Using the proposed algorithm, we have created two-port firewall with the constant aggregated throughput of 62.5 million packets per second.

6. CONCLUSION

We have proposed a novel algorithm for fast packet classification using perfect hash functions. Our algorithm introduces intended hash collisions to reduce memory requirements. By creating custom hash function, we make sure that all pseudorules are hashed to associated rule, which means that no pseudorule has to be stored in the memory and significantly less memory is needed. The results in Section 5 show that the only larger amount of memory is utilized to store Perfect Hash Table, even for large rulesets.

Because only two external memory accesses are needed to classify a packet, 150 million packets per second can be processed with commodity FPGA and SRAM. This packet rate corresponds to 100 Gbps Ethernet for the shortest packets. Moreover, the throughput doesn't depend on ruleset complexity and is well scalable with number of external memories.

According to our knowledge, the proposed algorithm is the first algorithm which requires reasonable amount of memory and has constant processing time even for complex ruleset. High throughput together with constant processing time makes the proposed algorithm fully competitive to widely used TCAM solutions. As the proposed solution uses commodity SRAM, the price and power consumption is significantly lower than classification with TCAM memory.

7. FUTURE WORK

We continue to explore this method to further improve memory efficiency by reducing size of the Vertex Table for large rulesets. If the memory requirements drop under certain limit, only on-chip memory can be used to store the Vertex Table and the classification process can be significantly faster. Moreover, if external memory is removed, the price and power consumption is decreased. We also believe that the idea of intended hash collisions has potential value for other tasks which allow relatively slow precomputation, but require extremely fast search times.

8. REFERENCES

[1] A hash function for hash table lookup. http://burtleburtle.net/bob/hash/doobs.html, December 2008.

[2] IDT Generic Part: 75K72100. http://www.idt.com/?catID=58523&genID=75K72100 , June 2008.

[3] Netfilter: firewalling, NAT and packet managing for Linux. http://www.netfilter.org/ , June 2008.

[4] PF: The OpenBSD Packet Filter. http://www.openbsd.org/faq/pf/ , June 2008.

[5] Xilinx Virtex–5 Family FPGAs. Xilinx, Inc.

[6] N. S. Artan and H. J. Chao. Tribica: Trie bitmap content analyzer for high-speed network intrusion detection. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 125–133, May 2007.

[7] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to CAMs? In *INFOCOM*, 2003.

[8] F. Baboescu and G. Varghese. Scalable packet classification. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 199–210, New York, NY, USA, 2001. ACM.

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

[10] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using Bloom filters. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212, New York, NY, USA, 2003. ACM.

[11] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood. Fast packet classification using Bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 61–70, New York, NY, USA, 2006. ACM.

[12] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

[13] S. Giordano, F. Oppedisano, G. Procissi, and F. Russo. A novel high-speed micro-flows classification algorithm based on perfect hashing and direct addressing. *Global*

Telecommunications Conference, 2007. GLOBECOM '07. IEEE, pages 448–452, November 2007.

[14] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Proc. Hot Interconnects*, 1999.

[15] P. Gupta and N. McKeown. Algorithms for packet classification, 2001.

[16] P. Gupta, B. Prabhakar, and S. P. Boyd. Near optimal routing lookups with bounded worst case performance. In *INFOCOM*, pages 1184–1192, 2000.

[17] 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.

[18] H. Lee, W. Jiang, and V. K. Prasanna. Scalable High-Throughput SRAM-Based Architecture for IP Lookup Using FPGA. In *FPL '08*. IEEE, 2008.

[19] J. Li, H. Liu, and K. Sollins. AFBV: a scalable packet classification algorithm. *SIGCOMM Comput. Commun. Rev.*, 32(3):24–24, 2002.

[20] Y. Lu, B. Prabhakar, and F. Bonomi. Perfect hashing for network applications. *Information Theory, 2006 IEEE International Symposium on*, pages 2774–2778, July 2006.

[21] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 213–224, New York, NY, USA, 2003. ACM.

[22] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using FPGA. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245, New York, NY, USA, 2005. ACM.

[23] I. Sourdis, D. Pnevmatikatos, S. Wong, and S. Vassiliadis. A reconfigurable perfect-hashing scheme for packet inspection. *Field Programmable Logic and Applications, 2005. International Conference on*, pages 644–647, Aug. 2005.

[24] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, 28(4):191–202, 1998.

[25] D. Taylor and J. Turner. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, pages 269–280, July 2005.

[26] D. E. Taylor and J. S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15(3):499–511, 2007.