Rule Caching for Packet Classification Support

Joji Philip, Manish Taneja, and Roberto Rojas-Cessa

Abstract—**The growth of the Internet and requirements for enhanced flexibility and versatility have resulted in packet classification becoming an essential part of network systems such as routers and firewalls. Classification of packets into flows based on multiple header fields is a complex problem. Several algorithms have been proposed to achieve wire speed classification performance. However, an ever widening gap between wire speed and memory access speed continues to motivate the quest for techniques to enhance classification performance. In this paper, we propose a caching technique to improve search performance of any classification algorithm using characteristics inherent to Internet traffic.**

Index Terms—**Caching, Cache memory, correlated traffic, packet classification, rule caching.**

I. Introduction

The internet has provided best-effort service until now, servicing packets in a first-come first-serve manner. The deployment of internet for commercial applications called for service differentiation techniques. This service differentiation was not provided by traditional routers because they treated all traffic going to the same Internet destination identically. Routers are now called upon to provide different qualities of service to different applications which means routers need new mechanisms such as admission control, resource reservation, per-flow queuing, and fair scheduling. All of these mechanisms require the router to distinguish packets belonging to different flows. The process of categorizing packets into flows in an Internet router is called packet classification. Such classification allows various forms of service differentiation: blocking traffic sent by insecure sites (firewalls), preferential treatment for premium traffic (resource reservation), and routing based on traffic type and source (QoS routing).

Packets are classified by matching the header fields of incoming packets with a set of rules in the database of the screening router. Fundamentally, it is a search problem where the search key is a combination of multiple header fields of the packet and the search space is the rules defined in the filter database. Specifically, one of the main problems involved in packet classification is determining the lowest-cost matching rule for an incoming packet. The classifier, or rule database, router consists of a finite set of rules, R_1, R_2…, R_N. Each rule is a combination of K values, one for each header field. Each field in a rule is allowed the following three kinds of matches. *Exact match*: In this matching, the header field of the packet should exactly match the rule field. *Prefix match*: In this matching, the rule field should be a prefix of the header field; it could be useful for blocking access from a subnet. *Range match*: In this matching, the header field should lie within the range specified by the rule, it could be useful for specifying port number ranges. Each rule R_i has a directive associated with it that specifies how the packet is to be treated, whether the packet is to be forwarded or blocked etc. Since a packet may match multiple rules in a database, each rule R in the database is associated with a nonnegative number, cost(R) - Cost Function. The classification process should yield the best matching rule corresponding to the least cost rule. The cost function generalizes the implicit precedence rules that are used in practice to choose between multiple matching rules. One possible approach is to place the rules in a specific linear order such that each rule can take precedence over a subsequent rule, i.e. cost(R) equals the position of rule R in the database- First Matching Rule.

II. Related Work

Packet classification on multiple header fields is theoretically a complex problem [8] and it has received significant attention by the research community. Due to its complexity, classification can potentially be the bottleneck process of network systems. Most of the existing research work has focused on improving the search performance to keep up with wire speed. Existing algorithms can be broadly classified as *algorithmic* and *architectural* [9]. There has been a long thread of algorithmic solutions proposed for the classification problem [1]-[7]. A parallel thread of research [10]-[14],

The authors are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102. Roberto Rojas-Cessa is the corresponding author. Email: rojas@njit.edu.

This work was partially supported by National Science Foundation under Grant Award 0435250.

has focused on finding architectural solutions to the problem with TCAM evolving as the most widely used device for classification in current day network systems. However TCAMs remain unsuitable for large classifiers because of low density, high power consumption and cost.

Promising new schemes have been proposed which exploit the statistical structure of the rule database and inherent characteristics observed in internet traffic. A comprehensive survey of existing schemes for packet classification can be found in [9], [15].

Many of the lookup algorithms for classification are well suited to hardware implementation. However the dynamic nature of the classifying rules requires that the rule database be stored in memory. Further, memory access speeds have not been able to keep up with improvements in hardware performance and increase in network traffic rates or wire speed. Even the most efficient algorithms for classification require several memory accesses for each packet and hence memory access speeds forms the bottleneck for classification performance in network systems.

Caching is a classical approach to mitigate the performance bottleneck caused by memory access speeds in Computer systems [22]. Hence this design space can be explored for the possibility of enhancing performance of classification algorithms. Several studies have observed strong temporal locality in internet traffic, i.e. arrival of a packet on an internet link implies a very high probability of arrival of other packets belonging to the same flow in the near future [16], [17]. Several research papers have attempted to apply caching schemes to improve performance of packet classifiers [18]-[21]. One technique called *flow caching* involves caching of packets. Header fields from incoming packets are used to create a unique signature for identifying packets belonging to a common flow. The signature is stored in a hash table along with a pointer to the best matching rule (BMR) for packets with this signature (actually a pointer to the directives associated with the BMR is needed). The first packet in a sequence of packets belonging to a flow would be searched in the rule database and its signature would be added to the cache along with the BMR association. Subsequent packets of the flow have a cache hit and obtain the best matching rule from the cache thus avoiding the need to perform search on the rule database. A cache eviction policy such as least recently used (LRU) or LFU has to be implemented to evict old signatures from the cache, thus attaining good hit rates with a reasonable cache size. The ageing interval has to be carefully chosen depending on the expected lifetime of packets belonging to a flow. With this approach the classification problem is reduced to an average case complexity of *O(1)* irrespective of the complexity of the original classification algorithm.

III. Flow Caching

Traffic sessions on the internet are comprised of a finite number of packets, all of which need similar treatment at a classifier. Further, packets belonging to a single flow occur in close proximity on the time axis. These characteristics suggest that applying some form of caching schemes can improve search performance of classification algorithms. However, Any caching technique improves the search performance at the cost of increased storage. The previous section introduced an approach using caching of flow identifiers.

As link speeds have increased, availability of sufficient temporal locality and hence the effectiveness of caching schemes has become questionable. This is due to the fact that average bandwidth requirement of individual flows has not increased at the same rate as the line rates. Thus on a high capacity link, several flows could be active simultaneously. Moreover, most flows are active only for a small duration resulting in a rather dynamic flow population at the network systems. Hence the number of concurrent flows expected on classification nodes is projected to increase as the link capacities on the internet increase. These factors indicate that the size of cache memory must scale with the link rate to ensure good hit rates in flow caching schemes.

IV. Rule Caching

Numerous algorithmic approaches proposed for packet classification offer varying tradeoffs between search speed, storage requirement, and update speed. We have explored the *caching* domain and propose a generic scheme that can be used to improve the search performance of any classification algorithm.

As mentioned in the previous section, flow caching schemes do not seem to scale well with incoming traffic at the classification nodes. The total number of rules in a classifier would generally be much smaller than number of concurrent flows active at that node. Moreover, the flows would change more dynamically than updates to the rule database. These observations suggest the possibility of caching rules instead of flows. The approach proposed in this paper involves maintaining a shadow rule database which acts as a cache storing the subset of rules which have highest probability of being the best matching rule for the packets being processed. A rule is added to the shadow database when a programmable threshold of packets finds that rule to be the BMR after search in the main rule database. All packets belonging to the same flow are expected to be classified by the same BMR. Hence once added to the shadow database, packets can arrive at their BMR much faster than an exhaustive search of the entire rule database. It should be noted that the asymptotic complexity of the search algorithm remains unchanged. Only the search space has been reduced, thus reducing the search time. Here again a suitable aging mechanism has to be implemented for cache eviction. It can be appreciated that the shadow database represents a dynamically changing rule database classifying packets in current time window. It should also be noted that high activity rules or rules which are shared by several flows would tend to remain in the cache for long durations improving the overall hit ratio.

V. DETAILS OF THE CACHING ALGORITHM

This algorithm defines two databases. First, the Main Rule Database where all the filters/rules are stored. In general all the packets are classified using this database. Second is the Rule Cache Database or *Shadow Database* where only a subset of rules are stored. Rules recording a threshold (user defined value) number of hits in the main rule database are assumed to be high frequency rules and are then moved to shadow database. For every incoming packet, this classification algorithm first searches the shadow database. If a best matching rule is found in the shadow database, it is used to classify the packet. However if a match is not found in the shadow database, the packet has to be matched with the rules in the main rule database. This adds an overhead to this classification scheme, but the inherent temporal locality of packets belonging to the same flow suggests that a high hit rate can be obtained in the shadow database. Rules in the shadow database have an initial timestamp, which is used to remove rules which have not had a cache hit for a programmable aging interval. Fig. 1 shows the basic architecture of the flow caching scheme.

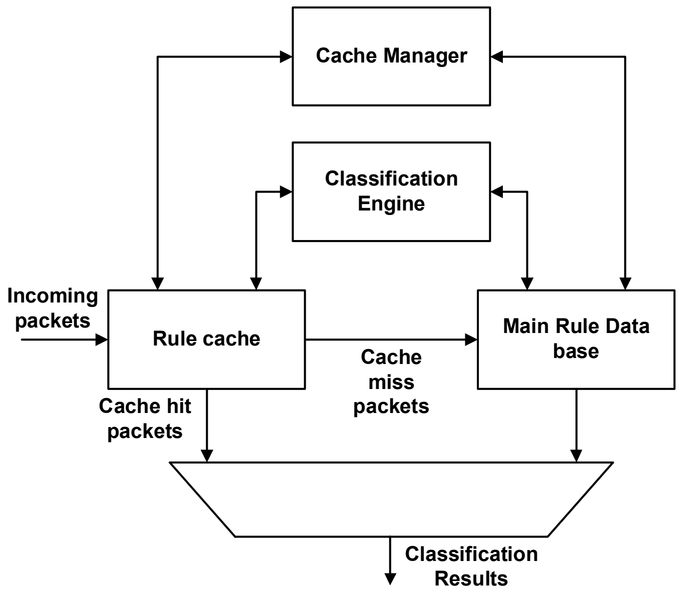

Figure 1 Rule caching architecture.

VI. SIMULATION STRATEGY

The proposed rule caching scheme was simulated to evaluate various metrics, such as hit ratio and cache size, and to estimate the impact of the tunable parameters on these metrics. *ClassBench* forms a primary component of the simulation setup. Quoting from the original paper introducing *ClassBench* [23]: *ClassBench* is a suite of tools for benchmarking packet classification algorithms and devices. *ClassBench* includes a *Filter Set Generator* that produces synthetic filter sets that accurately model the characteristics of real filter sets. Along with varying the size of the filter sets, it provides high-level control over the composition of the filters in the resulting filter set. The tool suite also includes a *Trace Generator* that produces a sequence of packet headers to exercise packet classification algorithms with respect to a given filter set. Along with specifying the relative size of the trace, the tool provides a simple mechanism for controlling locality of reference.

Fig. 2 illustrates the block diagram of the setup used for simulating the rule caching scheme.

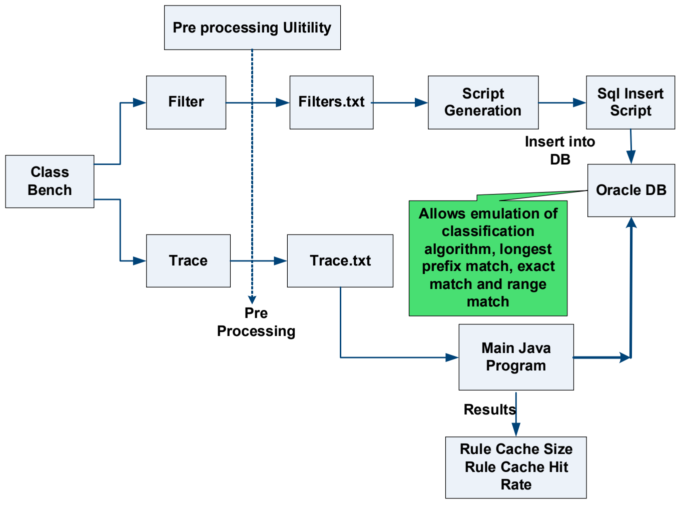

Figure 2 Simulation setup.

Filter and trace generators from *ClassBench* are used to generate rules and packet traces for the different tests. The tools allow us to build rules and traces according to several tunable parameters like locality of reference, randomness, and smoothing. Preprocessing is performed on the generated filter set and on traces to convert them into formats compatible with the simulation setup. The simulation involves reading each line from the trace file and searching the packet's header fields in filter set file to obtain the best match rule. Emulating the complex matching algorithms in software would involve considerable effort and the simulation times could be large for inefficient emulations. Hence an Oracle database was used to emulate the rule database and SQL commands were used to implement the searching and matching functions of the classifier. Oracle DB is already optimized for efficient search on large databases. Moreover, it enables longest prefix matching, range matching, exact matching and selection of highest priority rules to be trivially implemented using simple SQL commands.

A main Java code performs the function of parsing the trace file and searching each packet in the rule database by issuing appropriate SQL query commands. The program also implements cache management functions like cache population, aging, and eviction. The program also tracks and maintains variables for cache performance metrics such as hit rate and cache size. The functioning of the program is summarized in the flowchart shown in Fig. 3.

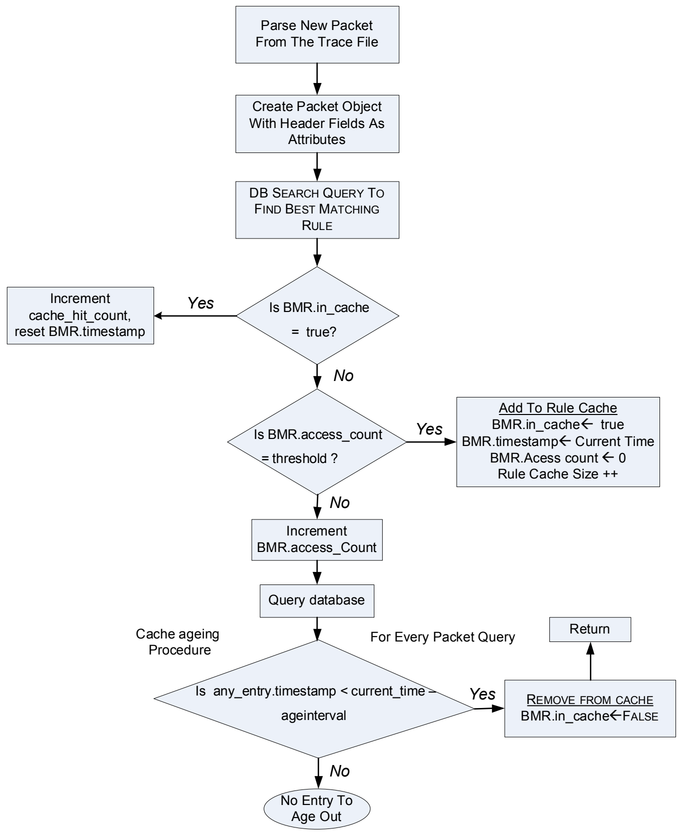
Parse New Packet From The Trace File
Create Packet Object With Header Fields As Attributes
DB Search Query To Find Best Matching Rule
Is BMR.in_cache = true?
Yes
Increment cache_hit_count, reset BMR.timestamp
No
Is BMR.access_count = threshold ?
Yes
Add To Rule Cache
BMR.in_cache← true
BMR.timestamp← Current Time
BMR.Acess count ← 0
Rule Cache Size ++
No
Increment BMR.access_Count
Query database
Cache ageing Procedure
For Every Packet Query
Is any_entry.timestamp < current_time – ageinterval
Yes
Remove from cache
BMR.in_cache←False
Return
No
No Entry To Age Out

Figure 3 Flow chart of the simulation java code.

VII. Results and Discussion

Tests were performed with different locality of reference, caching threshold and aging factors. The initial tests were run with 30,000 packets and the final tests with 100,000 packets. Test results are summarized in Table1. For each test, input parameters, such as the aging factor (AF) and the caching threshold factor (CF), and cache parameters, such as Hit ratio (CHR) and maximum cache size (CZ), are tabulated. From the results, it can be observed that the proposed scheme offers good cache hit ratios when there is high locality of reference. Moreover there is a tradeoff between hit ratio and the cache size. Caching threshold and aging factors have to be suitably tuned to find the right balance between hit rate and cache size.

VIII. Conclusions

It can be concluded that rule caching offers a significantly better alternative than flow caching for improving search performance of classification algorithms. Further research needs to be done to evaluate performance of this scheme for real network traces obtained at appropriate points in the internet.

Table 1

Test	Packets	Locality	TF	AF	CHR	CZ
Test1	30000	Low	0	NA	70.23%	-
Test2	30000	Low	0	50	51.15%	-
Test3	30000	Low	2	100	35.16%	-
Test4	30000	Low	4	200	30.13%	-
Test5	100000	High	0	100	98.348%	18
Test6	100000	High	3	500	95.945%	25
Test7	100000	Low	0	500	95.004%	52
Test8	100000	Low	3	1000	94.991%	35
Test9	100000	NA	0	NA	33.459%	1433
Test10	100000	NA	3	1000	12.383%	354
Test11	200000	High	3	500	95.225%	25

REFERENCES

[1] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," in Proceedings of ACM SIGCOMM, pp. 147-160, August 1999.

[2] P. Gupta and N. McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," Hot Interconnects VII, August 1999.

[3] T.V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multidimensional Range Matching," in Proceedings of ACM SIGCOMM, pp. 203-214, September 1998.

[4] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and Scalable Layer Four Switching," in Proceedings of ACM SIGCOMM, pp. 191-202, June 1998.

[5] F. Baboescu and G. Varghese, "Scalable Packet Classification," in Proceedings of ACM SIGCOMM, pages 191-202, August 2001.

[6] A. Feldmann and S. Muthukrishnan, "Tradeoffs for Packet Classification," in Proceedings of IEEE Infocom, pp. 1193-1202, March 2000.

[7] T.Y. C. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," in Proceedings of IEEE INFOCOM, Vol. 3, pp. 1213-1222, March 2000.

[8] M.H. Overmars and A.F. Van der Stappen, "Range searching and point location among fat objects," Journal of Algorithms, 21(3), pp. 629–656, November 1996.

[9] D.E. Taylor. "Survey & Taxonomy of Packet Classification Techniques". Applied Research Laboratory, Washington University in Saint Louis, May 2004

[10] G. Gibson, F. Shafai, and J. Podaima, "Content Addressable Memory Storage Device," United States Patent 6,044,005, SiberCore Technologies, Inc, March 2000.

[11] R.A. Kempke and A.J. McAuley, "Ternary CAM Memory Architecture and Methodology," United States Patent 5,841,874. Motorola, Inc, November 1998.

[12] A. J. McAulay and P. Francis, "Fast Routing Table Lookup Using CAMs," in Proceedings of IEEE INFOCOM, Vol. 3, pp. 1382-1391, 1993.

[13] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," in Proceedings of IEEE International Conference on Network Protocols (ICNP), pp. 120-131, 2003.

[14] K. Lakshminarayanan, A. Rangarajan, S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs." ACM SIGCOMM Computer Communication Review, Volume 35, Issue 4, pp. 193–204, October 2005.

[15] P. Gupta and N. McKeown, "Algorithms for Packet Classification," Computer Systems Laboratory, Stanford University," in Proceedings of IEEE Network, pp. 24-32, March 2001.

[16] K. Thompson, G. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," in Proceedings of IEEE Network, pp. 10-23, 1997.

[17] S. McCreary and K. claffy, "Trends in wide area IP traffic patterns - a view from ames internet exchange," Tech. Rep., CAIDA, 2000.

[18] J. Xu, M. Singhal, and J. Degroat, "A novel cache architecture to support layer-four packet classification at memory access speeds," in Proceedings of INFOCOM, Vol. 3, pp. 1445–1454, 2000.

[19] F. Chang, K. Li, and W.C. Feng, "Approximate cache for packet classification," In Proceedings of IEEE INFOCOM, pp. 2196-2207 vol. 4, 2004.

[20] M.M.I. Chvets, "Multi-zone Caches for Accelerating IP Routing Table Lookups," in Proceedings of High-Performance Switching and Routing, pp. 121-126, 2002.

[21] K. Li, F. Chang, D. Berger, and W.C. Feng, "Architectures for Packet Classification Caching," in Proceedings of IEEE ICON, pp. 111-117, 2003.

[22] J.L. Hennessy and D.A. Patterson, "Computer Architecture A Quantitative Approach," Morgan Kaufmann Publishers, Inc., 4 ed., 2006.

[23] D.E. Taylor and J.S. Turner, "ClassBench: A Packet Classification Benchmark," Tech. Rep. WUCSE- 2004-28, Department of Computer Science & Engineering, Washington University in Saint Louis, May 2004.