Packet Classification in Presence of Wildcard Expressions
EE384X Course Project

Omid Mashayekhi, and Vaibhav Chidrewar

***Abstract*— Most conventional packet classifiers fail to work when both rules and keys contain wildcard expressions. Moreover, usually they find only the highest priority filter that matches the incoming packet. However, new networking applications such as network analyzers, intrusion detection systems, and load balancers, depend on generalized expressions for rules and keys using wildcards, and require all (or the first few) matching filters during classification. The most commonly proposed hardware based solutions to multi-match classification are TCAM based and, as a result, suffer from several disadvantages such as higher cost, power consumption, and inflexibility. This paper introduces a software based algorithm which is memory efficient, does not need expansion for rules and keys containing wildcards, is scalable, and provides multi-match results.**

I. Introduction

PACKET classification is the fundamental building block of nowadays networking systems. In fact, we are almost at a turning point in designing data networks. While, hardware based packet switches with specialized software on them has been dominant approach in practical designs for the past few decades, we are at a turning point that Software Defined Networks (SDN) and programable switches are becoming more interesting in the industry. There are lots of advantages that expedite this new revolution, flexibility of programmable switches provided by new protocols like OpenFlow [1], novel network failure detection and analysis like FlowVisor [2] and Header Space Analysis [3], and facilitating the realization of innovations in networking research to name a few. In all of them, packet classification plays a major role.

As there are new practical applications for SDN, there is an inevitable demand for more generalized abstractions, and having wildcard expressions in rules and keys is one of the most common abstractions for flow definition. However, previous classifiers demonstrate poor performance either from scalability or manipulation latency points of view, in this general case. In this paper, we introduce a novel software based classifier implementation which is scalable and efficient from memory and latency points of view, in presence of wildcard expressions.

Next subsection will give a brief review about the related work. Just after that, it will be discussed why we need a new classifier and what should be expected.

A. Related Work

There have been lots of efforts to solve this very basic but important issue in the networking research area. Here we will just give a brief review of the most important methods.

This report has been prepared as a part of EE384X course project, Spring 2012, Stanford University, Stanford CA. (e-mail: omidm@stanford.edu, vaibhavc@stanford.edu).

Ternary Content Addressable Memories (TCAMs) have been adopted to solve the multi-match classification problem due to their ability to perform fast parallel matching. There has been extensive research on power and memory efficient multi-match packet classification algorithms using TCAM [4], [5], [6], [7]. However, TCAMs are expensive and consume large amounts of power. They do not scale well in throughput and power with classifier size. Also, TCAM, being a hardware based solution, does not offer flexibility for grouping the rules of a classifier. Such a flexibility is required in applications like Software Defined Networks where there is need to have a separate classifier for each network slice being controlled.

TCAM also provides the most straightforward hardware solution to single-match packet classification. However, its high cost and high power consumption led to extensive research into several alternative algorithmic solutions and some of them can be extended to report multi-match results. For example, Grid of Tries [8] is proposed to solve the two dimension classification problem. It can be extended for multiple fields with caching techniques. Other heuristic algorithms like Recursive Flow Classification (RFC) [9], HiCuts and HyperCuts [11], [10] work well for real world rule sets for single-match classification. However, due to memory inefficiency for multi-match classification with rules containing wildcards, these heuristic based solutions need pre-computation for building a decision tree and are slow to update.

B. Motivation

As stated above, packet classification has been considered extensively in literature. However, the case when both keys and rules have wildcard expressions as building blocks has not been well addressed. In fact, other than exhaustive search algorithms, there are no other methods with acceptable performance in this general case. Although for specific set of rules and keys it may be possible to modify some of previous schemes to make them compatible with special situation, it is not feasible to reconfigure the classifier every time one rule is added (deleted) to (from) the system. Specifically, it can be shown that packet classification in presence of wildcard expression cannot be generally done with complexity better than $O(N)$, where N is the number of rules. Following Theorem addresses this issue.

***Theorem** 1:* Consider the case when both rules and keys have unknown number of wildcard expression with unknown positions. It is impossible to design a classifier in general with complexity better than $O(N)$. In other words, without any constraint on the rules and keys, it is necessary to compare the key with all the rules.

Proof: Basically, the relation between a key and a rule containing wildcard expressions is NOT transitive. Putting it in words, in presence of wildcards, the matching relation between a key, say K, and a rule, say R_1, has no information, in general, about the relation between K and an other rule R_2, even if the relation of $R1$ and R_2 is known. So, in general, it is not possible to provide a procedure which compares the key with only a subset of rules. In other words, the complexity of the classifier should be at least $O(N)$.■

Inorder to make the Theorem 1 more clear, consider the following examples.

***Example** 1:* Assume that the headers have length of 2. Let $K = 1X$, $R_1 = X1$, and $R_2 = 01$. It is easy to see that K matches R_1, and R_1 matches R_2. However, K does not match R_2.■

***Example** 2:* Consider the case when key is purely wildcards. Then the classifier should return all the rules as possible matches. In other words, all the rules should be visited once.■

Based on Theorem 1, it seems that for an incoming key it is necessary to hit all the rules at least once. So, why should we consider this problem anymore, while the linear classifier already exists with complexity $O(N)$? The answer is that, it may be possible to improve the performance of the exhaustive search with a wise design that aggregates a set of manipulations into one straight forward operation of CPU (e,g. AND operation). Moreover, it is interesting to take the advantages of the wildcards, instead of being worried about their uncertain presence.

In this paper, we introduce a novel classifier which operates at least one order of magnitude faster than ordinary exhaustive one. In the following section. we will explain the proposed design and implementation.

The rest of the paper in organized as follows. Next section introduces the new technique. In section III, we will discuss the performance of the scheme analytically, and compare it with ordinary exhaustive search classifier. Moreover, the simulation results will be provided in this section. Conclusions and future work is in Section IV.

II. Proposed Scheme

In this section, the novel implementation is introduced. Firstly, we will make the whole idea more clear by providing the big picture of the solution. Next, the specific design to achieve a fast operation will be discussed.

A. The Big Picture

Assume that we have a key, K, and set of rules, $\mathbf{S}$, in the space of $\{0, 1, X\}^L$, where L is the number of bits in key/rule. The goal is to determine which rules match this key in every bit. In order to find this subset, we are going to divide it into a simpler problem. Specifically, let $\mathbf{S}_i$ be the set of rules that match in the i^{th} bit with K. Then, the desired subset would be the intersection of $\mathbf{S}_i$s, $\bigcap_1^L \mathbf{S}_i$. Note that if the i^{th} bit of K is X, then $\mathbf{S}_i$ would be the whole set of rules and so has no effect in the intersection operation. Figure 1 demonstrates this idea.

The key point and main contribution of this work, is how to find the set of rules according to each bit position in an organized and scalable manner. Moreover, a fast and efficient way to manipulate the intersection of these sets is introduced. In the next part, the ideas will become more clear.

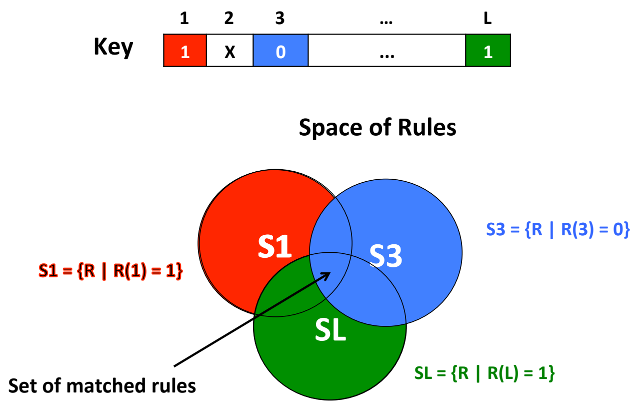

Fig. 1. The big picture for proposed packet classification scheme.

B. Implementation Specifications

Note that the main characteristic of the system that we care about is how fast a key can be classified. On the other hand, adding and deleting rules could be done with more freedom. So, it sounds reasonable to spend more time and organize the rules as they are added to system, so that later it can help for a faster classification. In fact, when a rule comes in, we will determine to which subsets it belongs.

Specifically, we have two $(N \times L)$-bit tables, Table of 1s, $T1$, and Table of 0s, $T0$, where N is the number of rules and L is the number of bits in the rule/key. Each column of these tables is assigned to one rule and is filed with a simple algorithm. Let the j^{th} column of the tables be assigned to R_j. Then, depending on the i^{th} bit of the R_j, $R_j(i)$, three different cases could happen:

- $R_j(i) = 0$: $T0(i,j) = 1$ and $T1(i,j) = 0$
- $R_j(i) = 1$: $T0(i,j) = 0$ and $T1(i,j) = 1$
- $R_j(i) = X$: $T0(i,j) = 1$ and $T1(i,j) = 1$

In fact, with this logic we have classified the rules based on their bits. For example, the first row of $T1$ shows all the rules that either have 1 or X in the first bit position. Figure 2 explains the process of adding rules as an example. This way, when the rules come in, we can incrementally change the tables with new columns. However, deleting the rules would be a little bit tricky as it can make holes in the table. Fortunately, this issue can be addressed by keeping track of empty columns as a meta data and fill the holes first, when a new rule comes in.

Now, when a key, K, is given we can find the set of matched rules by finding the intersection of $\mathbf{S}_i$s (defined above). In fact, depending on the i^{th} bit of the K, $K(i)$, we can extract the $\mathbf{S}_i$ from the tables:

- $K(i) = 0$: $\mathbf{S}_i$ is the i^{th} row of $T0$.
- $K(i) = 1$: $\mathbf{S}_i$ is the i^{th} row of $T1$.
- $K(i) = X$: $\mathbf{S}_i$ is the universal set (neutral in intersection).

The last part of the solution is an efficient way to find the intersections of $\mathbf{S}_i$s. Since these sets are saved as bits in tables,

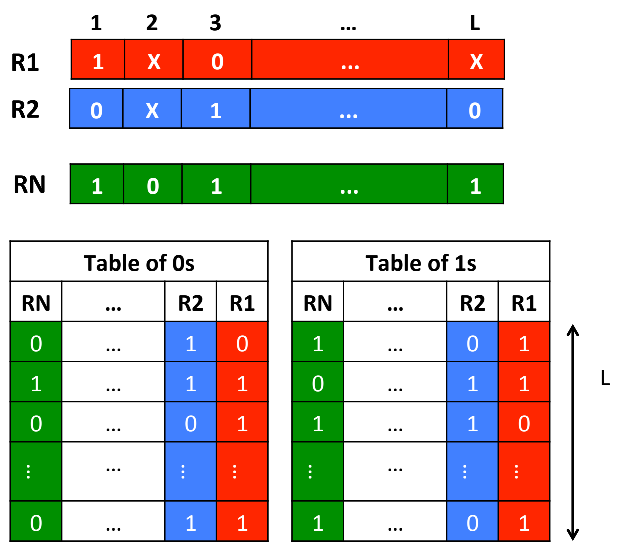

Fig. 2. An Example of adding rules to the tables.

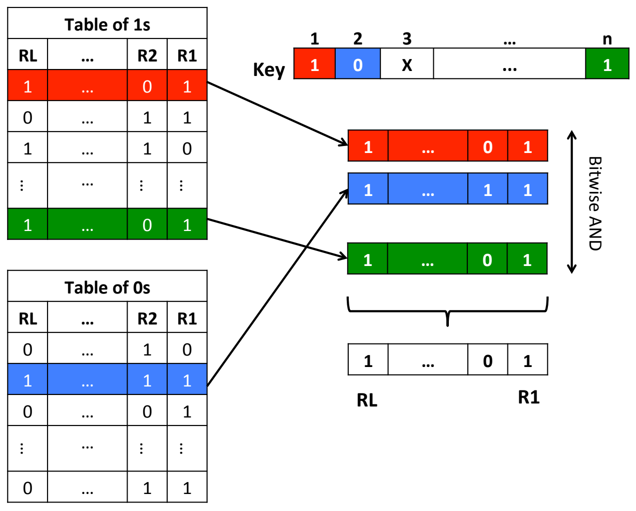

Fig. 3. Classifying a key based on the example in Figure 2. Here, it is assumed that $L < W$ such that all the bits in a row could be compact in one long integer for AND operation in CPU.

we can take the advantages of bitwise AND in the processor. Specifically, assume the processor can manipulate W-bit word length AND. Then we will group the columns of the table in sets of size W (e.g. for PC CPUs, $W = 64$ or 32). With this trick, we can decrease the number of operations with a factor of W. Figure 3 gives an example of this procedure.

Note that the wildcard expressions in a key has no effect on the process. Moreover, as soon as the the result of AND operation becomes 0 in a step, it is not necessary to consider the rest of the bits of the key. So, we can even improve the performance with considering the more important bit positions first (those with less wildcard expressions). For example, if the IP part of the header needs more precise discrimination, then it would be more likely that the rules does not match the key in this field. So, it is better to consider this field first in the comparisons.

III. Analysis and Simulations

In this section we will give a brief performance analysis of the proposed scheme and compare it with the most common exhaustive classifier, linear search.

First, let's take a look at the add/delete complexities. While deleting is $O(N)$ for both schemes, the adding would be $O(N)$ for our method, and $O(1)$ for linear classifier. Remember that the proposed algorithm looks for the holes (resulted from deletion) and fills them first. So, for each addition we may need to search over the tables to find the holes. However, another possible solution could be to keep track of holes in a meta data such that the searching time would become $O(1)$.

Now, it would be interesting to compare the methods based on the classification time. We have implemented both our method and the linear classifier in C++, and compared the classification times for various number of rules.The results are shown in Figure 4. Note that all rules are generated randomly from the space of $\{0, 1, X\}^L$. Moreover, the depicted result is averaged out over classification time for 1000 different random

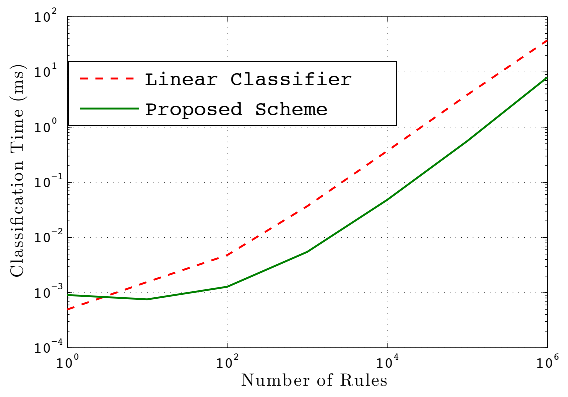

Fig. 4. The comparison of classification time for proposed scheme and linear search. All rules are created random. For each set of rules the experiment is performed for 1000 number of random keys and the average of completion time is calculated and plotted.

keys. As expected both of the curves increase linearly with the number of rules. However, our scheme is around one order of magnitude faster than the ordinary linear search.

It is also worth considering the memory usage of the scheme, since its success is highly dependent on the data tables. In order to store the tables we may need a $(2 \times N \times L)$-bit size of memory. Fortunately, this is not a very huge number. For example, for 1 million number of rules and 1Kb size of headers, the required memory would be around 2Gb, which is an ordinary number for nowadays RAMs

IV. Conclusions and Future Work

In this paper we provided a novel implementation for a software based classifier. It was shown that when both rules and keys contain wildcard expression, then it is not possible to provide a general solution with complexity better than $O(N)$. However, it is possible to improve the manipulation time and achieve the scalability with the proposed scheme. Simulation results show that the new technique classifies one order of magnitude faster than ordinary linear search.

One possible extension of this work would be to make the information in the tables more compact. In fact, the operations could become more efficient by squeezing and interleaving the rows of the table with intersection and/or union operations. Indeed, it may be possible to decrease the redundant information in the tables. Moreover, as stated in the paper, there could be priorities for more important bits (positions with less wildcards) such that the classification could find the differences faster and so terminate faster.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed," *Proc. OSDI (October 2010)*, 2010.

[3] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," *NSDI*, 2012.

[4] M. Faezipour and M. Nourani, "Wire-speed tcam-based architectures for multimatch packet classification," *Computers, IEEE Transactions on*, vol. 58, no. 1, pp. 5–17, 2009.

[5] F. Yu, T. Lakshman, M. Motoyama, and R. Katz, "Efficient multimatch packet classification for network security applications," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 10, pp. 1805–1816, 2006.

[6] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary cams," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 193–204.

[7] M. Nourani and M. Faezipour, "A single-cycle multi-match packet classification engine using tcams," in *High-Performance Interconnects, 14th IEEE Symposium on*. IEEE, 2006, pp. 73–80.

[8] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to cams?" in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1. IEEE, 2003, pp. 53–63.

[9] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 147–160, 1999.

[10] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 213–224.

[11] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *Micro, IEEE*, vol. 20, no. 1, pp. 34–41, 2000.