Hardware Assisted Packet Filtering Firewall

Shubhash Wasti
Department of Computer Science
University of Saskatchewan
57 Campus Drive
Saskatoon, SK S7N 5A9
Canada
email: shw320@cs.usask.ca

Supervisor: Ralph Deters

Abstract

A packet filter is a hardware or software mechanism that can be configured to select packets from a traffic stream based on some criteria. Many research works have been done in the past in the area of classifying packets based on one or more packet header fields. Those works are directed towards finding efficient algorithms and implementation architectures for high speed routing, Quality of Service (QoS), security enforcement through the use of firewalls, and other similar applications. This paper presents an overview of existing packet classification schemes, with particular attention to the ones that deal with classification based on multiple fields, are suitable for firewalls, and have possibility of efficient implementation using hardwares such as Content Addressable Memories (CAMs). With the availability of more flexible and faster CAMs in the past few years, their use in search applications has become more promising. Therefore, a major portion of this document also goes into discussing the possibilities of using CAMs for packet filtering.

1 Introduction

A firewall is a system designed to prevent unauthorized access to or from a private network. Firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets, although the use of firewalls to protect single PCs from the Internet is also growing, as more and more computer users become aware of security implications of the networked world. Firewalls can be implemented in both hardware and software, or a combination of both, and generally fall into one of the following categories: **packet filter**, **application gateways**, **circuit-level gateways**, and **proxy servers**. While they all have different levels of usefulness and drawbacks in different situations, this paper only deals with packet filters, as they are more likely to be efficiently implemented in hardware.

Filtering packets based on their header fields is a well known method of restricting access to network resources [Chapman, 1992, Corbridge et al., 1991]. Filtering requires the ability to classify packets according to specified filter rules. The rules can be viewed as logical functions on the packet header fields. Classification of packets also arises in other areas of computing, such as routing, policy based routing, differentiated Quality of Service, traffic billing, etc. [Gupta & McKeown, 1999]. However, not all of them use or require classification based on multiple fields in the packet header. For instance, a simple packet forwarding router only needs to classify packets based on one field (destination IP address) to perform the routing correctly. The concepts *filtering* and *classification* are tightly tied together, and so, these terms will be used rather loosely in this document, unless a distinction is necessary to avoid confusion.

The idea of packet filtering, or classification in general, was initiated in [Mogul et al., 1987] and later expanded by others [McCanne & Jacobson, 1993, Yuhara et al., 1994, Begel et al., 1999]. Although packet classification problem occurs in many areas of Computer Networks, the main focus of this paper is in the context of firewalls. Fast and effective classification of packets based on multiple header fields is a challenging problem. There are mainly two approaches to the implementation of filtering system: **interpreter-based** and **pattern-based**. The first approach consists of a set of instructions (compiled from the rule specification) and an interpreter engine to interpret the instructions (with the use of some form of processing power), while the latter approach matches a pattern using some comparison mechanism and does not require an interpreter engine. It is also possible to use both of those approaches together to achieve a balance between convenience and efficiency.

1.1 Filters and Rules

The criteria for classifying packets are called *filters*; in firewall terminology they are called *rules*. In this paper, these terms will be used interchangeably unless the distinction can improve clarity. A filter can be defined more formally as an ordered pair (F, A), where F is a boolean expression and A is the action to be taken if the expression is true.

The information relevant to classification of a packet is contained in the header of each packet. For instance, the relevant fields of an IPv4 packets could be the destination IP address (32 bits), the source IP address, the transport level protocol field (8 bits), the source port (16 bits), the destination port (16 bits), and the TCP flags (8 bits). The filter database, also known as *access list*, consists of a set of filters. Each filter requires the matching of a number of fields. There are two administrative approaches for using a firewall: **that which is not expressly forbidden is permitted** and **that what is not expressly permitted is forbidden**. If none of rules in the access list matches a packet, in the first approach the packet is accepted, whereas in the second approach the packet is rejected. The second approach is more conventional and secure, and so used by many existing firewalls as default.

[Molitor, 1995] describes a general architecture for packet filtering. He defines an access policy as *a collection of lines of text, each line describing a certain class of data (for example, TCP packets from host A to host B) and a reaction to take to such a data (e.g. drop it and log an access violation to host D)*. He also insists that the definition of *data* and the *reaction* should be left as open-ended as possible, so that a perfect architecture for implementing access policy would permit anything whatsoever to appear in those imaginary lines of text. He stresses more on flexibility than on the performance issues, giving the reason that *performance is in general adequate*, which unfortunately is not true any more.

The following is an example of a partial set of rules. These rules can be viewed as lines of text as described above or from a more abstract level, in which case, these are human-readable representations of logical expressions.

RULE	DIRECTION	TYPE	SRC ADDR	DEST ADDR	DEST PORT	ACTION
1	in	tcp	135.29.*	169.228.63.5	25	permit
2	out	tcp	*	128.245.9.29	any	deny
3	any	any	any	any	any	deny

One of the major shortcomings of packet filtering as an access policy paradigm is the stateless nature of it, i.e. the only information about the packet currently being examined can be used to make the filtering decision. However, applications are not in general stateless, and often one would like to be able to gather information and examine state at a level closer to that of the application.

More advanced packet filtering systems known as *dynamic packet filters* or *stateful packet filters* are also in existence, which offer state tracking and/or protocol checking. These systems provide the ability to do things which can not be done otherwise, but they also add complications such as exposing themselves to a number of denial of service attacks and address forging. A stateful packet filtering system allows to specify rules like the following [Zwicky et al., 2000, pp.169-70]:

```
Let the incoming UDP packets through only if they are responses to outgoing UDP packets already seen.
```

1.2 Filter Matching Problem

Packet filtering or classification problem, using multiple packet-header fields and allowing range matches, is difficult to implement at high speeds and with a large rule set [Lakshman & Stiliadis, 1998]. We need to consider mainly three types of matching of a field [Srinivasan et al., 1998]:

- **Exact match**: This type of matching requires exact matching of the filter field, i.e. there is only one value given in the filter for that field. Requiring the source port of the packet to be equal to `80` would be an exact match.
- **Prefix match**: It requires the filter field to be the prefix of the header filed of the incoming packet. If the rule is designed to block access from any source with the IP address `128.233.*.*`, then it requires a prefix match. A slight variation of prefix matching can be achieved by allowing arbitrary masking of the filter field, but this type of matching is rarely useful. The arbitrarily masked filters will be called *mask-based filters* in this paper.
- **Range match**: A range match requires the value of a header field to lie in a range specified by the filter. This is useful in specifying port number ranges like `1 – 1024`.

Both the range match filters and prefix match filters represent a collection of exact match filters, but they are not expressible in terms of each other. [Woo, 2000] points out that there is no direct mapping from range-based filters to mask-based filters and vice-versa. In other words, some range filters (e.g. 9 – 21) can not be expressed as a single equivalent mask-based filter and some mask-based filters (e.g. *1*1) can not be expressed as a single equivalent range filter. He further mentions that when a range-based filter is expressed in terms of mask-based filters, the number of mask-based filters ranges from 1 to $f - s + 1$, where s and f are the lowest and highest values in the range.

The problem of IP routing is a simplified version of a more general problem of packet classification based on multiple fields. The main difference between routing table lookup performed by a router and the filter matching done by other packet classification systems is usually in the number of rules and number of fields that they have to deal with. A router's routing table may consists of a large number of filters, whereas a packet filtering firewall would probably contain a much smaller rule database. Moreover, the routing decision of the router is usually based on only one field (destination IP address), which is not the case for more general packet classifiers such as a firewall.

The general packet classification problem can be viewed as a classical problem in computational geometry known as *point-location problem in multidimensional space* [Lakshman & Stiliadis, 1998]. The point-location problem is defined as follows: Given a point in a d-dimensional space, and a set of n d-dimensional objects, find the object that the point belongs to. Existing algorithms dealing with this problem are limited to the case of non overlapping objects. The best algorithms in this category are either an $O(\log^{d-1} n)$ time-complexity with $O(n)$ space-complexity, or an $O(\log n)$ time-complexity with $O(n^d)$ space. These algorithms measure well for a system with small number of rules, but become impractical for large number of rules with many dimensions [Lakshman & Stiliadis, 1998]. For the special case of two dimensions and non-overlapping rectangles, there exist some solutions that are much more efficient time and space-wise. However, these can not be applied to the general case of packet filtering problem simply because of the number of dimensions and overlapping rules.

A realistic packet filtering system contains more than one filter. When there are a number of filters, they have to be represented by some data structures in an efficient way so that the collection can be searched as quickly as possible. There is a delicate balance between the search speed and the amount of storage used. Various software based packet filters have their own algorithms to search the filters, and structures to store them in memory. Mostly, these structures are variations of PATRICIA [Morrison, 1968] and some form of trie [Clément et al., 1999] trees. [Woo, 2000] has noted that an efficient search requires a global optimization of the data structures, which can be computationally very expensive, and the suitability of a search algorithm can be highly dependent on the total number of filters.

When more than one filters are applicable to an IP packet, the packet classifier has to make a decision as to which filter to apply. Most packet filtering systems use the technique of filter prioritization. The highest priority filter is applied in the case of conflict. However, [Hari et al., 2000] claims that conflict resolution by prioritization does not always work. The authors of the paper model each filter by a node of a directed graph and show that the filter conflict problem can be formulated as a *cycle elimination problem* in a directed graph $\mathcal{C}$. They have shown that if the graph $\mathcal{C}$ contains a cycle, it is not possible to resolve the filter conflict by filter re-ordering. Since conflict resolution of rules is an entirely different issue and a difficult problem to solve in general, it will not be further discussed in this paper.

2 Design Issues of a Packet Filter

Although the design of a packet filtering system is primarily influenced by its use, more flexibility and efficiency are almost always desired. Efficient use of available resources and maintainability of the system are of high priority in all packet filtering systems. The following considerations are necessary to make robust and flexible packet filtering firewall [Feldmann & Muthukrishnam, 2000, Jayaram & Cytron, 1995]:

- **Resource limitations**: There should be a balance in the memory requirement and the time taken to classify the packet. While the goal should be to minimize the time required to classify each packet, it should not lead to an unreasonably high memory demand.
- **Number of rules to be supported**: The packet filter should not have arbitrary limits on number of rules that can be present in the system.
- **Number of fields (dimensions) used**: The packet filter should allow the user to specify any number of header fields for the purpose of packet classification. Arbitrary restrictions on the use of headers for classification reduces the flexibility and control that might be necessary for a desired level of protection.
- **Nature of rules**: The rules should allow all types of matches (exact, mask-based, and range-based matches) as far as possible.
- **Protocol independence**: The filtering system should be protocol independent, so that filtering is supported for different protocols and at different levels.
- **Specification language**: The filter specification language should be general and sufficiently expressive to specify various types of filters.
- **Fragmentation**: The packet filter should be able to handle packet fragmentation.
- **Scalability**: It should be scalable with the number of rules as well as the volume of traffic.
- **Efficient rule update**: The system should allow the addition and removal of rules with minimum disruption in the processing of packets.
- **Auditing**: The packet filter should have mechanisms to audit. In other words, it should be able to keep a log of all access attempts, both accepted and blocked, if required, as well as information that may be of statistical significance.
- **Rules prioritization**: In the case where a packet matches more than one rules, the packet filter should allow arbitrary priorities to be imposed on those rules, so that only one rule will be finally applicable.
- **Arbitrary field matching**: The packet filter should be able to match arbitrary fields, including that from link-layer, network layer, and transport layer. In some cases, even application layer headers might be of interest.
- **Maintainability**: The set of rules in a packet filtering system can be large and these rules might have been added to the list over a period of time, and possibly by more than one person. Any reordering of the rules within the list can completely alter the semantics of the rule set. Furthermore, there might be conflicting rules which can cause undesirable filtering behavior. This naturally makes the verification of the access list and its maintenance difficult and error-prone. If the packet filter system can provide easy and visual ways to specify the rules and if it can visually show the semantics and relationships of the rules in the access list, that would be a very highly desired feature.

Although the above list reflects what is desirable in a packet filtering system, not all existing packet filters meet all of the requirements. Some packet filters might focus more on some of the above aspects while ignoring other aspects that they consider less important or are too difficult to implement.

3 Current Packet Classification Schemes and Implementation Issues

3.1 Classification Algorithms

A packet filtering system requires a set of rules to operate properly. Each rule R specifies a *class* that a packet may belong to based on some criteria on the packet headers. An alternative way of viewing the rule R is *the set of all packets with headers that could match R* [Gupta & McKeown, 1999]. This set theoretic view of rule allows us to define some terms that are useful in discussing the rules. Two distinct rules are said to be either *partially overlapping*, *non-overlapping*, or *one is a subset of the other*. In many access lists, not all rules are disjoint. In such a case, the filters are prioritized according to their ordering or in some other ways, and the highest priority rule is said to have matched a packet.

The suitability of an algorithm to a packet filter depends on many factors such as the number of rules, number of dimensions, the target environment (e.g. hardware or software), expected volume of traffic, etc. Therefore, an algorithm developed for a particular situation may not be the best choice for another situation. Many research works in the past have focused on the single-field packet classification, used by many routers for packet forwarding [Filippi et al., 1999, Harbaum et al., 1998, McAuley & Francis, 1993, Gupta et al., 1998, Waldvogel et al., 1997].

While it is difficult or impossible to examine the algorithms and approaches used by many of the commercial implementations of packet filters, there are some algorithms and their implementations that are published in the literature, which can be used to explore packet filtering techniques. Not all of those algorithms were designed or optimized for hardware implementation. Some of the most popular packet classification algorithms are listed below with short descriptions:

1. **Sequential matching**: For each arriving packet, each rule is evaluated sequentially until a rule is found that matches all the packet headers that are of concern. Despite its simplicity and efficient use of memory, it has poor scaling properties since the time to perform classification grows linearly with the number of rules.

2. **Hardware algorithms using ternary CAMs**: Ternary CAMs (content addressable memories) can store words with three-valued-digits: ‘0’, ‘1’ and ’*’ (wildcard). Each rule is represented by one or more entries in the CAM. Since the CAM does not support some operators such as *greater than* or *less than*, some rules need to be broken down into multiple entries in the CAM to be represented by the available three-digit based patterns. The CAMs also have built in priority mechanism so that the entries with lower address values have higher priority in the case of multiple matches. The comparison is performed in parallel, which makes it an attractive alternative to the linear search algorithm.

3. **Grid of tries**: [Srinivasan et al., 1998] proposes a scheme in which the trie data structure is extended to two fields. This provides a good solution if there are only two fields, however, for more than two fields, it does not extend very well. They also describe another more general solution called *‘Crossproducting’*, which can handle more than two fields. According to their experiment, it uses 1.5 MB of memory for 50 rules. For more rules, they suggest a form of caching technique.

4. **Bit-parallelism in hardware**: [Lakshman & Stiliadis, 1998] describes a scheme optimized for hardware implementation, which employs bit-level parallelism to match multiple fields concurrently. The authors implemented five dimensional packet classification in a high speed router prototype using a FPGA device and five 1M-bit SRAMs. They achieved a classification rate of 1 million packets per second (in the worst case) for 512 rules. Two variations of the algorithms have been suggested, one is space efficient while other is time efficient.

5. **RFC classification**: [Gupta & McKeown, 1999] points out that in reality the packet filtering rule sets contain a considerable structure and redundancy that can be exploited by the classification algorithms. They estimate that a multi-stage algorithm, called RFC (recursive flow classification) can be used to classify 30 million packets per second in pipelined hardware, or one million packets per second in software.

This paper focuses primarily on the hardware implementation of the packet filter, and so, will not go into depth of all of the algorithms listed above.

3.2 Existing Packet Filters

There are a number of packet filters currently available, both commercial and non-commercial. Due to the lack of knowledge about the underlying implementation of commercial packet filters, only the publicly available packet filtering systems are discussed below:

1. **CSPF (CMU/Stanford Packet Filter)**: CSPF is an interpreter-based filtering mechanism. The filter specification language is stack-based and contains binary operations which are used to evaluate boolean operations. The filter programs evaluate a logical expression on the fields of the header of the received packet, accepting packets based on the result. The expressions to be evaluated is structured as a tree, with internal nodes performing boolean operations (OR/AND) and leaf nodes performing comparisons. Some of the shortcoming of CSPF are redundant computations caused by the tree model of evaluation, lack of indirection operator in the specification language (which forces to deal with only fixed length fields), and lack of support for composition of filters [Jayaram & Cytron, 1995].
2. **BPF (BSD Packet Filter)**: BPF is also an interpreter-based filtering mechanism, however, the filter specification language is register-based (instead of stack-based). It provides support for handling varying length fields. Furthermore, it uses DAGs as model to evaluate filter expressions, avoiding redundant computations inherent in CSPF.
3. **MPF (Mach Packet Filter)**: MPF extends the BPF filter specification language by adding instructions for handling filter composition and fragmentation. The main focus of MPF is on demultiplexing packets to multiple endpoints efficiently by composing filters and on handling fragmented packets. MPF supports only one specific case of filter composition, namely, the case where the filters share an identical prefix and followed by a variation at a single point in the header.
4. **PATHFINDER**: PATHFINDER [Bailey et al., 1994] is a pattern-based filtering mechanism which allows more general composition of filters with common prefixes than MPF. It is suitable for both software and hardware implementations.
5. **BPF+**: BPF+ extends BPF by adding powerful optimization techniques. It includes a filter program translator, a byte code optimizer, a byte code safety verifier, and a just in time assembler to convert byte codes to efficient native code [Begel et al., 1999].
6. **DPF (Dynamic Packet Filters)**: DPF uses the declarative packet-filter language that is aggressively optimized using dynamic code generation. Dynamic code generation is the creation of executable code at run time. The basis of DPF is also filter interpretation, as is the case for many other filters including BPF and MPF. The authors of DPF claim that the performance of DPF compares very closely to in-kernel routines while retaining all the flexibility of the packet filter model [Engler & Kaashoek, 1996].

3.3 Problems in Packet Filter Implementations

A number of difficulties arise in the design and implementation of a secure and efficient packet filtering firewall. Some of the problems that need to be addressed are listed below:

- Although packet filtering systems generally do not depend on the source IP address, there are some which depend on it (especially the stateful packet filters). As the source IP address can be spoofed, a packet filter may wrongly classify a packet. Filtering based on source port faces similar problem, as the source machine might be running an unsuspected client or server on that port.
- Since the IP packet header can have variable length (due to the *options* field), it can be difficult to locate the higher level protocol information, such as TCP/UDP headers, when using simple offset-based pattern matching techniques.
- A packet may be fragmented to allow it to pass through a network that only supports smaller frame size. When a packet is fragmented, only the first fragment contains the higher level protocol headers from the original packet,

which may be necessary to make filtering decisions for each of the fragments. Some packet filters just drop the first fragment hoping that the other fragments will be useless to the receiver. However, this assumption can be dangerous, since hackers have found ways to fool the system through *tiny fragment attacks* and *overlapping fragment attacks* [Ziemba et al., 1995]. Another possibility is to keep a cache of recently seen packets and simulate the reassembly process [Corbridge et al., 1991], however, it can significantly complicate the packet filtering process.

- Many filter engines come with predefined header fields that can be used in packet filtering. This has severe impact on flexibility. Unless the administrator can specify precisely which header fields are to be used in decision making, the desired security policy can not be effectively implemented. For instance, one may wish to block packets with TCP "SYN" flag set, but the packet filter may not allow this field to be used for filter specification.

4 Hardware Implementation of Packet Filters

1. PATHFINDER [Bailey et al., 1994] is one of the early attempts of hardware implementation of packet filters. It uses an offset based pattern matching algorithm and can be implemented in either hardware or software. A filter is represented by a *line*, which is a collection of *cells*. Each cell is given by a tuple consisting of *offset, length, mask,* and *value*. If all the cells in a line match a packet, then the line is said to match the packet. Intuitively, a line attempts to match a single protocol header; each cell matches a different field in that header.

   For the hardware prototype implementation, they have used Altera FPGAs, but suggest that the use of custom VLSI chips can increase both the number of cells and the speed. In their implementation, each cell matches a single value, but they mention that range matching can be implemented by using two comparators. The authors report that their hardware implementation can keep up with the OC-12 (622 Mbps) link. A detailed description of PATHFINDER and its hardware/software implementation can be found in [Bailey et al., 1994].

2. [Woo, 2000] suggests a modular approach to packet classification which can be implemented in either hardware, software or a combination of both. The idea behind the algorithm is divide-and-conquer, which operates first by decomposing large filter table into smaller *filter buckets* of a fixed maximum size (from 8 to 128), and processing filter buckets of limited size to find the match. It can optimize the search data structure by taking into account the relative usage of the individual filters in a filter table. In an abstract level, this approach divides the search space into three layers: *index jump table*, *search tree,* and *filter buckets*. The jump table records the location of each group of filters (represented by trees) which have been categorized using some initial prefixes of selected dimensions. The search trees are 2^m-ary trees, which are constructed by examining m bits of the filters at a time, and dividing them into 2^m groups. The filter buckets consist of the set of filters left at the leaf nodes when the division process terminates. In other words, a filter bucket contains the filters that are not distinguished further. The search of the data structure is performed by first "jumping" to a specific subtree via the jump table followed by a sequence of branching over the trees and finally landing in a filter bucket. Then the bucket is searched using a linear search, a binary search, or a CAM-assisted search. The author mentions that a CAM can be used to store each dimension (prepended with the filter bucket ID) of a filter in a filter bucket. Each dimension can be concurrently searched and the results can be combined in a parallel step to obtain a match.

3. Lakshman and Stiliadis [Lakshman & Stiliadis, 1998] have suggested an algorithm based on bit-parallelism. Due to its use of very simple operations, it it can be implemented in hardware quite well. It only performs integer comparison in binary search and parallel AND operation. The authors believe that despite the linear time complexity of the algorithm, the use of bit-parallelism significantly accelerates the filtering operation. The algorithm works in the following way:

   A set of n packet filtering rules in k dimension are defined. Let's denote the set of ranges that define rule r_m in the k dimension by $\{e_{1m}, e_{2m}, \ldots, e_{km}\}$. There are two stages of the algorithm:

   - **Filter setup stage**: For each dimension j (where $1 \leq j \leq k$), project all intervals e_{ji} (where $1 \leq i \leq n$) on the j-axis, by extracting the j^{th} element of every filter rule for all n filter rules. There are a maximum of

$2n + 1$ non-overlapping intervals that are created on each axis. Let's denote the k sets of these intervals by P_j (where $1 \leq j \leq k$).

For each interval $i \in P_j$, for all j in $\{1, 2 \ldots, k\}$, create sets of rules R_{ij} (where $1 \leq i \leq 2n+1, 1 \leq j \leq k$), such that a rule r_m belongs in set R_{ij} if and only if the corresponding interval i overlaps with e_{jm} in the j^{th} dimension, i.e. if and only if $i \subseteq e_{jm}$ where e_{jm} is the j^{th} element of rule r_m.

Without loss of generality, it can be assumed that the rules are sorted based on their priorities, i.e. higher priority rule comes first. Let's suppose that a packet with fields $E_1, E_2, \ldots, E_k$ arrives in the system.

- **Packet filtering stage**: For each dimension j, find the interval i_j on set P_j that E_j belongs to. The authors suggest using binary search or any other efficient algorithm for this purpose. After the intervals are found, create the intersection of all sets $R_{i_j j}$ (where $i_j \in \{1, 2, \ldots, 2n + 1\}$ by ANDing the corresponding bit-vectors in the bit arrays associated with each dimension and determine the highest priority entry in the resultant bit vector. The rule corresponding to the highest priority is applied to the arriving packet.

4. [McAuley & Francis, 1993] describes the use of CAMs for fast routing table lookup techniques, where the table is composed of hierarchical IP addresses. Hierarchical addresses are the addresses that are arranged in the fashion similar to the national telephone network. The hierarchies are created based on the initial prefixes of the IP address. In the fixed-position hierarchical addresses, a fixed number of bits are used to create the hierarchies whereas in the variable-position hierarchical addresses, a variable number of bits (not necessarily prefix) are used to create the hierarchies. This scheme can be viewed as the use of masks to group similar IP addresses. Although the paper primarily deals with the case of packet forwarding by routers and switches, the idea can possibly be extended to more general packet filtering scenarios such as firewalls.

5 Use of CAMs and Associated Problems

A number of devices known as *communications processors* are in use in the networking community for classifying network traffic and enforcing policy functions. The programmability of those relatively new pieces of hardware, consisting of components such as CAMs and microprocessors, ranks them superior to more traditional implementations of traffic classification based on ASICs [Walsh, 1999]. Although use of such communication processors to build a firewall seems attractive, their use in building low cost devices such as network card with built in firewall, seems unlikely due to the complexity and cost.

Another possibility for building a packet filter is to make use of CAMs. Memory applications involving searching, have previously been implemented in programmable logic devices (PLDs) using random access memories (RAM). Searching for an item in RAM can take many clock cycles depending on the depth of the RAM block. Unlike RAM-based table-lookup implementations, CAMs can perform table lookups at much higher speed, as many as 66 million searches per second [Paul, 2000]. Ternary CAMs are typically used in data communications applications such as carrier class edge and enterprise routers, core Internet Gigabit and Terabit routers, SONET systems, and ATM switching equipments. A packet filtering firewall typically has to perform a large number of table lookups for matching the rules. So, we can expect that these CAM devices will find extensive use in building firewalls. Considering the transformation of CAM technology from binary CAMs to ternary CAMs and now ternary DRAM CAMs, we can predict that these are going to be powerful tools for the search oriented applications. Kawasaki LSI has listed "Classification CAMs", "Longest Match Engines", "Gigabit CAMs", and other variations, in their product catalogue (*http://www.klsi.com/products/CAM.htm*). Although, some of these are general purpose CAMs, others are specialized for more specific tasks such as longest prefix matching. It should only be a matter of time before some specialized CAMs appear in the market which can perform multidimensional pattern matching with some support for arithmetic comparison (as required by range matching). In the present state, however, the use of CAMs requires extra hardware and logic. Specifically, the following issues need to be addressed when implementing a packet filtering firewall using currently available CAMs:

- Currently available ternary CAMs can only perform pattern based comparison, with the support for arbitrary bit-masking. As they lack the numerical comparison, the range matching required by some of the packet header fields (such as port numbers) can not be performed directly. As mentioned in section 1.2, there is no one to one mapping of ranges to masks. This leads to the situation where all the rules have to be pattern based. Conversion from range based rules to pattern based rules can significantly increase the number of entries in the table, leading to higher memory requirements, and adds conversion overhead.

- Another problem that requires attention is that the CAMs can not perform any logical operations when matching a pattern. i.e. when multiple fields of a packet header are to be matched, one CAM can only find a match for one field, which means different fields have to be matched using different CAMs and combined in a separate step. The combination stage can be tricky, because these partial matches should be further checked to make sure that only the ones that are part of the same rule are combined.
- A rule may require positive matching or negative matching. i.e. a rule may specify that a particular header filed should not match a given value, rather than saying that it should match that value. In this case, the header field needs to be checked against the complement of the value. While finding the complement value of a range can be relatively easy, it can be complicated for the case of IP addresses. For instance, if the destination address should not match `128.245.65.*`, there is no easy way to specify one or two IP addresses that would be complement of this. It is certainly possible to negate the CAM match flag if the CAM contains all negative matching fields. However, that doubles the number of required CAMs, since now we have to take care of negative as well as positive matching.
- A functional firewall should not have to reload the complete set of rules merely to adjust to some transient requirements, such as adding a rule or removing a rule from the filter table, as that would make it very inefficient. Addition of a rule to the table is straightforward process, since it can be added to the next available location, however, deletion can be a problem. Since the rules have different priorities depending on which location they are, reordering of rules is not allowed. This means deleting a rule would require either shifting up all the rules below the deleted rule, or tagging the deleted rule as deleted and leaving it there. While the first approach is unacceptable due to the large overhead, the second approach leads to inefficient memory use.

6 Conclusion and Future Work

This paper presented an overview of various packet classification schemes and their use in building packet filtering firewalls. Our main goal was to explore the possibilities of implementing packet filtering firewall using hardware devices. However, without enough knowledge about the existing software based techniques, it would be impractical to dive into hardware solutions. Therefore, software based techniques and hardware based techniques are intermingled almost throughout this paper. We have identified some of the problems associated with the attempt of building firewalls using CAM memory devices. Some of those issues might be resolved by more advanced and flexible CAMs that might be available in future, while others require further investigation to determine their severity and solutions.

A natural extension to this paper can be an in-depth exploration of the possibility of the use of CAMs in building packet filters and their effect on the performance, cost, and flexibility. We plan to find systematic resolutions to the issues pointed out in the previous section. The CAMs can be simulated by softwares and the effectiveness of various architectures and algorithms can be studied. After a reasonable architecture and methodology is developed, a real hardware prototype can be built and experiments can be performed.

References

[Bailey et al., 1994] Bailey, M. L., Gopal, B., Pagels, M. A., Peterson, L. L., & Sarkar, P. (1994). PATHFINDER: A pattern-based packet classifier. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[Begel et al., 1999] Begel, A., McCanne, S., & Graham, S. L. (1999). BPF+: Exploiting global data-flow optimization in a generalized packet filter architecture. In *Proceedings of ACM SIGCOMM'99*.

[Chapman, 1992] Chapman, D. B. (1992). Network (in)security through IP packet filtering. In *Proceedings of the Third USENIX UNIX Security Symposium*.

[Clément et al., 1999] Clément, J., Flajolet, P., & Vallée, B. (1999). Dynamical sources in information theory : A general analysis of trie structures. Tech. rep., INRIA - Institut National de Recherche en Informatique et en Automatique.

[Corbridge et al., 1991] Corbridge, B., Henig, R., & Slater, C. (1991). Packet filtering in an IP router. In *Proceedings of the fifth Large Installation Systems Administration Conference, San Diego, California, USA*.

[Engler & Kaashoek, 1996] Engler, D. & Kaashoek, M. F. (1996). DPF: Fast, flexible message demultiplexing using dynamic code generation. In *Proceedings of the ACM SIGCOMM'96*.

[Feldmann & Muthukrishnam, 2000] Feldmann, A. & Muthukrishnam, S. (2000). Tradeoffs for packet classification. In *Proceedings of IEEE INFOCOMM 2000*.

[Filippi et al., 1999] Filippi, E., Innocenti, V., & Vercellone, V. (1999). Address lookup solutions for gigabit switch/router. *CSELT Technical Reports*, 27 (3).

[Gupta et al., 1998] Gupta, P., Lin, S., & McKeown, N. (1998). Routing lookups in hardware at memory access speeds. In *Proceedings of the IEEE INFOCOMM'98*.

[Gupta & McKeown, 1999] Gupta, P. & McKeown, N. (1999). Packet classification on mutiple fields. In *Proceedings of ACM SIGCOMM'99*.

[Harbaum et al., 1998] Harbaum, T., Meier, D., Zitterbart, M., & Brøkelmann, D. (1998). Hardware assist for IPv6 routing table lookup. In *International Symposium on Broadband European Networks (SYBEN)*.

[Hari et al., 2000] Hari, A., Suri, S., & Parulkar, G. (2000). Detecting and resolving packet filter conflicts. In *Proceedings of IEEE INFOCOMM 2000*.

[Jayaram & Cytron, 1995] Jayaram, M. & Cytron, R. K. (1995). Efficient demultiplexing of network packets by automatic parsing. Tech. Rep. WUCS-95-21, Washington University, Department of Computer Science, One Brookings Drive, St. Louise, MO.

[Lakshman & Stiliadis, 1998] Lakshman, T. & Stiliadis, D. (1998). High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of ACM SIGCOMM'98*.

[McAuley & Francis, 1993] McAuley, A. J. & Francis, P. (1993). Fast routing table lookup using CAMs. In *Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking*.

[McCanne & Jacobson, 1993] McCanne, S. & Jacobson, V. (1993). The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the Winter 1993 USENIX Conference*.

[Mogul et al., 1987] Mogul, J. C., Rashid, R. F., & Accetta, M. J. (1987). The packet filter: An efficient mechanism for user-level network code. Tech. rep., DEC Western Research Laboratory, Palo Alto, CA.

[Molitor, 1995] Molitor, A. (1995). An architecture for advanced packet filtering. In *Proceedings of the fifth USENIX UNIX Security Symposium*.

[Morrison, 1968] Morrison, D. R. (1968). PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the Association of Computing Machinary*, 15 (4):514–534.

[Paul, 2000] Paul, D. (2000). Ternary DRAM CAM: Now and the future. ElectronicNews Online. (URL checked: march 30, 2001).
URL `http://www.electronicnews.com/enews/Issue/RegisteredIssues/2000/05012000/z26f-2.asp`

[Srinivasan et al., 1998] Srinivasan, V., Varghese, G., Suri, S., & Waldvoge, M. (1998). Fast and scalable layer four switching. In *Proceedings of ACM SIGCOMM'98*.

[Waldvogel et al., 1997] Waldvogel, M., Varghese, G., Turner, J., & Plattner, B. (1997). Scalable high speed IP routing lookups. In *Proceedings of ACM SIGCOMM'97*.

[Walsh, 1999] Walsh, P. (1999). Communication processors: A buyer's guide. (URL checked: march 30, 2001).
URL `http://www.csdmag.com/main/1999/09/9909feat4.htm`

[Woo, 2000] Woo, T. Y. C. (2000). A modular approach to packet classification: Algorithms and results. In *Proceedings of IEEE INFOCOMM 2000*.

[Yuhara et al., 1994] Yuhara, M., Maeda, C., Bershad, B. N., & Moss, J. E. B. (1994). Efficient packet demultiplexing for multiple endpoints and large messages. In *Proceedings of the Winter 1994 USENIX Conference*.

[Ziemba et al., 1995] Ziemba, G., Reed, D., & Traina, P. (1995). Security considerations for IP fragment filtering. Request for Comments, RFC-1858.
URL `ftp://ftp.isi.edu/in-notes/rfc1858.txt`

[Zwicky et al., 2000] Zwicky, E. D., Cooper, S., & Chapman, D. B. (2000). *Building Internet Firewalls*. O'Reilly & Associates, Sebastopol, CA, USA, 2nd edn.