Report Number: WUCSE-2005-9

2005-03-03

On using content addressable memory for packet classiﬁcation

David E. Taylor and Edward W. Spitznagel

Packet switched networks such as the Internet require packet classification at every hop in order to ap-ply services and security policies to traffic flows. The relentless increase in link speeds and traffic volume imposes astringent constraints on packet classification solutions. Ternary Content Addressable Memory (TCAM) devices are favored by most network component and equipment vendors due to the fast and de-terministic lookup performance afforded by their use of massive parallelism. While able to keep up with high speed links, TCAMs suffer from exorbitant power consumption, poor scalability to longer search keys and larger filter sets, and inefficient support of... Read complete abstract on page 2.

On using content addressable memory for packet classification

David E. Taylor and Edward W. Spitznagel

Complete Abstract:

Packet switched networks such as the Internet require packet classification at every hop in order to ap-ply services and security policies to traffic flows. The relentless increase in link speeds and traffic volume imposes astringent constraints on packet classification solutions. Ternary Content Addressable Memory (TCAM) devices are favored by most network component and equipment vendors due to the fast and deterministic lookup performance afforded by their use of massive parallelism. While able to keep up with high speed links, TCAMs suffer from exorbitant power consumption, poor scalability to longer search keys and larger filter sets, and inefficient support of multiple matches. The research community has responded with algorithms that seek to meet the lookup rate constraint with greater efficiency through the use of com-modity Random Access Memory (RAM) technology. The most promising algorithms efficiently achieve high lookup rates by leveraging the statistical structure of real filter sets. Due to their dependence on filter set characteristics, it is difficult to provision processing and memory resources for implementations that support a wide variety of filter sets. We show how several algorithmic advances may be leveraged to im-prove the efficiency, scalability, incremental update and multiple match performance of CAM-based packet classification techniques without degrading the lookup performance. Our approach, Label Encoded Content Addressable Memory (LECAM), represents a hybrid technique that utilizes decomposition, label encoding, and a novel Content Addressable Memory (CAM) architecture. By reducing the number of implementation parameters, LECAM provides a vehicle to carry several of the recent algorithmic advances into practice. We provide a thorough overview of CAM technologies and packet classification algorithms, along with a detailed discussion of the scaling issues that arise with longer search keys and larger filter sets. We also provide a comparative analysis of LECAM and standard TCAM using a collection of real and synthetic filter sets of various sizes and compositions.

2005-9

On using content addressable memory for packet classification

Authors: Taylor, David E.; Spitznagel, Edward W.

March 3, 2005

Abstract: Packet switched networks such as the Internet require packet classification at every hop in order to apply services and security policies to traffic flows. The relentless increase in link speeds and traffic volume imposes astringent constraints on packet classification solutions. Ternary Content Addressable Memory (TCAM) devices are favored by most network component and equipment vendors due to the fast and deterministic lookup performance afforded by their use of massive parallelism. While able to keep up with high speed links, TCAMs suffer from exorbitant power consumption, poor scalability to longer search keys and larger filter sets, and inefficient support of multiple matches. The research community has responded with algorithms that seek to meet the lookup rate constraint with greater efficiency through the use of commodity Random Access Memory (RAM) technology. The most promising algorithms efficiently achieve high lookup rates by leveraging the statistical structure of real filter sets. Due to their dependence on filter set characteristics, it is difficult to provision processing and memory resources for implementations that support a wide variety of filter sets. We show how several algorithmic advances may be leveraged to improve the efficiency, scalability, incremental update and multiple match performance of CAM-based packet classification techniques without degrading the lookup performance. Our approach, Label Encoded Content Addressable Memory (LECAM), represents a hybrid technique that utilizes decomposition, label encoding, and a novel Content Addressable Memory (CAM) architecture. By reducing the number of implementation parameters

On using content addressable memory for packet classification

David E. Taylor, Edward W. Spitznagel

WUCSE-2005-9

3 March 2005

Applied Research Laboratory
Department of Computer Science and Engineering
Washington University in Saint Louis
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130
det3@arl.wustl.edu

Abstract

Packet switched networks such as the Internet require packet classification at every hop in order to apply services and security policies to traffic flows. The relentless increase in link speeds and traffic volume imposes astringent constraints on packet classification solutions. Ternary Content Addressable Memory (TCAM) devices are favored by most network component and equipment vendors due to the fast and deterministic lookup performance afforded by their use of massive parallelism. While able to keep up with high speed links, TCAMs suffer from exorbitant power consumption, poor scalability to longer search keys and larger filter sets, and inefficient support of multiple matches. The research community has responded with algorithms that seek to meet the lookup rate constraint with greater efficiency through the use of commodity Random Access Memory (RAM) technology. The most promising algorithms efficiently achieve high lookup rates by leveraging the statistical structure of real filter sets. Due to their dependence on filter set characteristics, it is difficult to provision processing and memory resources for implementations that support a wide variety of filter sets. We show how several algorithmic advances may be leveraged to improve the efficiency, scalability, incremental update and multiple match performance of CAM-based packet classification techniques without degrading the lookup performance. Our approach, *Label Encoded Content Addressable Memory* (*LECAM*), represents a hybrid technique that utilizes decomposition, label encoding, and a novel Content Addressable Memory (CAM) architecture. By reducing the number of implementation parameters, *LECAM* provides a vehicle to carry several of the recent algorithmic advances into practice. We provide a thorough overview of CAM technologies and packet classification algorithms, along with a detailed discussion of the scaling issues that arise with longer search keys and larger filter sets. We also provide a comparative analysis of *LECAM* and standard *TCAM* using a collection of real and synthetic filter sets of various sizes and compositions.

Table 1: Example filter set.

SA	*DA*	*Prot*	*DP*	*PT*	*Action*
*	1101 0100	[0:15]	TCP	9	deny
1010 1101	1101 0100	3	UDP	1	QID 3
1101 0100	1010 1101	3	UDP	1	QID 9
1101 01*	1010 11*	[0:15]	*	2†	decrypt
1010 11*	1101 01*	[0:15]	*	2†	encrypt
1101*	1010 11*	[4:15]	TCP	5	QID 4
0*	1010 11*	[4:15]	TCP	5	QID 5
0*	*	[0:15]	*	8	QID 11
*	*	[0:15]	*	10	deny

1 Introduction

Packet classification is the process by which a router identifies the flow or set of flows to which a given packet belongs. It is a fundamental datapath task that enables a multitude of security, monitoring, and Quality of Service applications. Filters define a flow or set of flows by specifying match conditions on packet header fields. Typically, filters define match conditions on the packet header fields comprising the IP 5-tuple: prefixes on the source and destination IP addresses, an exact match or wildcard on the transport protocol, and arbitrary ranges on the source and destination transport port numbers. Routers resolve the flow for a given packet by searching the set of filters for the subset of matching filters.

An example filter table is shown in Table 1. In this simple example, filters specify match conditions on four packet headers fields: prefixes of 8-bit source and destination addresses, exact match or wildcard on the transport protocol, and a range for a 4-bit destination port number. Note that the filters in Table 1 also contain an explicit priority tag *PT* and a non-exclusive flag denoted by †. These additional values provide ease of maintenance and a supportive platform for a wider variety of applications. Priority tags allow filter priority to be independent of filter ordering. Packets may match only one exclusive filter, allowing Quality of Service and security applications to specify a single action for the packet. Packets may also match several non-exclusive filters, providing support for transparent monitoring and usage-based accounting applications. Note that a parameter may control the number of non-exclusive filters, r, returned by the packet classifier. Like exclusive filters, the priority tag is used to select the r highest priority non-exclusive filters.

Consider an example search using the following packet header fields: 1101 0100 (IP source address), 1010 1101 (IP destination address), 3 (destination port), UDP (protocol). The third, fourth, and last filters match the packet. The third filter is the highest priority exclusive filter that matches, and the fourth filter is the highest priority (and only) non-exclusive filter that matches. The packet would be processed by the decrypt application and placed in the queue with Queue Identifier (QID) 9.

Fast and scalable solutions are necessary to prevent packet classification from becoming a performance bottleneck in the router datapath. Due to the complexity of the multi-field search problem and the inefficiencies of existing solutions, packet classification remains an important and active area of research. While a number of clever algorithms have been developed, they have yet to supersede Content Addressable Memory (CAM) technologies as the prominent solution in use. Current generation router line cards typically employ one or several Ternary Content Addressable Memory (TCAM) devices, depending on the size of the filter sets they support. The commonly cited benefit of TCAM technology is fast, deterministic lookup times. TCAMs perform a massively parallel search over every entry. The tradeoffs for providing “single cycle” lookups are poor scalability, area inefficiency, and exorbitant power consumption. Additionally, there are a variety of implementation constraints that reduce the achievable lookup rate. Searches on long search keys require several clock cycles in order to transmit the search key across the fixed-width data bus (typically 72 bits wide). Similarly, TCAMs do not efficiently support multiple matches. For example, in multiple-match mode a current TCAM requires approximately 12 clock cycles per match (i.e. 36 clock cycles to return 3 matching filters). Clearly the multiple match performance of TCAMs could be improved with additional hardware and engineering effort, but transmitting multiple results across the output interface require several cycles. We provide a more detailed discussion of TCAMs, their inefficiencies and scalability limits in Section 2.1.

While TCAMs can be engineered to provide single cycle lookups, bandwidth limited I/O interfaces place a fundamental limit on the amount of key data that a device can receive and matching filter data that a device can transmit. Clearly, there is an opportunity to trade off single cycle lookups in order to achieve greater scalability and efficiency without degrading the achievable lookup performance. In response, we introduce Label Encoded Content Addressable Memory (LECAM), a novel packet classification technique that blends algorithmic techniques with a modified CAM architecture. Using this hybrid approach to the problem, LECAM can match TCAM lookup rates, reduce the hardware requirements, and efficiently scale to large filter sets and filters classifying on fields beyond the standard 5-tuple. As discussed in Section 2, we leverage the decomposition and label encoding techniques employed in the Distributed Crossproducting of Field Labels (DCFL) algorithm [1]. The label encoding technique leverages the redundancies observed in real filter sets to more efficiently represent the set of stored filters. Similar to other decomposition techniques, DCFL uses independent search engines for each filter field, where the search engines are optimized for the type of match condition. The results from the search engines are aggregated in a distributed fashion using efficient set membership data structures. While providing an efficient and highly scalable solution, DCFL presents several implementation challenges due to the number of design parameters. By aggregating the search engine results with a modified CAM architecture, LECAM provides a higher performance and conceptually simpler aggregation mechanism. Section 3 presents a detailed description of the LECAM technique, focusing on the novel CAM architecture.

Using a collection of real and synthetic filter sets of varying size and composition, we analyze the performance and resource requirements of LECAM in Section 6. The real filter sets were provided by research colleagues, ISPs, and network equipment vendors. Synthetic filter sets that model the real filter sets were generated using the *ClassBench* tool suite [2]. We focus our performance evaluation of LECAM on the size of the aggregation CAM relative to a standard TCAM. Since power consumption scales approximately linearly with the size of a CAM, a reduction in area corresponds to a proportional reduction in power consumption. In addition to the relative CAM area, Section 6 also reports the area consumed by the RAM required for the field search engines. A discussion of the implementation considerations for LECAM is provided in Section 8. We conclude with an overview of algorithmic alternatives and a discussion of the deployment challenges for packet classification algorithms in Section 9.

2 Background & Related Work

Due to the importance and complexity of the problem, packet classification has been a vibrant area of computer communications research. We view the body of work as proceeding along two major threads: algorithmic and architectural. A few pioneering groups of researchers posed the problem, provided complexity bounds, and offered a collection of algorithmic solutions [3, 4, 5, 6]. Subsequently, the design space has been thoroughly explored by many offering new algorithms and improvements upon existing algorithms [7, 8, 9]. Given the inability of early algorithms to meet performance constraints imposed by high speed links, researchers in industry and academia devised architectural solutions to the problem. This thread of research produced the most widely-used packet classification device technology, Ternary Content Addressable Memory (TCAM) [10, 11, 12, 13].

In this section, we provide an overview of the foundational work upon which our contribution rests. A survey of the packet classification literature yields a number of commonalities [14]. In particular, two algorithmic techniques are present in a number of the most compelling contributions: utilizing decomposition [5, 3, 6, 15, 1] and leveraging the statistical structure of filter sets [16, 4, 9, 17, 18, 1]. Section 2.2 introduces the concept of decomposition and its use in existing packet classification techniques. Section 2.3 presents a synopsis of the label encoding technique employed in the *Distributed Crossproducting of Field Labels* (*DCFL*) algorithm [1]. Our technique represents a hybrid approach to the packet classification problem, a blend of algorithmic strategies and a modified CAM architecture. The techniques mentioned in this section are small portion of an extensive body of work. In Section 9 we discuss algorithmic alternatives to our approach, as well as the barriers to deploying algorithmic solutions. We believe that *LECAM* strikes a favorable balance among the various design tradeoffs and limits the number of implementation parameters, and thus has the potential to carry several algorithmic ideas into practice. We begin with an overview of Content Addressable Memory (CAM) technologies, including Ternary Content Addressable Memory (TCAM) which is the dominant packet classification solution in use today.

2.1 Content Addressable Memory Technologies

Content Addressable Memories (CAMs) are fully associative storage devices. Fixed-length binary words can be stored in any location in the device. The memory can be queried to determine if a particular word, or key, is stored, and if so, the address at which it is stored. This search operation is performed in a single clock cycle by a parallel bitwise comparison of the key against all stored words. Since it is possible for multiple CAM entries to match the same query, CAMs often include priority resolution logic to return a single match. This is often a simple mechanism that returns the first match in the device, requiring the set of stored words to be sorted according to priority.

While binary CAMs perform well for exact match operations and can be used for route lookups in strictly hierarchical addressing schemes [12], the wide use of address aggregation techniques like CIDR requires storing and searching entries with arbitrary prefix lengths. In response, Ternary Content Addressable Memories (TCAMs) were developed with the ability to store an additional “Don’t Care” state [10, 11, 12, 13]. A stored “don’t care” bit is a wildcard which matches any query bit in that position. This mechanism allows for representation of wildcards and prefixes by using “don’t care” bits in the remainder of the field, and also supports arbitrary bit mask matching quite naturally.

This capability allows TCAMs to provide single clock cycle lookups for arbitrary prefix matching for IP lookup. TCAMs can also be used for packet classification by taking each filter and storing the concatenation of its fields in a TCAM entry. A search is performed by using the concatenation of packet header fields as a search key to the TCAM. It is due to their deterministic lookup performance that TCAMs are the dominant packet classification solution in use. For standard IPv4 5-tuple, current TCAM vendors claim search rates ranging between 40M and 125M lookups per second [19, 20, 21]. Some devices allow the TCAM to be partitioned into a small number of subsets. Note that some vendors increase the advertised search rate by a factor of n if the device supports parallel searches over n partitions. Some devices also support multiple matches at the cost of decreased lookup rate. Specifically, the protocol for multiple matches is often to submit the search key to the device to receive the highest priority match, then submit subsequent search commands to return the next highest priority matches. Multiple match searches often require a minimum number of clock cycles between queries. For example, retrieving the three highest priority matches from a current device requires 36 clock cycles, dropping the lookup rate from 125M searches per second to

value	*a1*	*a2*
Don't Care	0	0
1	0	1
0	1	0
undefined	1	1

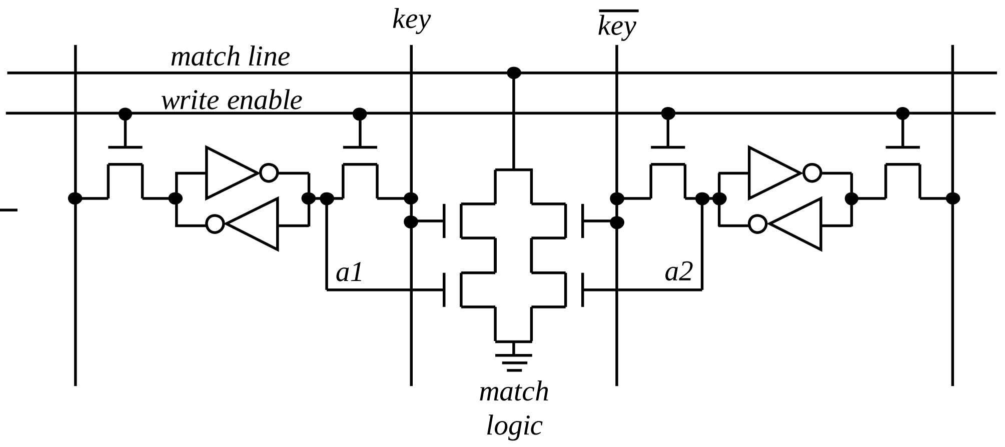

Figure 1: Circuit diagram of a standard TCAM cell; the stored value (0, 1, Don't Care) is encoded using two registers *a1* and *a2*.

approximately 7M searches per second.

Clearly the multiple match performance of TCAMs could be improved with additional hardware and engineering effort, but transmitting multiple results across the output interface require several cycles. Note that TCAMs typically operate the I/O interfaces at twice the rate of the core match logic. While this doubles the I/O bandwidth, it does not remove the fundamental constraints imposed by bandwidth limited I/O interfaces. While TCAMs can be engineered to provide single cycle lookups, bandwidth limited I/O interfaces place a fundamental limit on the amount of key data that a device can receive and matching filter data that a device can transmit. Clearly, there is an opportunity to trade off single cycle lookups in order to achieve greater scalability and efficiency without degrading the achievable lookup performance. Given a fixed bandwidth input interface, larger input keys require additional clock cycles to transmit across the interface, providing additional clock cycles for the searching task.

TCAMs also suffer from four primary deficiencies: (1) storage inefficiency, (2) high power consumption, (3) limited scalability to long input keys, and (4) inefficient support of multiple filter matches. The storage inefficiency comes from two sources. First, arbitrary ranges must be converted into prefixes. In the worst case, a range covering w-bit port numbers may require $2(w-1)$ prefixes. Note that a single filter including two port ranges could require $2(w-1)^2$ entries, or 900 entries for 16-bit port numbers. As reported in [14], an analysis of 12 real filter sets found that the *Expansion Factor*, or ratio of the number of required TCAM entries to the number of filters, ranged from 1.0 to 6.2 with an average of 2.32. This suggests that designers should budget at least seven TCAM entries per filter, compounding the hardware and power inefficiencies described below. The second source of storage inefficiency stems from the additional hardware required to implement the third "Don't Care" state. In addition to the six transistors required for binary digit storage, a typical TCAM cell requires an additional six transistors to store the mask bit and four transistors for the match logic, resulting in a total of 16 transistors and a cell 2.7 times larger than a standard SRAM cell [13]. A circuit diagram of a standard TCAM cell is shown in Figure 1. Some proprietary architectures allow TCAM cells to require as few as 14 transistors [10].

The massive parallelism inherent in TCAM architecture is the source of high power consumption. Each "bit" of TCAM match logic must drive a match word line which signals a match for the given key. The extra logic and capacitive loading result in access times approximately three times longer than SRAM. Current TCAM devices provide 18Mb of ternary storage and consume on the the order of 20 - 30 Watts, 1.1 to 1.6 micro-Watts per "bit" compared to 20 to 30 nano-Watts per bit for DDR SRAM [22]. Supporting large filter sets may require several TCAM devices working in concert, consuming a significant portion of the board real estate, power and thermal budget on high-performance line cards. Larger filter sets and longer search keys only compound these issues, providing increased incentives for designers to tackle the implementation challenges of solutions providing greater efficiency.

Recent work has addressed the power-consumption and inefficient support of arbitrary ranges in TCAMs. *Extended TCAM* (E-TCAM) reduces power consumption by over 90% relative to standard TCAM by partitioning the TCAM into blocks, and only searching a subset of the blocks on each query [23]. Each block has an associated *index filter* which covers all filters stored in the block, and is used to determine whether that block must be searched for a given query. A search in an E-TCAM first checks the query against all index filters; for each index filter that matches the query, the corresponding TCAM block is enabled for search in the second step. Filters are organized carefully in order to minimize the number of blocks that must be searched for any given query. The *Extended TCAM* architecture also includes direct hardware support for range checking [24]; this avoids the need to replicate rules for handling range match fields as described earlier. While E-TCAM provides an elegant solution to the power consumption problem, it does not address the scalability issues inherent in TCAMs for longer search keys and larger filter sets. All TCAM solutions suffer from limited scalability to longer search keys due to its use of the exhaustive search approach. As discussed in Section 5, the explicit storage of each filter becomes more inefficient as filter sizes increase and the number of unique match conditions remains limited. If the additional filter fields require range matches, this effect is compounded due to the previously described inefficiency of mapping arbitrary ranges to prefixes.

2.2 Decomposition

Decomposing the multi-field search problem allows solutions to take advantage of the parallelism provided by Application-Specific Integrated Circuits (ASICs), Field-Programmable Gate Arrays (FPGAs), and various parallel processing architectures (SIMD, MIMD, and multi-processor systems). Given the different types of matching conditions for each packet header field, decomposition techniques typically divide the problem into two stages as illustrated in Figure 2. The first stage utilizes independent search engines to find the match conditions for each packet header field. For example, prefix matching search engines find the matching address prefixes specified by the filters in the filter set for the IP addresses in the packet header; range matching search engines find the matching port ranges specified by the filters in the filter set for the the transport port numbers in the packet header; etc. While this specific approach is employed by several algorithms [5, 15, 1], there are a variety of other ways to decompose the search.

The consequence of decomposition is a collection of intermediate results generated by the independent search engines. As shown in Figure 2, a second aggregation stage must aggregate the results and return the highest priority matching filter(s). Priority resolution may be performed as an independent third stage or integrated with the aggregation stage. There are several proposed techniques that explore various design tradeoffs for aggregating intermediate results. The seminal *Parallel Bit-Vectors* (*BV*) scheme used a simple bit-vector intersection technique that maps well to hardware [5]. The presorting of filters by priority and poor scalability to large filter sets prevent this technique from being a sustainable option. The *Crossproducting* technique performs aggregation in a single step by precomputing a table of crossproducts, a table of all possible combinations of match conditions specified by the filters in the filter set [6]. Each search engine returns the most specific match condition; the combination of these intermediate results corresponds to an entry in the table of crossproducts; each entry in the table stores the best matching filter(s) for the packet matching those conditions. While *Crossproducting* minimizes the number of memory accesses required for the aggregation step, memory requirements scale exponentially with the number of filters and the extensive use of precomputation precludes dynamic updates at high rates.

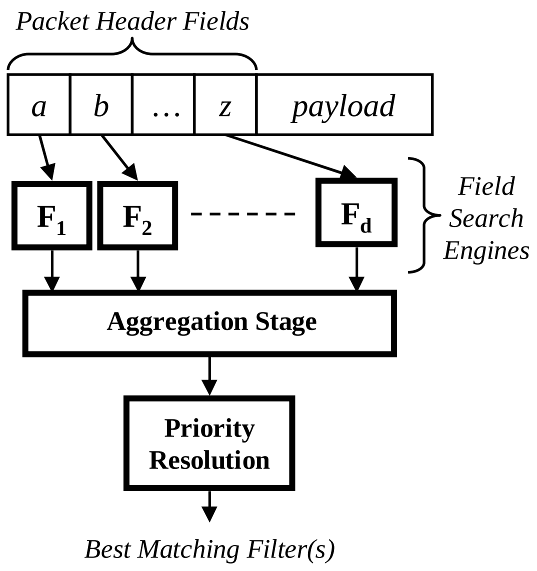

Figure 2: Block diagram of decomposition techniques for packet classification.

Similar to the *Crossproducting* technique, *Recursive Flow Classification* (*RFC*) performs independent, parallel searches on "chunks" of the packet header, where "chunks" may or may not correspond to packet header fields [3]. The results of the "chunk" searches are combined in multiple phases, rather than a single step as in *Crossproducting*. The result of each "chunk" lookup and aggregation step in *RFC* is an equivalence class identifier, *eqID*, that represents the set of potentially matching filters for the packet. The number of *eqIDs* in *RFC* depends upon the number of distinct sets of filters that can be matched by a packet. *RFC* lookups in "chunk" and aggregation tables utilize indexing, causing *RFC* to make very inefficient use of memory. The index tables used for aggregation also require significant precomputation in order to assign the proper *eqID* for the combination of the *eqID*s of the previous phases. As with *Crossproducting*, such extensive precomputation precludes dynamic updates at high rates.

The *Parallel Packet Classification* (P^2C) scheme also falls into the class of techniques using decomposition and is the most closely related technique to *LECAM* [15]. P^2C starts by precomputing the elementary intervals formed by the projections of the match conditions for each filter field. Similar to the *Parallel Bit-Vector* and *RFC* techniques, P^2C performs parallel searches in order to identify the elementary interval covering each packet field. The authors introduce three techniques for encoding the elementary intervals that explore the design tradeoffs between update speed, space efficiency, and lookup complexity. For each field of each filter, P^2C computes the common bits of all the encodings for the elementary intervals covered by the given filter field. This computation produces a ternary search string for each filter field. The ternary strings for each field are concatenated and stored in a TCAM according to the filter priority. For the single field searches, the authors employ the *BARTs* technique which restricts independent field searches to be either prefix or exact match [25]. Arbitrary ranges must be converted to prefixes, increasing the number of unique field specifications. The primary deficiency of P^2C is its use of elementary intervals, as a single filter update may add or remove several elementary intervals for each field. When using the most space efficient encoding techniques, it is possible for one filter update to require updates to every primitive range encoding. Using the most update efficient encoding, the number and size of intermediate results grows super-linearly with the number of filters. For a sample filter set of 1733 filters, P^2C required 2k bytes of SRAM and 5.1k bytes of TCAM. The same filter set requires 24k bytes using a standard TCAM exclusively, thus P^2C reduced TCAM storage requirements by a factor of 4.7 and required only 1.2 bytes of SRAM per filter. In

Section 3, we present a technique that utilizes a more efficient CAM architecture and can reduce the CAM area requirement by a factor of 6.7 or more.

2.3 Label Encoding

Several recently introduced algorithms leverage the statistical structure of filter sets to achieve faster lookup rates and greater memory efficiency [3, 7, 9, 15, 26, 4, 17]. Many of the observed characteristics of filter sets arise due to the administrative policies that drive their construction. The most complex packet filters typically appear in firewall and edge router filter sets due to the heterogeneous set of applications supported in these environments. Firewalls and edge routers typically implement security filters and network address translation (NAT), and they may support additional applications such as Virtual Private Networks (VPNs) and resource reservation. Typically, these filter sets are created manually by a system administrator using a standard management tool such as CiscoWorks VPN/Security Management Solution (VMS) [27] and Lucent Security Management Server (LSMS) [28]. Such tools conform to a model of filter construction which views a filter as specifying the pair of communicating networks or hosts (IP source and destination prefixes), then specifying the application(s) (transport protocol and port ranges). Hence, we can view each filter as having two major components: an address prefix pair and an application specification. Intuitively, we expect that a pair of communicating networks will appear in multiple filters, one for each application communicating between the two networks. Similarly, an application will appear in multiple filters, one for each pair of communicating networks using the application.

This intuition has been confirmed by several studies of real filter sets from a variety of application environments [26, 18, 14, 29]. This structure yields two characteristics of interest:

- *Match Condition Redundancy*: for each filter field, the number of unique match conditions specified by filters in the filter set is significantly less than the number of filters in the filter set
- *Matching Set Confinement*: for each filter field, the number of match conditions that can be matched by a single packet header field is small and remains small as filter sets scale in size

The amount of *match condition redundancy* varies widely among the fields within a given filter sets, but is a consistent characteristic among filter sets from a variety of application environments [29]. Similarly, *matching set confinement* is typically limited to seven or fewer match conditions per packet header field [29]. Table 2 reports *matching set confinement* results from 12 real filter sets provided by Internet Service Providers (ISPs), a network equipment vendor, and research colleagues. *Match condition redundancy* can also be observed in Table 2, as the number of unique match conditions for each filter field is significantly less than the number of filters in each filter set.

The label encoding technique employed by the *Distributed Crossproducting of Field Labels* (*DCFL*) algorithm leverages *match condition redundancy* and *matching set confinement* to construct fast, efficient aggregation data structures [1]. The first step in label encoding is to construct the sets of unique matching conditions for each filter field. Table 3 shows the sets of unique matching conditions for each filter field in the example filter set shown in Table 1. For each set, a locally unique label is assigned to each match condition. In our example in Table 3, we use lowercase Latin letters; in practice, we would simply use m-bit binary integers where $m = \log_2 M$ and M is the maximum number of unique match conditions. We discuss the provisioning of label space in Sections 6 and 8. For each unique match condition, we also maintain a count of the number of filters that specify the condition. The count values provide efficient support of incremental updates. We discuss the incremental update procedure for *LECAM* in Section 7.

Table 2: Matching set confinement for 12 real filter sets; number of unique match conditions for each filter field is given in parentheses.

Filter Set		*Fields*					
	Size	*Src Addr*	*Dest Addr*	*Src Port*	*Dest Port*	*Prot*	*Flag*
fw2	68	3 (31)	3 (21)	2 (9)	1 (1)	2 (5)	
fw5	160	5 (38)	4 (35)	3 (11)	3 (33)	2 (4)	2 (11)
fw3	184	4 (31)	3 (28)	3 (9)	3 (39)	2 (4)	2 (11)
ipc2	192	3 (29)	2 (32)	2 (3)	2 (3)	2 (4)	2 (8)
fw4	264	3 (30)	4 (43)	4 (28)	3 (49)	2 (9)	
fw1	283	4 (57)	4 (66)	3 (13)	3 (43)	2 (5)	2 (11)
acl2	623	5 (182)	5 (207)	1 (1)	4 (27)	2 (5)	2 (6)
acl1	733	4 (97)	4 (205)	1 (1)	5 (108)	2 (4)	2 (3)
ipc1	1702	4 (152)	5 (128)	4 (34)	5 (54)	2 (7)	2 (11)
acl3	2400	6 (431)	4 (516)	2 (3)	6 (190)	2 (5)	2 (3)
acl4	3061	7 (574)	5 (557)	2 (3)	7 (235)	2 (7)	2 (3)
acl5	4557	3 (169)	2 (80)	1 (1)	4 (40)	1 (4)	2 (2)

Table 3: Sets of unique match conditions for each filter field.

SA	*Label*	*Count*
*	a	2
1010 1101	b	1
1101 0100	c	1
1101 01*	d	1
1010 11*	e	1
1101*	f	1
0*	g	2

DA	*Label*	*Count*
1101 0100	a	2
1010 1101	b	1
1010 11*	c	3
1101 01*	d	1
*	e	2

DP	*Label*	*Count*
$[0:15]$	a	5
3	b	2
$[4:15]$	c	2

PR	*Label*	*Count*
TCP	a	3
UDP	b	2
*	c	4

Table 4: Example filter set.

SA	*DA*	*Prot*	*DP*	*Label*
*	1101 0100	[0:15]	TCP	(a,a,a,a)
1010 1101	1101 0100	3	UDP	(b,a,b,b)
1101 0100	1010 1101	3	UDP	(c,b,b,b)
1101 01*	1010 11*	[0:15]	*	(d,c,a,c)
1010 11*	1101 01*	[0:15]	*	(e,d,a,c)
1101*	1010 11*	[4:15]	TCP	(f,c,c,a)
0*	1010 11*	[4:15]	TCP	(g,c,c,a)
0*	*	[0:15]	*	(g,e,a,c)
*	*	[0:15]	*	(a,e,a,c)

Once we construct the sets of unique match conditions, we can encode a filter by concatenating the labels for the match conditions specified by the filter. Table 4 shows the label encoding for the filters in the example filter set in Table 1. Label encoding essentially assigns a unique fixed-length key value to each filter. Rather than a combination of variable-length prefixes, arbitrary ranges, exact values, or wildcards, the filter can be uniquely identified by a combination of fixed-length labels. *DCFL* leverages the label encoding scheme to represent sets of unique match condition combinations and to construct efficient set membership data structures for its distributed aggregation technique. While this yields a fast and efficient algorithm, there are a number of implementation parameters that affect the achievable performance. Implementation parameters, such as the width of the internal memory interfaces, can have a drastic effect on the lookup rate and memory efficiency, and thus demand prudent tuning in order to achieve a favorable balance. In an effort to reduce the number and sensitivity of implementation parameters, we introduce a conceptually simpler aggregation technique that leverages label encoding and a modified CAM architecture. The result is a high-performance packet classification technique that is practical to implement that has the potential to carry several algorithmic ideas into practice.

3 Description of LECAM

Label Encoded Content Addressable Memory (*LECAM*) is a practical, scalable, and high-performance packet classification technique that employs decomposition, label encoding, and a modified CAM architecture. As shown in the example in Figure 3, the first stage of *LECAM* is a collection of parallel search engines, one for each filter field. Each search engine is optimized for the type of matching condition specified by the associated filter field: prefix matching for IP address prefixes, range matching for transport port ranges, exact matching for the transport protocol field, etc. We briefly discuss suitable algorithms for the various types of search engines in Section 4. The input to each search engine is a packet header field and the output is a set of labels corresponding to the match conditions matched by the packet header. In practice, the only meta-data that need to be stored by the data structures for the match conditions are labels. A control processor can manage the count values for each match condition and trigger additions and deletions to the search engine data structures only when counters change from a zero to a one, or vice versa. We discuss incremental update support in Section 7.

Once the search engines resolve the sets of labels for the match conditions on each field, the results are fed to a modified CAM architecture that aggregates the results and returns the set of matching filters for the packet. Note that there are various ways to architect the *Aggregation CAM*. Here we describe an architecture in which the set of *all* matching filters is returned by the *Aggregation CAM* and priority resolution is deferred

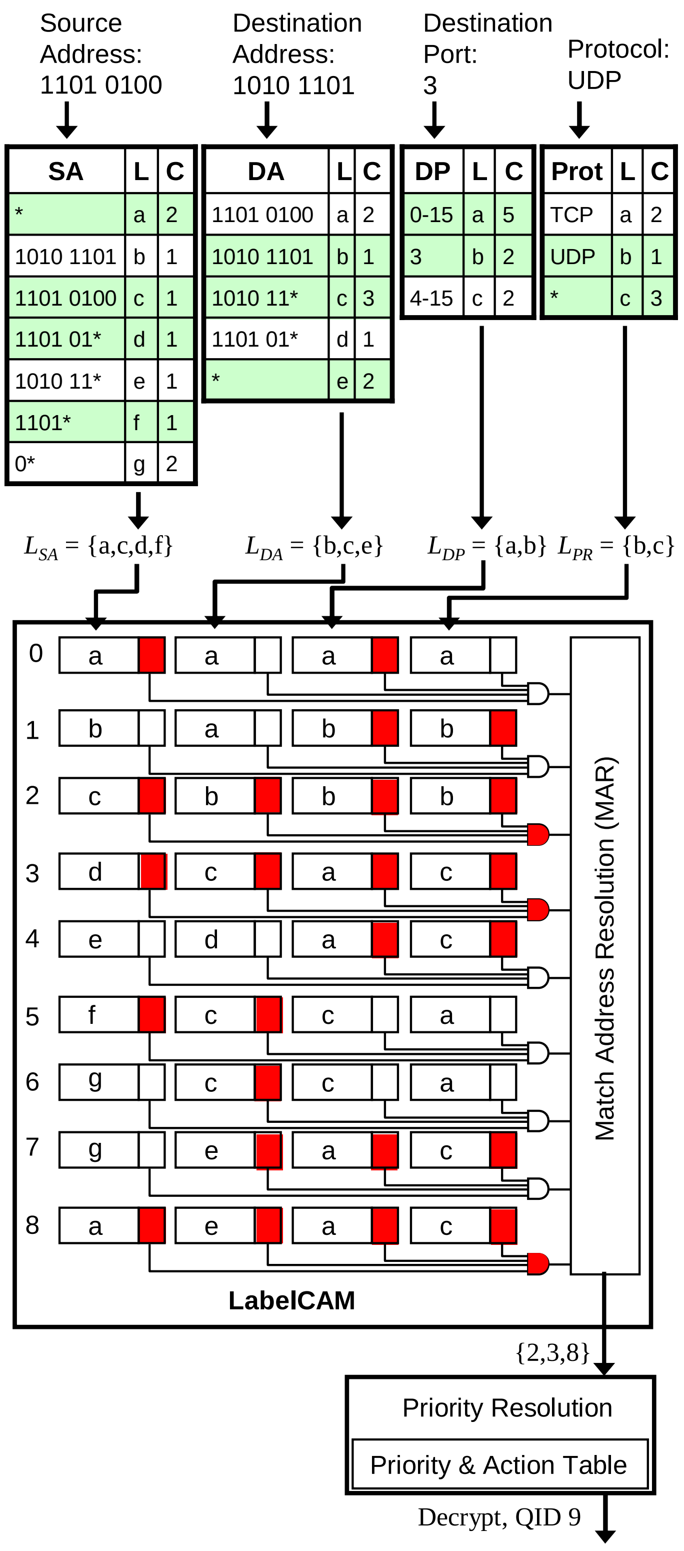

Figure 3: Example search of the filter set in Table 1 using *Label Encoded Content Addressable Memory* (*LECAM*).

to a third stage. We discuss the consequences of this design decision and alternative *Aggregation CAM* architectures in Section 8.

As shown in Figure 3, the *Aggregation CAM* is organized as a two-dimensional array of label cells, where each row stores the label combinations for the filters in the filter set. Note that provisioning *Aggregation CAM* rows is significantly simpler than provisioning TCAM slots, as each filter only consumes one *Aggregation CAM* row. Recall that filters may consume hundreds of TCAM slots due the conversion of multiple range fields to prefix fields [14, 29]. The columns of label cells in the *Aggregation CAM* correspond to filter fields. Each label cell contains a match flag that is cleared at the beginning of each search. The *Aggregation CAM* resolves the matching filters for a packet by performing exact match queries to each column using the label sets returned by the field search engines. Match flags are set in the label cells containing matching labels. In the example in Figure 3, match flags are denoted by the small boxes adjacent to the label cells; a shaded box denotes that the match flag is set. Once the column queries are completed, the set of matching filters is determined by simply computing the logical *AND* of the match flags of the label cells in each row. In this CAM architecture, the *Matching Address Resolution* (*MAR*) block is a simple encoder that detects the matching rows and passes the set of matching addresses (filters) to the priority resolution stage.

Aggregation CAM columns are queried in parallel; thus, the time to complete this stage of the search is determined by the maximum matching set size among all the filter fields. Recall that the *matching set confinement* property limits the maximum matching set size for each filter field to seven or less for the 12 real filter sets that we studied. In order to provide guarantees on the worst-case lookup rate of *LECAM*, we would like to constrain the maximum matching set size. *Field Splitting* is a mechanism for placing a limit on the matching set size by partitioning the sets of match conditions [1]. Note that the maximum matching set size for exact match conditions is always less than or equal to two; therefore, *field splitting* is applied to sets of prefix matching and range matching conditions that generate matching set sizes that are greater than a threshold. For example, assume that we want to limit the maximum matching set size to three labels. Assume that one set of range match conditions for 4-bit port numbers contains the following ranges: $[0 : 15]$, $[0 : 3]$, $[3 : 3]$, $[2 : 14]$, $[11 : 12]$, $[14 : 14]$. If a packet contains the port number 3, then the matching set size would be four; thus, we must partition the set of ranges. Let A be the set of ranges $\{$ $[0 : 15]$, $[0 : 3]$, $[3 : 3]$, $[11 : 12]$, $[14 : 14]$ $\}$ and let B be set of ranges $\{$ $[0 : 15]$, $[2 : 14]$ $\}$. The maximum matching set size is three for set A and two for set B. Note that we included a default match, $[0 : 15]$, in set B. By ensuring that every set of match conditions has a wildcard entry, matching sets contain at least one label which simplifies the design of the *Aggregation CAM*. The Appendix contains a description of an $O(NW)$ algorithm for computing the splits for address prefixes and an $O(N \log N)$ algorithm for computing the splits for arbitrary ranges that ensure a maximum matching set size; N is the number of match conditions in the set and W is the length of the IP address in bits.

Each new set of match conditions resulting from *field splitting* adds an additional label to the combination of labels that identify a filter. Consequently, additional columns must be added to the *Aggregation CAM*; however, it does not necessarily degrade the achievable lookup rate. The additional field searches and column queries occur in parallel. If we assume a fixed bandwidth input interface, inputing longer search keys requires more clock cycles. Consequently, this provides additional clock cycles for field searches and column queries to complete, relaxing the lookup rate constraint and reducing the need for *field splitting*. Also note that while *field splitting* results in additional columns, it reduces the number of unique labels that each column must store, and thus reduces the minimum label cell size. We discuss heuristics for provisioning column widths (label cell sizes) in Section 8. A label cell consists of b bits of CAM and match flag latch. Recall that a CAM bit requires six transistors for bit storage and four transistors for match logic. The match line of the label cell drives the set input of the match flag latch which can be implemented with five transistors.

Note that the match line is only driven by the CAM bits in the label cell, rather than the entire row (or slot) as in a standard CAM or TCAM. This reduces the capacitive loading on the match line which can potentially allow the *Aggregation CAM* to be operated at higher clock frequencies than standard CAM technologies of comparable capacity. Note that the architecture of the *Aggregation CAM* does not reduce the capacitive loading of distributing the input key to every row. Using driver trees, it is possible to reduce this loading effect and achieve relatively high clock frequencies. An analysis of the capacitive loading and achievable clock frequency is beyond the scope of this paper, but we seek to highlight the opportunities to engineer a solution that operates at faster rates than current TCAMs.

4 Search engine options

A significant advantage of all decomposition techniques, including *LECAM*, is the ability to search each filter field with a search engine optimized for the particular type of match condition. The label encoding technique provides a clean layer of indirection in that it does not place any additional constraints on the search engines. It only requires a small amount of meta-data, a label, to be stored with each entry. While our focus is on the performance and scalability of the *Aggregation CAM*, we briefly discuss options for implementing the parallel search engines.

4.1 Prefix Matching

Since *LECAM* does not precompute sets of matching filters, it requires that the search engines for the IP source and destination addresses return *all* matching prefixes for the given addresses. Any longest prefix matching technique can support All Prefix Matching (APM), but some more efficiently than others. The most computationally efficient algorithm for longest prefix matching is *Binary Search on Prefix Lengths* [30]. When precomputation and marker optimizations are used, the technique requires at most five hash probes per lookup for 32-bit IPv4 addresses. Real filter sets contain a relatively small number of unique prefix lengths, thus the realized performance should be better for real filter sets. Markers direct the search to longer prefixes that potentially match, skipping shorter prefixes that may match. In order to support APM, *Binary Search on Prefix Lengths* must precompute all matching prefixes for each “leaf” in the trie defined by the set of address prefixes. While computationally efficient for searches, this technique does present several challenges for hardware implementation. Likewise, the significant use of precomputation and markers degrades the dynamic update performance, as an update may require many memory transactions.

As demonstrated in [31], compressed multi-bit trie algorithms readily map to hardware and provide excellent lookup and update performance with efficient memory and hardware utilization. Specifically, our implementation of the Tree Bitmap algorithm [32] requires at most 11 memory accesses per lookup and approximately six bytes of memory per prefix. Each search engine consumes less than 1% of the logic resources on a commodity FPGA. As discussed in [31], there are a number of optimizations to improve the performance of this particular implementation. Use of an initial lookup array for the first 16 bits reduces the number of memory accesses to at most seven. Coupled with a simple two-stage pipeline, the number of sequential memory accesses per lookup can be reduced to at most four. Trie-based LPM techniques such as Tree Bitmap easily support all prefix matching with trivial modifications to the search algorithm. The set of matching prefixes can be compiled as the search traverses the trie.

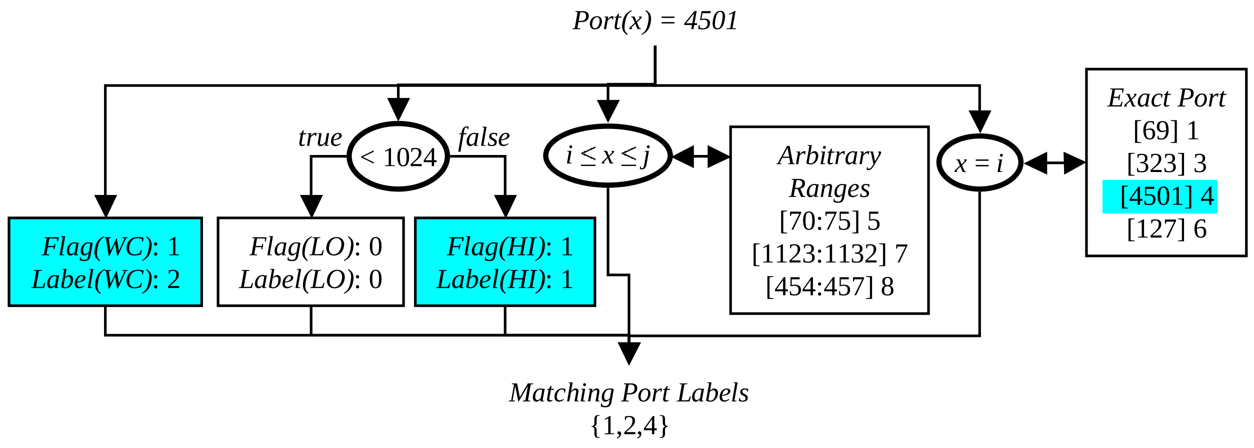

Figure 4: Block diagram of range matching using parallel searches for each port class.

4.2 Range Matching

Searching for all arbitrary ranges that overlap a given point presents a greater challenge than prefix matching. Based on observations of the structure of port range conditions in real filter sets, range matching can be made sufficiently fast by using a set of parallel searches. These observations of commonly specified port ranges also influenced the model of filter set structure developed for the *ClassBench* tools [2] which defines five *port classes*: wildcard (WC), user port range (HI) $[1024 : 65535]$, system port range (LO) $[0 : 1023]$, exact matches (EM), and arbitrary ranges (AR). Figure 4 shows an example where parallel searches are performed for each *port class*. For the first port class, wildcard (WC), the search simply consists of a flag specifying whether or not the wildcard is specified by any filters in the filter set and a register for the label assigned to this range specification. We also add logic to check if the port is less than 1024; this checks for a match on the commonly specified user (HI) and system (LO) port ranges, $[1024 : 65535]$ and $[0 : 1023]$, respectively. Similar to the WC class, the HI and LO searches require flags denoting whether or not the ranges are specified by any filters in the filter set and registers for the labels assigned to those range specifications.

For the 12 real filter sets we studied, the number of exact port numbers specified by filters was at most 183. The port ranges in the EM port class may be efficiently searched using any sufficiently fast exact match data-structure. Entries in this data-structure are simply the port number and the assigned label. A simple hash table could bound searches to at most two memory accesses. Finally, the set of arbitrary ranges in the AR port class may be searched with any range matching technique. Fortunately, the set of arbitrary ranges tends to be small; the 12 real filter sets specified at most 27 arbitrary ranges. A simple balanced interval tree data-structure requires at most $O(k \lg n)$ accesses, where k is the number of matching ranges and n is the number of ranges in the tree. Other options for the AR search include the *Fat Inverted Segment Tree* [8] and converting the arbitrary ranges to prefixes and employing an all prefix matching algorithm. Given the limited number of arbitrary ranges, adding multiple prefixes per range to the data-structure does not cause significant memory inefficiency. With sufficient optimization, we assume that range matching can be performed with at most four sequential memory accesses and the data-structures for the AR and EM port classes easily fit within a standard embedded memory block.

4.3 Exact Matching

The protocol and flag fields may be easily searched with a simple exact match data-structure such as a hash table. Given the small number of unique protocol and flag specifications in the real filter sets (less than 9 unique protocols and 11 unique flags), the time per search and memory space required is trivial. We expect that additional filter fields will also require exact match search engines. Given the ease of implementing hash functions in custom and reconfigurable logic, we do not foresee any performance bottlenecks for the search engines for these fields.

5 LECAM Scalability

A key advantage of *LECAM* relative to TCAM-based approaches is the ability to efficiently support large filter sets and long search keys. Exhaustive search techniques, such as linear search and TCAMs, enjoy a favorable linear memory requirement, $O(N)$, where N is the number of filters in the filter set. We have already pointed out that the constants can get quite large due to filter replication in TCAMs, but we also wish to illustrate the efficiency that may be achieved by taking advantage of the *match condition redundancy* property of real filter sets.

A filter using the standard IPv4 5-tuple requires about 168 bits to specify explicitly. With that number of bits, we can specify 2^{168} distinct filters. Typical filter sets contain fewer than 2^{20} filters, suggesting that there is potential for a factor of eight savings in memory. It is this observation that led to the previously described label encoding technique. Let a filter be defined by fields $f_1 \dots f_d$ where each field f_i requires b_i bits to specify. For example, a filter may be defined by a source address prefix requiring 64 bits[1], a destination address prefix requiring 64 bits, a protocol number requiring 8 bits, etc. By this definition, the memory requirement for explicitly representing the filter set for an exhaustive search is

$$N \sum_{i=1}^{d} b_i \tag{1}$$

Let $B = \sum_{i=1}^{d} b_i$. If the filter set contained 2^B filters (where each filter field specifies a unique match condition), then exhaustive search techniques would have an optimal storage requirement. In practice, this not the case.

By leveraging the *match condition redundancy* property and utilizing an efficient encoding, the storage requirement can be reduced from linear in the number of filters to logarithmic in the number of unique match conditions. Now let $u_1 \dots u_d$ be the number of unique match conditions in the filter set for each filter field i. In order to represent the filter set, we only need to store the unique match conditions for each field once. If we assign a minimum size label to the unique match conditions in each field, then the number of bits required for each label is $\lg(u_i)$. Using the label encoding technique, the memory requirement becomes

$$\sum_{i=1}^{d} (u_i \times b_i) + N \sum_{i=1}^{d} \lg u_i \tag{2}$$

The first term accounts for the storage of unique match conditions and the second term accounts for the storage of the label encoded filters.

[1]We are assuming a 32-bit address where an additional 32 bits are used to specify a mask. There are more effi cient ways to represent a prefi x, but this is tangential to our argument.

The savings factor for a given filter set is simply the ratio of Equation 1 and Equation 2. For simplicity, let $b_i = b \forall i$ and let $u_i = u \forall i$; the savings factor is:

$$\frac{Nb}{ub + N \lg u} \tag{3}$$

In order for the savings factor to be greater than one, the following relationship must hold:

$$\frac{u}{N} + \frac{\lg u}{b} < 1 \tag{4}$$

Note that $u \leq 2^b$ and $u \leq N$. Thus, the savings factor increases as the number of filters in the filter set and the size (number of bits) of the match conditions increases relative to the number of unique match conditions.

We anticipate that future filter sets will include filters with more match conditions [14]. It is also likely that the additional fields will contain a handful of unique values. As this occurs, the linear memory requirement of techniques explicitly storing the filter set will become increasingly sub-optimal. In the near term, migration to IPv6 also makes a compelling case for the label encoding approach. For the standard IP 5-tuple, search keys will swell from 104 bits to 296 bits. Making the reasonable assumption that the *match condition redundancy* property will continue to hold with IPv6, we assert that *LECAM* is a compelling solution for IPv6 packet classification.

6 Comparative performance evaluation

Our analysis focuses on the area requirements of *LECAM* relative to the area requirements of TCAM. As we have discussed, *LECAM* can be engineered to provide equivalent search rates by making use of the extra cycles required to transfer search keys across the input interface and employing the *field splitting* optimization to ensure that match set sizes do not exceed a threshold. This threshold is determined by the clock cycle budget set by the target line rate and the speed of the *Aggregation CAM* implementation. With equivalent lookup rates, the primary advantage of *LECAM* is area efficiency and scalability. Note that a reduction in area results in a proportional reduction in power consumption.

We use a collection of real and synthetic filter sets of various sizes and compositions for our analysis. The 10 real filter sets were provided by Internet Service Providers (ISPs), a network equipment vendor, and research colleagues. The filter sets range in size from 160 to 4557 entries and represent a variety of application environments and filter set formats: firewall, Access Control List (ACL), and IP Chain (IPC). All filter sets specified match conditions on six fields, the standard IPv4 5-tuple plus protocol flags. Synthetic filter sets that model the real filter sets were created using *ClassBench*, a suite of publicly-available tools for packet classification benchmarking [2]. We selected three real filter sets and generated scaled models of these filters sets containing 10k, 20k, and 50k filters. We did not adjust the *scope* or *smoothness* parameters of the *Synthetic Filter Set Generator*.

For each filter set, we estimate the TCAM area requirements by first computing the number of TCAM slots required. The number of TCAM slots, S, depends on the amount of filter replication due to range to prefix conversion. Assuming a standard 16 transistor TCAM cell and 112 bit search key, we estimate the total transistor requirement as follows:

$$A_{TCAM} = S \times 112 \times 16 \tag{5}$$

In order to estimate the area requirements for *LECAM*, we first estimate the amount of SRAM required for the search engines. Given that the sets of unique IP address prefixes are significantly larger that the sets of other unique match conditions, we base our estimates on the 8 bytes (or less) per prefix required by the Tree Bitmap algorithm. We make the assumption that small range trees and hash tables for the range match and exact match conditions, respectively, will not significantly skew the average memory requirement per entry. Assuming a standard 6 transistor SRAM cell, we estimate the number of transistors for SRAM to be:

$$A_{SRAM} = 6 \times 64 \times \sum_{i=1}^{d} u_i \tag{6}$$

where u_i is the number of unique match conditions for field i.

We evaluated two heuristics for provisioning *Aggregation CAM* label cell sizes. The first heuristic, *minimum sizing*, provisions label cells of size $b_i = \lg u_i$ bits. Recall that label cell size is uniform among cells in the same column. This heuristic maximizes the efficiency provided by the *match condition redundancy* property, but it yields an *Aggregation CAM* whose capacity varies according to filter set characteristics. For this reason, we also analyzed a *conservative sizing* heuristic that provisions label cells of size $b_i = \lg N \forall i$, where N is the number of filters in the filter set. This heuristic results in an *Aggregation CAM* that supports any filter set of N filters; every filter may contain unique match conditions. By not leveraging the *match condition redundancy* property, the *conservative sizing* heuristic is less efficient but robust against variations in filter set characteristics. The two sizing heuristics represent the endpoints in a design space where the tradeoffs are area efficiency and robustness against filter set variations. We discuss the development of a practical sizing heuristic that strikes a favorable balance among the efficiency and robustness tradeoffs in Section 8. As described in Section 3, we assume that each label cell requires $10 \times b_i + 5$ transistors. The total number of transistors for the *Aggregation CAM* is:

$$A_{CAM} = N \times \sum_{i=1}^{d} (10 \times b_i + 5) \tag{7}$$

We express the area requirement for *LECAM* relative to *TCAM* as the following area ratio:

$$R = \frac{A_{SRAM} + A_{CAM}}{A_{TCAM}} \tag{8}$$

Figure 5 shows the area ratio for the 10 real filter sets using the *minimum sizing* heuristic for the *Aggregation CAM*. The total area requirement for *LECAM* ranges between 10 and 20 percent of the area required for standard TCAM, a reduction in area by a factor of 10 to 5, respectively. Note that the relative area requirement for the *Aggregation CAM* is 15 percent or less for all filter sets, a reduction in CAM area by a factor of 6.7 or better. Clearly, the label encoding approach is capable of significantly reducing the amount of CAM area required. The most closely related packet classification approach, *Parallel Packet Classification* (P^2C), reduces the amount of TCAM required by a factor of 4.7 [15]. Via the use of the *BARTs* longest prefix matching algorithm [25], P^2C achieves outstanding SRAM efficiency, 1.2 bytes per filter. *LECAM* allows any single field search algorithms, thus the SRAM requirements could be reduced from the 8 bytes per filter that we assumed. The focus of our analysis is on the efficiency of the label encoding technique and *Aggregation CAM* architecture. Figure 6 shows the area ratio for the 10 real filter sets using the *conservative sizing* heuristic for the *Aggregation CAM*. The total area requirement for *LECAM* ranges between 13 and 37 percent of the area required for standard TCAM, a reduction in area by a factor of 7.7 to 2.7, respectively. Clearly, the choice of sizing heuristic significantly affects the achievable area efficiency.

Figure 7 shows the area ratio for 12 synthetic filter sets using the *minimum sizing* heuristic for the *Aggregation CAM*. The total area requirement for *LECAM* ranges between 14 and 54 percent of the area required for standard TCAM, a reduction in area by a factor of 7.1 to 1.9, respectively. The relative area

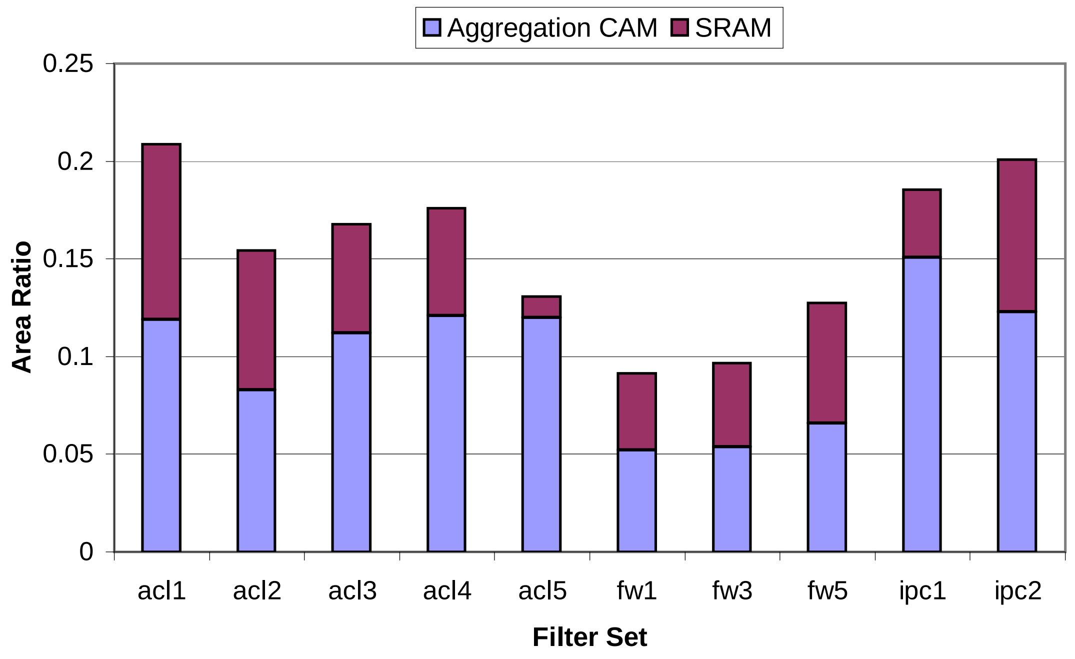

Figure 5: *LECAM* area relative to TCAM area for real filter sets; *LECAM* columns provisioned using minimum sizing heuristic.

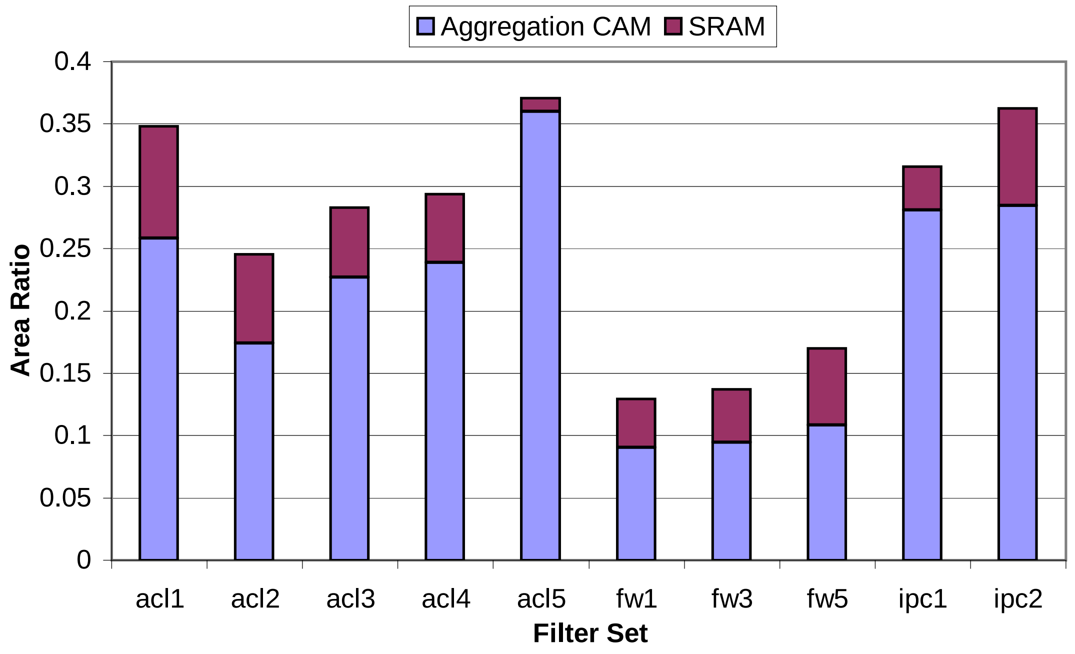

Figure 6: *LECAM* area relative to TCAM area for real filter sets; *LECAM* columns provisioned using conservative sizing heuristic.

requirement for the *Aggregation CAM* is 25 percent or less for all filter sets, a reduction in CAM area by a factor of 4 or better. Figure 8 shows the area ratio for 12 synthetic filter sets using the *conservative sizing* heuristic for the *Aggregation CAM*. The total area requirement for *LECAM* ranges between 20 and 90 percent of the area required for standard TCAM, a reduction in area by a factor of 5 to 1.1, respectively.

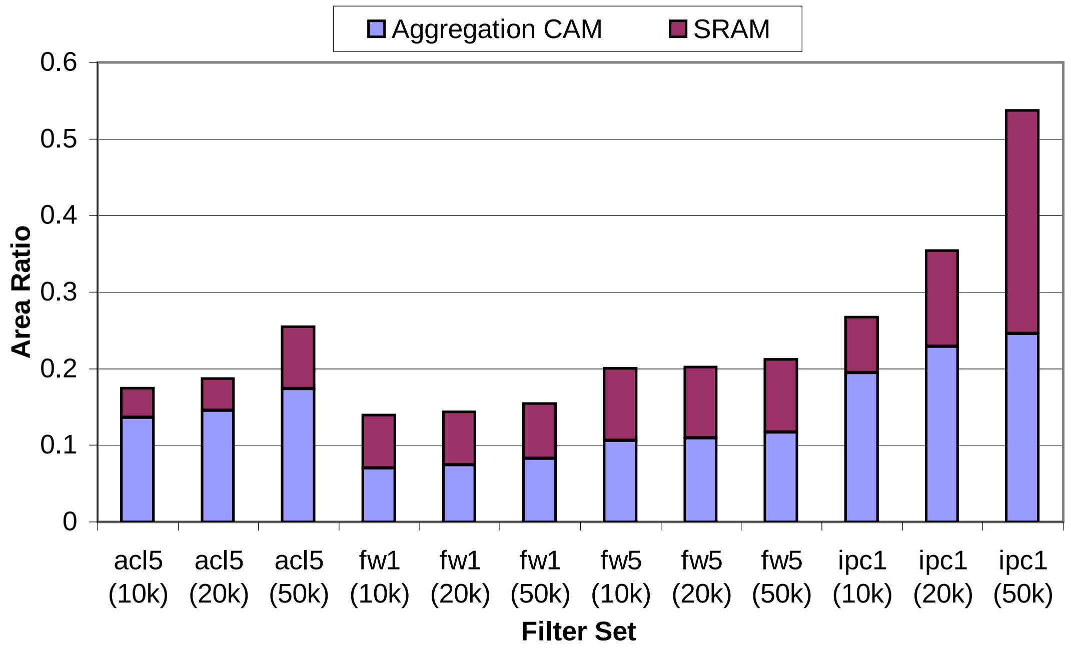

Figure 7: LECAM area relative to TCAM area for real filter sets; LECAM columns provisioned using minimum sizing..

The relative area requirement for the *Aggregation CAM* is 60 percent or less for all filter sets, a reduction in CAM area by a factor of 1.7 or better. The reduction in efficiency for the large synthetic filter sets is due to the characteristics of the filter sets generated by *ClassBench*. In order to scale filter sets orders of magnitude larger while maintaining important properties like prefix nesting thresholds and application distributions, *ClassBench* is forced to generate filters with new address prefixes. The number of unique application specifications is restricted, because it is difficult to predict the structure of filters for future applications. The tools do not generate new port ranges or protocol specifications that could be specified by many filters. This results in the roughly linear scaling in area requirements with the number of filters. If filter sets are to achieve sizes in the tens of thousands, we believe that new application specifications will be present. We also expect the *match condition redundancy* property to be more prevalent than in the filter sets generated by *ClassBench*.

7 Incremental update support

While the vast majority of current filter sets are manually configured, we anticipate that future applications, such as automated worm detection and mitigation, will be capable of dynamically generating large numbers of complex filters. High-performance router architectures typically include a dedicated control processor for configuring ports, managing route tables, etc. The following discussion presumes the existence of such a processor with sufficient memory and processing capacity for managing the filter sets at each router port. We also assume that the control processor memory stores the data structures representing the sets of unique match conditions for each filter field, along with their associated labels and count values.

Coupled with a control processor, *LECAM* is capable of supporting incremental updates at high rates. Consider the addition of a new filter into the filter set. Upon receipt of a filter add command, the control processor queries each set of unique match conditions using the associated match conditions specified by

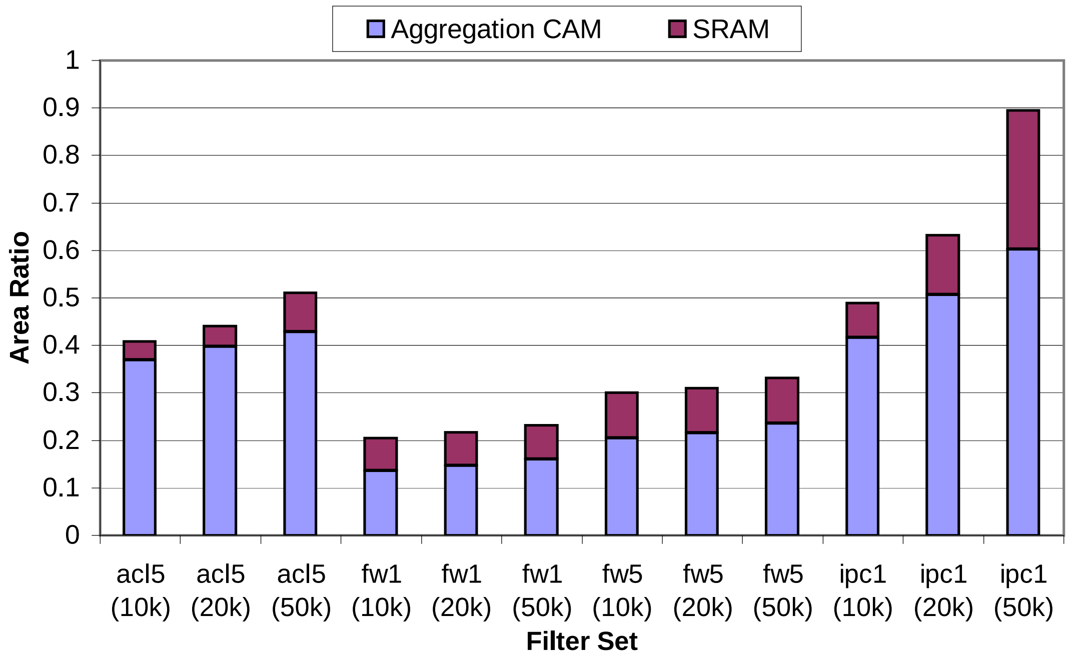

Figure 8: LECAM area relative to TCAM area for real filter sets; LECAM columns provisioned using conservative sizing.

the filter. Each set query proceeds as follows. If the match condition is contained in the set, then the count value is incremented and the label for the match condition is returned. If the match condition is not in the set, then a new entry is allocated, the count value is initialized to one, a locally unique label is assigned to the match condition, an *add match condition* flag and the new label is returned. The result of the queries is the set of labels, whose combination forms the filter label that must be added to the *Aggregation CAM*, and a set of updates to the field search engine data structures (if any of the queries returned and *add match condition* flag). These results form an *update key* to the *LECAM*.

The *update key* is inserted into the sequence of packet headers waiting for classification and flows through the search pipeline in the same manner. In parallel, each search engine processes the match condition add, if necessary. Depending on update latency of the field search engines, the *LECAM* may require backpressure mechanisms and buffering between pipeline stages. Once the field search engine updates complete and the *update key* proceeds to the *Aggregation CAM*, the field search engines can begin processing the next packet header.

The *Aggregation CAM* can be engineered to support efficient updates by including a valid bit for each row and including it in the bitwise *AND* of the label match flags for the row. If the row contains a valid entry, the valid bit is set to one, otherwise zero. In order to process the *update key* for a filter add, the *Aggregation CAM* simply scans the rows for the first row with a zero valid bit, then fills the label cells with the labels comprising the filter label and sets the valid bit to one.

Filter deletions proceed in much the same manner. The control software first resolves the set of field search engine updates and the label for the filter to be removed. In each match condition set in the control memory, the count values are decremented for the match conditions specified by the filter. Match conditions are removed from the set and updates are generated for the field search engines when the result of a count value decrement is zero. The *update key* is processed by the field search engines in the same way, with each search engine removing a match condition from its data structure, if necessary. The *Aggregation CAM* simply locates the row containing the filter label and resets the valid bit to zero.

While the processing of an *update key* may require additional clock cycles, it does not require significant computation, pipeline flushes, or reorganization of the *Aggregation CAM* entries (to restore priority order). Furthermore, we expect updates to field search engine data structures to be rare relative to filter updates due to the *match condition redundancy* property. Thus, *LECAM* supports incremental updates at high rates without significant degradation of lookup performance.

8 Implementation considerations

We have shown how the combination of label encoding and a modified CAM can yield a more efficient and scalable packet classification solution. Here we discuss several of the more significant design decisions and tradeoffs that must be addressed.

The priority-independent architecture described in Section 3 simplifies the *Match Address Resolution* (*MAR*) logic and the incremental update procedure. It also does not explicitly constrain the number of results returned by the CAM. Statistical analyses of real filter sets show that the maximum number of matching filters for any packet is typically seven or less. It is also possible to enforce a limit on the maximum number of matching filters through preprocessing the filter set. If the limit is reasonably large, then there should be minimal degradation of the incremental update performance. It is also possible to implement priority circuitry that performs exclusive and non-exclusive priority resolution within the *Aggregation CAM*.

High lookup rates can be achieved by pipelining the *LECAM* architecture. By including a small set of pipeline buffers between the field search engines and the *Aggregation CAM*, the field search engines can start processing the next packet header before the *Aggregation CAM* completes the label aggregation step. While this is a simple two-stage pipeline, it does require that we balance the worst-case performance of field search engines and the *Aggregation CAM*. For the field search engines, it is possible to meet a performance constraint by local optimizations. For the *Aggregation CAM*, the worst-case performance depends on the maximum matching set size which can be limited by employing the *field splitting* optimization.

The number of filters supported by *LECAM* is a function of the amount of *SRAM* available to store the sets of unique match conditions and the number of rows in the *Aggregation CAM*. Due to the *match condition redundancy* property of filter sets and the efficiency of algorithms for prefix, range, and exact matching, we expect *SRAM* requirements to scale linearly (or better) with the number of filters. Similarly, each filter requires one row in the *Aggregation CAM*; the number of rows scales linearly with the number of filters.

Perhaps the most important design decision is provisioning the number and width of *Aggregation CAM* columns. The choice of sizing heuristic for column width will likely depend on the application environment. If *LECAM* is to be used in a large core router, then the column width should be selected based on analysis of core router filter sets and suitable engineering margins. In this case, the efficiency of *LECAM* can approach the efficiency achieved with the *minimum sizing heuristic*. If *LECAM* is used in a general packet classifier or “network search engine” product, then the sizing heuristic should be more conservative. Likewise, the efficiency will more closely resemble that achieved with the *conservative sizing heuristic*. When provisioning the number of columns, we must consider two factors: support for additional filter fields and overhead for *field splitting*. For filter fields with exact match conditions, *field splitting* is not required, as matching sets will contain at most two labels. For the filter sets we studied, maintaining a maximum matching set size of four required that prefix match and/or range match fields be split no more than once. For the highest performance applications where *field splitting* is required to provide line speed lookup rates, one additional column for prefix and range match fields would be sufficient.

In addition to provisioning extra *Aggregation CAM* columns, classifying on additional filter fields and *field splitting* require additional field search engines and associated SRAM. In order to support flexibility in a hardware implementation, we can provision field search engine modules consisting of embedded SRAM and reconfigurable logic blocks [33, 34]. While reconfigurable logic technology is area-inefficient relative to custom logic, the amount of logic required for field search engines is typically small [31].

9 Algorithmic Alternatives

In addition to optimizations for TCAM-based solutions, the research community has produced a number of algorithmic solutions to the packet classification problem [14]. Introduced by Singh, Baboescu, Varghese, and Wang, *HyperCuts* is one of the most promising algorithmic solutions to emerge [17]. *HyperCuts* improves upon the *HiCuts* algorithm developed by Gupta and McKeown [4] and also shares similarities with the *Modular Packet Classification* algorithms introduced by Woo [9]. In essence, *HyperCuts* is a decision tree algorithm that attempts to minimize the depth of the tree by selecting multiple “cuts” in d-dimensional space that optimally segregate packet filters into lists of bounded size. The data structure is searched by traversing the decision tree, where branching decisions are made according to the cuts defined by the node and the packet header fields. Once a leaf list is reached, a linear search over the list is performed in order to find the best matching filter. Performance evaluations with real and synthetic filter sets showed that traversing the *HyperCuts* decision tree requires between 8 and 35 memory accesses, and memory requirements for the decision tree ranged from 5.4 to 145.9 bytes of per filter. Additional memory accesses and space are required for leaf nodes. It should also be noted that *HyperCuts* cannot efficiently handle filters with wildcards in both the source and destination IP address, thus the authors suggest that a separate data structure be used. While the *HyperCuts* results are generally promising, it is unclear if the technique is capable of providing the necessary performance and scalability. A pipelined hardware implementation of the technique could provide sufficient throughput, but provisioning memory for each pipeline stage would be challenging due to the wide variance in memory requirements among filter sets. As with any decision-tree technique, larger filter sets and filters with additional fields necessarily require a deeper decision tree.

The *Distributed Crossproducting of Field Labels* (*DCFL*) technique is capable of providing the necessary lookup and update performance, while scaling to support large filter sets classifying on additional filter fields [1]. As noted in Section 2, *DCFL* leverages the *match condition redundancy* and *matching set confinement* properties of real filter sets through the use of decomposition, field labeling, and a network of aggregation nodes that performs *crossproducting* in a distributed fashion. The nodes in the aggregation network compute the intersection of the set of labels for the match condition combinations specified by filters and the set of labels for all possible match condition combinations matched by the packet. Using efficient set membership data structures, *DCFL* takes advantage of the distributed embedded memory blocks available in current generation Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs). While *DCFL* achieves high performance and scalability, it requires judicious selection of several design parameters in order to achieve peek performance. For example, the width of the datapath (word size) of the embedded memory blocks has a significant effect on lookup performance and memory efficiency. As the word size increases, the lookup performance increases but the memory efficiency decreases. While it is possible to strike a favorable balance for a given filter set, it may be difficult to provision the word size for a wide variety of filter sets. Other parameters such as the amount of embedded memory and type of data structure employed for each aggregation node also present implementation challenges.

10 Summary

HyperCuts and *DCFL* acquire diametric properties by leveraging filter set structure: greater lookup throughput and scalability, but nondeterministic latency and resource utilization. Due to the resource provisioning challenges and performance volatility, network component and equipment vendors are reluctant to deploy algorithmic solutions. With *Label Encoded Content Addressable Memory* (*LECAM*) we seek to leverage the observed properties of filter sets, decomposition, and field labeling while simplifying the task of hardware implementation. While the resource provisioning challenges are not eliminated, we believe that they are manageable. As the length of search keys and the size of filter sets continues to grow, the performance and scalability advantages of *LECAM* relative to TCAM will become even more pronounced and provide sufficient incentive to designers to tackle the implementation challenges. Commercial router designers increasingly struggle with the board real estate and power consumed by TCAMs on router line cards. We believe *LECAM* strikes a favorable balance among performance, efficiency, scalability, and implementation complexity.

11 Acknowledgments

We would like to thank Dr. Jonathan S. Turner for the wealth of good ideas and advice that he has provided. We would like to thank Venkatachary Srinivasan and Will Eatherton for making real filter sets available for study. Finally, we would like to thank our colleagues in the research community who have provided invaluable feedback on our observations and ideas.

References

[1] D. E. Taylor and J. S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels," in *Proceedings of IEEE Infocom*, March 2005.

[2] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," in *Proceedings of IEEE Infocom*, March 2005.

[3] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," in *ACM Sigcomm*, August 1999.

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

[5] T. V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," in *ACM SIGCOMM'98*, September 1998.

[6] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and Scalable Layer Four Switching," in *ACM Sigcomm*, June 1998.

[7] F. Baboescu and G. Varghese, "Scalable Packet Classification," in *ACM Sigcomm*, August 2001.

[8] A. Feldmann and S. Muthukrishnan, "Tradeoffs for Packet Classification," in *IEEE Infocom*, March 2000.

[9] T. Y. C. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," in *IEEE Infocom*, March 2000.

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

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

[12] A. J. McAulay and P. Francis, “Fast Routing Table Lookup Using CAMs,” in *IEEE Infocom*, 1993.

[13] R. K. Montoye, “Apparatus for Storing “Don’t Care” in a Content Addressable Memory Cell.” United States Patent 5,319,590, June 1994. HaL Computer Systems, Inc.

[14] D. E. Taylor, “Survey & Taxonomy of Packet Classification Techniques,” Tech. Rep. WUCSE-2004-24, Department of Computer Science & Engineering, Washington University in Saint Louis, May 2004.

[15] J. van Lunteren and T. Engbersen, “Fast and scalable packet classification,” *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 560–571, May 2003.

[16] V. Srinivasan, S. Suri, and G. Varghese, “Packet classification using tuple space search,” in *SIGCOMM 99*, pp. 135–146, 1999.

[17] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet Classification Using Multidimensional Cutting,” in *Proceedings of ACM SIGCOMM’03,* August 2003. Karlsruhe, Germany.

[18] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell, “Directions in Packet Classification for Network Processors,” in *Second Workshop on Network Processors (NP2)*, February 2003.

[19] SiberCore Technologies Inc., “SiberCAM Ultra-18M SCT1842.” Product Brief, 2002.

[20] Micron Technology Inc., “Harmony TCAM 1Mb and 2Mb.” Datasheet, January 2003.

[21] IDT, “75k72100 Network Search Engine.” Datasheet, June 2003.

[22] Micron Technology Inc., “36Mb DDR SIO SRAM 2-Word Burst.” Datasheet, December 2002.

[23] E. Spitznagel, D. Taylor, and J. Turner, “Packet Classification Using Extended TCAMs,” in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2003.

[24] E. W. Spitznagel, “CMOS Implementation of a Range Check Circuit,” Tech. Rep. WUCSE-2004-39, Department of Computer Science & Engineering, Washington University in Saint Louis, July 2004.

[25] J. van Lunteren, “Searching very large routing tables in wide embedded memory,” in *Proceedings of IEEE Globecom*, vol. 3, pp. 1615–1619, November 2001.

[26] F. Baboescu, S. Singh, and G. Varghese, “Packet Classification for Core Routers: Is there an alternative to CAMs?,” in *IEEE Infocom*, 2003.

[27] Cisco, “CiscoWorks VPN/Security Management Solution,” tech. rep., Cisco Systems, Inc., 2004.

[28] Lucent, “Lucent Security Management Server: Security, VPN, and QoS Management Solution,” tech. rep., Lucent Technologies Inc., 2004.

[29] D. E. Taylor, *Models, Algorithms, & Architectures for Scalable Packet Classification*. PhD thesis, Department of Computer Science & Engineering, Washington University in Saint Louis, August 2004.

[30] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing table lookups," in *Proceedings of ACM SIGCOMM '97*, pp. 25–36, September 1997.

[31] D. E. Taylor, J. S. Turner, J. W. Lockwood, T. S. Sproull, and D. B. Parlour, "Scalable IP Lookup for Internet Routers," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 522–534, May 2003.

[32] W. N. Eatherton, "Hardware-Based Internet Protocol Prefix Lookups," Master's thesis, Washington University in Saint Louis, May 1999.

[33] IBM and Xilinx, "IBM, Xilinx shake up art of chip design with new custom product." Press Release, June 2002. http://www-3.ibm.com/chips/products/asics/products/cores/efpga.html.

[34] IBM Blue Logic, "Embedded SRAM Selection Guide," November 2002.

Appendix

In order to provided worst-case performance guarantees, we would like to ensure that no packet header results in a matching set containing more labels (matching conditions) than a given threshold, t. Here we describe a generalization of the *field splitting* optimization introduced with the *DCFL* algorithm that provides a way to do this. The idea is to create sets of ranges where the range overlap is less than or equal to t. Note that *field splitting* only applies to prefix and range matching conditions. Since prefixes are just a special case of arbitrary ranges, we can view prefix match conditions as arbitrary ranges on the address space. Consider the construction of the sets of unique matching conditions for each filter field. We can compute the maximum range overlap, m, for the given filter field by adding the set of unique match conditions (ranges) to a segment tree. Given an overlap threshold, t, the number sets required is $C = \lfloor \frac{m}{t-1} \rfloor$. We then create C sets into which we sort the set of unique match conditions. We start by adding wildcards to each of the C sets. For each range $[i : j]$, we identify the set, c_i, containing the minimum number of overlapping ranges using a segment tree constructed from the ranges in the set. We insert $[i : j]$ into set c_i. Once the sorting is complete, we assign locally unique labels to the match conditions in each set. The original match condition specified by the filter field is now identified by a combination of labels: the label for the condition (unique to its set) and the labels for the wildcard conditions in the other sets.