Scaling IP Lookup to Large Databases using the CRAM Lens

Robert Chang and Pradeep Dogga, *University of California, Los Angeles;*
Andy Fingerhut, *Cisco Systems;* Victor Rios and George Varghese,
University of California, Los Angeles

https://www.usenix.org/conference/nsdi25/presentation/chang

This paper is included in the
Proceedings of the 22nd USENIX Symposium on
Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the
22nd USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Scaling IP Lookup to Large Databases using the CRAM Lens

Robert Chang[1] Pradeep Dogga[1] Andy Fingerhut[2] Victor Rios[1] George Varghese[1]

[1]*University of California, Los Angeles* [2]*Cisco Systems*

Abstract

Wide-area scaling trends require new approaches to Internet Protocol (IP) lookup, enabled by modern networking chips such as Intel Tofino [35], AMD Pensando [2], and Nvidia BlueField [55], which provide substantial ternary content-addressable memory (TCAM) and static random-access memory (SRAM). However, designing and evaluating scalable algorithms for these chips is challenging due to hardware-level constraints. To address this, we introduce the CRAM (CAM+RAM) lens, a framework that combines a formal model for evaluating algorithms on modern network processors with a set of optimization idioms. We demonstrate the effectiveness of CRAM by designing and evaluating three new IP lookup schemes: RESAIL, BSIC, and MASHUP. RESAIL enables Tofino-2 to scale to 2.25 million IPv4 prefixes—likely sufficient for the next decade—while a pure TCAM approach supports only 250k prefixes, just 27% of the current global IPv4 routing table. Similarly, BSIC scales to 390k IPv6 prefixes on Tofino-2, supporting 3.2 times as many prefixes as a pure TCAM implementation. In contrast, existing state-of-the-art algorithms, SAIL [83] for IPv4 and HI-BST [65] for IPv6, scale to considerably smaller sizes on Tofino-2.

1 Introduction

For many, Internet Protocol (IP) lookup is considered a challenge of the past. With over 40 years of research and hundreds of papers (e.g., [7, 19, 21, 22, 29, 45, 65, 83, 89]) focused on supporting IP lookup at scale, numerous schemes have been developed—some of which have been in practical use for two decades. However, these classical approaches are *single-resource* solutions, designed for conventional switch chip architectures that provided either specialized hardware like ternary content-addressable memory (TCAM) or commodity random-access memory (RAM), such as on-chip static RAM (SRAM) coupled with off-chip dynamic RAM (DRAM), but not both. TCAM enables parallel searches across wildcarded entries in a single clock cycle but requires three times more transistors per bit than SRAM and consumes hundreds of watts [5]. RAM-based approaches are cheaper but require additional complexity and memory compared to pure TCAM solutions. Thus far, commercial switch chip vendors have scaled these *single-resource* solutions by increasing hardware resources.

In this paper, we contend that it is important to reconsider IP lookup due to the continued growth of lookup tables and a recent inflection point in network hardware. A slew of new application-specific integrated circuits (ASICs), such as Intel Tofino [35], AMD Pensando [2], and Nvidia BlueField [55], have transformed the networking chip market [58]. These chips are built on two modern packet processing architectures—Reconfigurable Match-Action Tables (RMT) [9] and disaggregated RMT (dRMT) [15]—which consist of match-action processors with access to large amounts of *both* TCAM and SRAM. We review these architectures at the start of §2. This leads us to our central question: *How can we leverage modern networking chips, utilizing both TCAM and SRAM, to develop new IP lookup algorithms that scale to larger databases than classical single-resource solutions?*

Two main challenges make designing scalable algorithms for RMT and dRMT chips difficult: **(1)** Lack of an abstract model for evaluating and comparing algorithms. Chip-specific arcana such as memory allocation, metadata storage, and action bits must be carefully considered. **(2)** Large but finite resources. While TCAM, SRAM, and pipeline stages are available in substantial amounts, they require careful algorithm design to scale effectively.

Solution: We introduce the CRAM (CAM+RAM) lens, an abstract model of modern packet processing architectures, such as RMT and dRMT, paired with a set of optimization idioms. The CRAM model enables us to estimate algorithm scalability using higher-order space and time metrics, without requiring simulation of ASIC-specific details such as TCAM block sizes, SRAM page sizes, and per-stage memory. The CRAM model goes beyond classical models such as random-access machine (RAM) [23] and parallel RAM (PRAM) [41] by adding TCAM operations and using match-action dependencies to measure time complexity. The optimization idioms provide eight strategies for designing scalable algorithms.

CRAM can be generalized to other hardware architectures, such as smart network interface cards (SmartNICs) [2, 32, 55] and field-programmable gate arrays (FPGAs) [3, 34], and applied to broader network applications like packet classification [30, 44, 74] and in-network machine learning (ML) [73, 91, 93]. However, this is not the focus of our paper. For completeness, we briefly discuss these extensions in §2.4 and §2.5. Instead, we concentrate on applying CRAM to *IP lookup* because of the following observations:

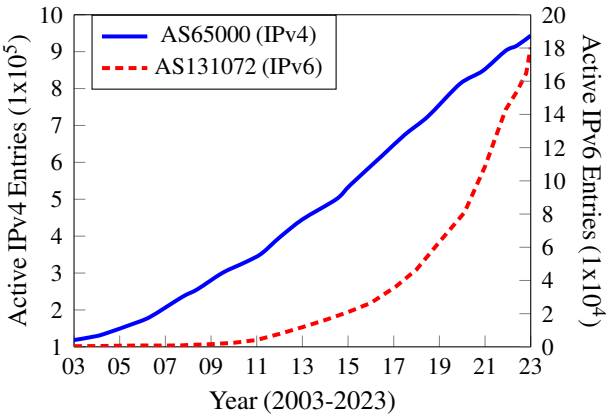


Figure 1: BGP routing table size over the past two decades

O1. Continued IPv4 growth: Over the past two decades, the global IPv4 routing table has grown *linearly* [26, 27], doubling in size every decade (Figure 1). If this trend continues, the IPv4 table could reach two million entries by 2033.

O2. Rapid IPv6 deployment: In the same period, the global IPv6 routing table has grown *exponentially* [25, 28], doubling every three years (Figure 1). Even if growth slows to a linear rate, the IPv6 table could still reach half a million entries by 2033. IPv6 prefixes are also four times wider, though typically, only the first 64 bits are used for global routing.

O3. Virtual private networks (VPNs): Some routers maintain hundreds of VPN routing tables. On such devices, publicly available routing tables account for only a fraction of the total capacity required.

O4. Other tasks: Routers need table memory for additional tasks such as network address translation (NAT) and firewalls. Minimizing the memory used for forwarding allows more features to fit on a chip.

CRAM enables us to rethink pre-existing IP lookup schemes, such as SAIL [83], DXR [89], and multibit tries, to develop new algorithms that scale to larger databases. We start with the best-of-breed algorithms from three classic IP lookup approaches—search on prefix *lengths*, search on prefix *ranges*, and trie-based search. We derive new algorithms using the optimization idioms (§2.2) and predict their scalability with the CRAM model (§2.1). Each new algorithm—rethinking SAIL (RESAIL), Binary Search with Initial CAM (BSIC), and MASHUP—offers unique strengths for different settings.

We chose Intel Tofino-2, an RMT switch chip, for our experiments because we had access to its development environment. While Intel recently announced it will not develop new Tofino models, it remains committed to supporting Tofino and Tofino-2 [53]. We expect our results to hold for dRMT, as RMT is a stricter version of dRMT with additional access restrictions.

This paper makes the following contributions:

1. IP Lookup Algorithms: Three new scalable IP lookup algorithms—RESAIL, BSIC, and MASHUP.

2. CRAM Model: An abstract model for quickly estimating the scalability of packet processing algorithms before detailed implementation.

3. Optimization Idioms: Eight design strategies for minimizing TCAM, SRAM, and pipeline stages.

4. Evaluation: Simulations and Tofino-2 implementations achieve 9X (IPv4) and 3X (IPv6) improvements over pure TCAM solutions, enabling scalability for the next decade.

The remainder of this paper is organized as follows. §2 introduces the CRAM model and optimization idioms, followed by an overview of the idioms in action. It also briefly explores other hardware architectures, broader network applications, and algorithmic requirements. We design three new IP lookup algorithms in § 3, § 4, and § 5. § 6 presents simulation and implementation results, followed by scalability experiments in §7. §8 evaluates the predictive accuracy of the CRAM model, and §9 surveys related work. We conclude in §10.

2 The CRAM Lens

We now formally introduce the CRAM model, list eight optimization idioms, and preview the idioms in action. Additionally, we briefly discuss how CRAM can be generalized to other hardware architectures, applied to broader network applications, and adapted to algorithmic requirements.

The CRAM model abstracts two modern packet processing architectures, RMT [9] and dRMT [15]. Figures depicting both can be found in Appendix A.1. A list of known RMT and dRMT implementations is provided in Appendix A.2.

RMT: RMT is a sequential pipeline architecture of match-action stages. TCAM and SRAM are partitioned among stages such that a stage cannot access the memory of other stages.

dRMT: In contrast, dRMT features programmable processors that execute match-action operations in any order. It disaggregates memory from processors by relocating TCAM and SRAM into a shared external pool.

2.1 The CRAM Model

The CRAM model adds two extensions to the RAM model [23]: first, the ability to perform an SRAM *or* TCAM table lookup; second, an explicit dependency structure between steps (as in RMT compilers [37]) that models the ability to execute multiple steps in parallel. Our goal is for the memory and run time measures of a CRAM model program to be within a small constant factor of the measures for actual hardware implementations.

Thus, a CRAM model program is parameterized by:

- A register size w, and a set R of (w-bit) registers. Let C denote the set of w-bit integers in the range $[0, 2^w - 1]$.
- Sets of unary ($Uops$) and binary ($Bops$) operators on w-bit values, e.g., $Uops = \{+, -, \sim, !\}$ and $Bops = \{+, -, <<, >>, ==, !=, <, \leq, >, \geq, \&, |, \hat{}, \&\&, ||\}$, with behavior as defined in languages like Java and P4 [58].

A CRAM model program consists of a parser P, a deparser D, and a directed acyclic graph G comprised of *steps*. A state S is a function from R to C. P is a function from all bit sequences representing packets to an initial state. D is a function from a final state back to all bit sequences representing packets.

A *step* may optionally begin with a single table lookup operation. A table t consists of a match kind (exact or ternary), a key selector function K_t, a maximum number of entries n_t, and a default value Z_t. K_t's result is a sequence of k_t bits, each representing a chosen bit position within one register of R. An entry e contains a key and associated data, with the d_t bits of associated data stored in a set of w-bit registers A_t.

For an exact match table, the key is a k_t-bit integer. A special case arises for exact match tables with $n_t = 2^{k_t}$, in which the key does not need to be explicitly stored, as it can be used to directly index into the table. For a ternary match table, the key is a pair of k_t-bit integers, a value v_e and a mask m_e, plus an integer priority p_e. All keys in the same table must be distinct.

A step consists of an optional table t followed by a sequence of statements in the form $if(cond) : dest = expr$. Here, $dest$ is an element of R, $expr$ contains a single unary or binary operator with operands from $R \cup A_t \cup C$, and $cond$ is a potentially nested expression with operands from $R \cup A_t \cup C$. No data dependencies are allowed within this sequence, i.e., for any statement in the sequence that assigns a value to $r \in R$, r may not appear in $cond$ or $expr$ of any later statements. This enables all statements within a step to be executed in parallel.

A step reads register r if any bit of r appears in the output of its key selector function K_t, or as part of $cond$ or $expr$ in any of its statements. A step writes register r if r appears as $dest$ in any of its statements.

For all steps u and v in G, if u writes r and v reads or writes r, then there must be a directed path (u,v) or (v,u). This prevents u and v from being executed in parallel. This condition must hold for all registers. A directed path (u,v) indicates that step u must be executed before step v. If there is no directed path between two steps, they may be executed in parallel.

The CRAM model introduces a set of higher-order space and time metrics. The memory footprint of a CRAM model program is evaluated by calculating the total TCAM and SRAM bits across all tables t in G. In a ternary (exact) match table, the memory used for the keys is $n_t k_t$ TCAM (SRAM) bits. For ternary match tables, we only count the v_e component of the key, as these are the logical bits involved in the match. For both types of tables, the memory used for the associated data is $n_t d_t$ SRAM bits. To compute the overall TCAM and SRAM totals, simply sum the bits used across all tables. The latency of a CRAM model program is evaluated by determining the number of steps (nodes) in the longest directed path in G.

2.2 Optimization Idioms

The following idioms can be applied together in various combinations to achieve different space-time trade-offs:

I1. Compress with TCAM: Entries containing wildcards must be expanded to fit into SRAM. For example, the prefix 1** would be stored as 100, 101, 110, and 111. However, by utilizing TCAM, these four SRAM entries can be compressed into a single TCAM entry (1**), thus saving nine bits.

I2. Expand to SRAM: In the dual of *I1*, replace a TCAM block with SRAM if the expanded forms of its prefixes are less than a small constant factor c of the original TCAM entries. We choose $c = 3$ because TCAM requires three times more transistors per bit than SRAM [82].

I3. Compress with SRAM: Despite their high memory cost, directly indexed data structures such as next hop arrays are used because they avoid the extra instructions needed for hashing. However, since most RMT and dRMT ASICs are designed with the cost of performing SRAM-based lookups—whether by hashing or direct indexing—being exactly the same, it is often more advantageous to use compressed forms of SRAM storage such as hash tables instead.

I4. Strategic Cutting: If several entries at a given node share a common prefix, we can save memory by strategically cutting at the bit position where the shared prefix ends, storing only one copy of the repeated bits. While this is how multibit tries [70] work, we extend the concept to TCAM nodes.

I5. Table Coalescing: To reduce memory waste, minimally populated logical tables can be coalesced in shared physical TCAM blocks or SRAM pages. They can be differentiated with tag bits [66]. Although tagging increases the lookup key width, it minimizes physical TCAM and SRAM fragmentation.

I6. Look-aside TCAM: IP lookup schemes are often optimized around common cases such as 24-bit IPv4 prefixes. As a result, uncommon entries (e.g., extremely short or long prefixes) tend to require undue computational or storage costs. We address this by moving the special prefixes into a separate look-aside TCAM that can be trivially searched in parallel.

I7. Step Reduction: A program's number of steps can be reduced by leveraging match-action unit (MAU) parallelism to consolidate data-independent lookups into a single stage.

I8. Memory Fan-out: In traditional RAM model architectures, a lookup table can be accessed multiple times per packet. However, many RMT and dRMT chips restrict each table to one memory access per packet. To address this limitation, we split the original table by fanning out its contents and storing entries accessed by different lookups in separate tables.

2.3 Idioms in Action

We briefly preview the idioms in action for three fundamental classes of IP lookup: search on prefix *lengths*, search on prefix *ranges*, and trie-based search.

From SAIL to RESAIL: Search on prefix *lengths* [10, 21, 40, 77, 83] splits IP lookup into two sub-problems: finding the *length* of the longest match and retrieving the *next hop*. SAIL [83], the best performing IPv4 lookup scheme in hardware settings with fast on-chip SRAM

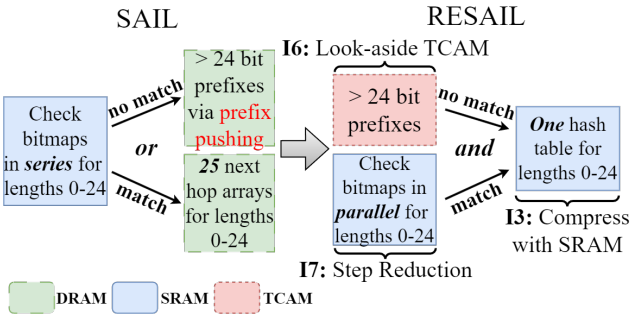


Figure 2: From SAIL to RESAIL via CRAM idioms

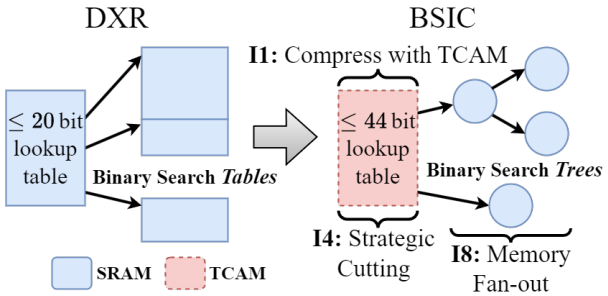


Figure 3: From DXR to BSIC via CRAM idioms

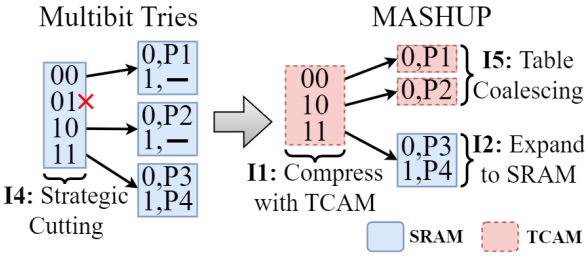


Figure 4: From multibit tries to MASHUP via CRAM idioms

and cheap off-chip DRAM, uses a bitmap of length 2^L to determine whether there is a matching prefix of length L. This works well for prefixes of up to length 24 (the vast majority); for prefixes longer than 24 bits, SAIL uses a complex scheme called pivot pushing that requires excessive prefix expansion [70]. In Figure 2, to obtain RESAIL (rethinking SAIL), we: *I6)* Move the small number of prefixes longer than 24 bits into a separate look-aside TCAM. *I3)* Compress all the next hop arrays into a single hash table using a standard encoding trick [76]. *I7)* Use MAU (stage) parallelism to reduce the number of steps by performing the bitmap lookups in parallel.

From DXR to BSIC: Search on prefix *ranges* [29,45,72,85,89] represents prefixes as range endpoint pairs (e.g., 0**→[000, 011]). Finding the longest matching prefix becomes equivalent to finding the smallest range that encompasses the lookup key. DXR [89], the fastest IPv4 software implementation of range-based searches, uses an initial lookup table to split the search space into multiple smaller binary search tables. In Figure 3, to obtain BSIC (Binary Search with Initial CAM), we: *I1)* Replace the SRAM-based initial lookup table, which supports up to 20-bit prefixes due to direct indexing, with a TCAM-based table that can store prefixes of up to 44 bits (Tofino-2 TCAM block width). *I8)* Replace each binary search *table* with a binary search *tree* that can be fanned out across stages. *I4)* Strategically cut the initial lookup table to balance TCAM required against binary search depth.

From multibit tries to MASHUP: For trie-based search [7,8,19,22,31,54,70], we specifically focus on multibit tries—tries that examine multiple bits, known as a stride, per lookup. We do not consider state-of-the-art *compressed* trie schemes like Poptrie [7] and Tree Bitmap [22], because in the CRAM model, one can directly compress with TCAM without the extra computational and storage costs of bitmap compression. Figure 4 shows a standard multibit trie for the prefixes P1 = 000*, P2 = 100*, P3 = 110*, and P4 = 111*, with a 2-bit stride at the root and a 1-bit stride at the next level—chosen by strategic cutting *(I4)* to minimize the number of downstream pointers. In Figure 4, to obtain MASHUP (mashup of CAM and RAM nodes), we: *I1)* Replace the SRAM root node with a TCAM node to eliminate the empty 01 entry, and do the same for the two upper-right nodes. *I2)* Leave the bottom-right node as SRAM, as it has no wasted space. *I5)* Coalesce the two upper-right TCAM nodes using tag bits (not shown). While the improvement is minimal in this simple example, § 5.1 demonstrates significant gains for large databases.

Although the CRAM versions of these classical schemes may seem simple, they require new algorithms to determine, for example, where to make strategic cuts and which nodes to coalesce. We elaborate on these details when describing RESAIL, BSIC, and MASHUP in §3, §4, and §5, respectively.

2.4 CRAM for other Hardware Architectures

CRAM can be generalized to hardware architectures beyond RMT and dRMT as follows: the space and time metrics of an algorithm specified in the CRAM model serve as lower bounds on the corresponding costs in any implementation, whether in programmable switch ASICs [16, 35, 52, 56], SmartNICs [2, 32, 55], FPGAs [3, 34], or purpose-built fixed-function ASICs designed solely to execute that algorithm. A faithful implementation of a CRAM algorithm achieves a minimum latency equal to that of the longest (critical) path in its directed acyclic graph. While an implementation may have a longer latency, it cannot be shorter. Similarly, the number of bits required may match or exceed the amount specified by the CRAM model, but it cannot be less.

2.5 CRAM for broader Network Applications

Although this paper focuses on IP lookup, we believe CRAM applies to other memory-intensive network applications. These include packet classification (with Access Control Lists (ACLs) and Quality of Service (QoS) as specific

instances), measurement algorithms (such as sketching), regular expression matching, and in-network ML.

In packet classification [30, 44, 74], packet headers are matched against a classifier, where the highest-priority match determines whether to allow or deny traffic, enforce a QoS policy, or apply a custom action. Measurement algorithms [17, 42, 68, 87], by contrast, dynamically build and update a stateful database that tracks network statistics such as per-flow counters, traffic volume, or frequency estimates. Regular expression matching [43, 50, 78] compares unstructured data streams against a database of predefined patterns, often represented as finite automata [14], to detect keywords, signatures, and anomalies. Lastly, in-network ML [73, 91, 93] performs inference by matching a feature vector against a classification model database, which contains decision rules mapping extracted feature values to inference labels.

In practice, these applications often rely on combinations of common data structures, such as decision trees, Bloom filters [11], tries, hash tables, and bitmaps, many of which we demonstrate how to optimize with CRAM. Consequently, our optimization idioms naturally extend beyond IP lookup. For example, the careful balancing of TCAM compression (*I1*) and SRAM expansion (*I2*) used in MASHUP to create a hybrid trie, can similarly be applied to packet classification [47, 67] and in-network ML [39, 79] algorithms that rely on decision trees. Likewise, the look-aside TCAM (*I6*) in RESAIL, which captures longer prefixes, can serve a similar role in offloading other specialized cases, such as multi-field wildcard classification rules [30], heavy-hitter flows with rare protocols [68], multi-line attack patterns [78], and fast-patch updates for classification models [92]. A more in-depth exploration of CRAM's broader applicability is left for future work.

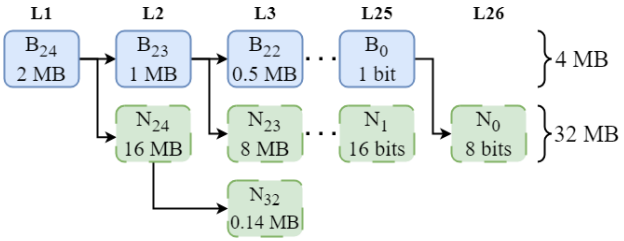
2.6 Other Algorithmic Requirements

Certain algorithmic requirements introduce additional considerations. We briefly examine how CRAM applies to algorithms that require atomic memory updates, stateful data plane operations, and pseudo-random keys.

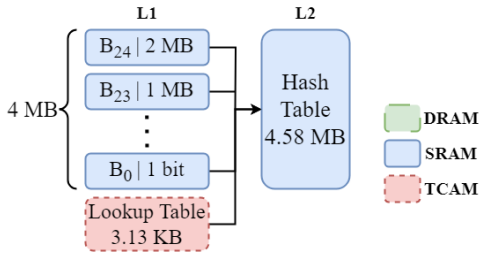
Atomic memory updates: CRAM neither facilitates nor hinders an algorithm's ability to support atomic updates [14, 46, 61]. The CRAM model provides an abstraction to estimate *performance*, not an execution model to predict *implementation feasibility*. If an algorithm requiring atomic updates can be implemented, the CRAM model should accurately predict latency and memory.

Stateful data plane operations: P4 register arrays are the primary mechanism for stateful operations in the data plane, as used in [6, 13, 68]. Stateful operations can be incorporated into the CRAM model by introducing a new SRAM-based register match table, and counting these memory bits separately alongside regular TCAM and SRAM bits.

Pseudo-random keys: For algorithms with pseudo-random keys [12, 36, 86], the efficacy of some CRAM optimization idioms is clearly reduced, as uniformly distributed random bytes are difficult, if not impossible, to compress. In contrast, other idioms are unaffected by key distribution. Specifically, table coalescing (*I5*), look-aside TCAM (*I6*), step reduction (*I7*), and memory fan-out (*I8*) can still be applied. Nevertheless, the benefits of the CRAM model are likely to be significantly diminished, as lack of compression eliminates a major source of optimization.



(a) CRAM model representation of SAIL



(b) CRAM model representation of RESAIL (*min_bmp*=0)

Figure 5: SAIL vs RESAIL for IPv4 prefixes in AS65000

3 RESAIL

By applying the optimization idioms to SAIL [83], we create a new IPv4 CRAM lookup algorithm called RESAIL. Refer to Figure 2 for the intuition to which we now add details.

SAIL Review: SAIL designates 24 as a pivot level and divides the forwarding information base (FIB) into short prefixes ($\leq$24) and long prefixes (>24). SAIL determines whether there is a length-i match for $i \leq 24$ by consulting a bitmap B_i of size 2^i in which bit p is set if and only if p is a length-i prefix in the FIB. Since the total memory footprint of the bitmaps is 4 MB, they are stored in on-chip SRAM. If a match is found in B_i, the next hop is retrieved by directly indexing into a next hop array N_i of size 2^i located in off-chip DRAM. SAIL handles prefixes of length $i > 24$ by using a complex scheme called pivot pushing that expands [70] them to 32-bit entries in N_{32}.

3.1 Applying the Optimization Idioms

We show the CRAM derivation of RESAIL using the IPv4 AS65000 BGP routing table (Sep 2023). Start with the CRAM model representation of SAIL in Figure 5a. To obtain RESAIL in Figure 5b, use the idioms as follows:

Entry	Prefix (Ternary)	Output Port
1	010100**	A
2	011*****	B
3	100100**	C
4	100101**	D
5	10010100	A
6	10011010	B
7	10011011	C
8	10100011	A

Table 1: Example routing table

Index	Key	Value
0	1001001	C
1	0101001	A
2	0111000	B
3	-	-
4	1001011	D

Table 2: Hash table for Table 1 (pivot level = 6)

1. Observe a large number (26) of data dependencies between the bitmaps and next hop arrays. This makes sense in the RAM model as it enables early exits which reduce average execution time. However, these are false dependencies because their lookup keys can be computed in parallel. Therefore, we apply step reduction (*I7*) to reduce all the bitmap and next hop array lookups into a single step. The next hop can then be determined by taking the highest priority result.

2. SAIL relies on a significant amount of DRAM (32 MB) for its directly indexed next hop arrays. Since DRAM is not available in CRAM, we replace the next hop arrays with a more compact data structure by either compressing with TCAM (*I1*) or compressing with SRAM (*I3*). Since the 25% memory penalty of d-left hashing [10] is less expensive than TCAM's 3X higher area cost, we compress with SRAM by replacing the next hop arrays with a single SRAM-based hash table.

3. SAIL uses a special next hop array N_{32} for prefixes of length >24, which are very uncommon. Its entries are prefix expanded to 32 bits. In the worst case, a single prefix may be expanded into 2^8 duplicate next hops. We address this memory inefficiency by replacing N_{32} with a look-aside TCAM (*I6*) that can store prefixes of length >24 without additional expansion.

4. The number of bitmaps serves as a trade-off between the amount of parallelism required and the hash table's memory footprint. For RESAIL, we introduce a parameter *min_bmp* that represents the smallest bitmap available. In Figure 5b, *min_bmp* is 0 which means there are a total of 25 bitmaps from B_{24} down to B_0. Increasing *min_bmp* reduces the number of parallel lookups at the cost of increased SRAM usage.

3.2 Building the Data Structures

Look-aside TCAM: Given a routing database, add all prefixes longer than 24 bits to the look-aside TCAM. Since there are very few IPv4 prefixes of length >24, little TCAM is used.

Bitmaps: For $i = 24$ down to $i = min\_bmp$, construct a bitmap (B_i) of length 2^i such that every prefix of length i in the routing database is marked as a 1 at the corresponding index. If *min_bmp* is not equal to 0, use prefix expansion [70] to combine B_0 to B_{min_bmp-1} into B_{min_bmp}. Start with length min_bmp-1 prefixes and work down linearly to length 0. A bit in B_{min_bmp} is flipped from 0 to 1 only if the bit is already a 0. This prevents incorrectly overwriting longer prefixes.

Hash Table: We use d-left [10] for the hash table because it has a low probability of collision even when the ratio of entries to memory is as high as 80%. Hashing the matched prefix directly would require a separate hash table for each length from *min_bmp* to 24, greatly fragmenting memory. Instead, RESAIL uses a standard trick [76] we call bit marking that enables us to generate hash keys of a fixed length. When an entry is added to bitmap B_i, its unique 25-bit hash key is produced by appending a 1 and left shifting by $24-i$ bits. Each hash key is paired with its next hop and inserted into the hash table. The boundary of each prefix can be determined by scanning from the right for the first 1. In effect, bit marking removes the need for multiple hash tables.

Table 2 shows a hash table using 7-bit hash keys for Table 1. For simplicity, this example assumes a pivot level of 6 and a maximum prefix length of 8. Since entries 5-8 from Table 1 are longer than the pivot length, they are not placed into the hash table (they are in the look-aside TCAM instead). The hash table has a size of 5 due to d-left's 25% memory penalty. 011, a 3-bit entry, is appended with a 1 and left shifted 3 times, thus resulting in the hash key 0111000.

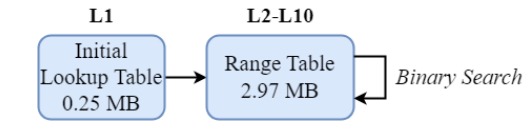
3.3 Performing Lookups

Start by performing two sets of lookups in parallel: **(1)** In the look-aside TCAM, perform a longest prefix match with the full 32-bit IPv4 address. **(2)** From B_{24} ($i = 24$) down to B_{min_bmp}, perform exact match lookups using the first i bits of the destination address to directly index into B_i.

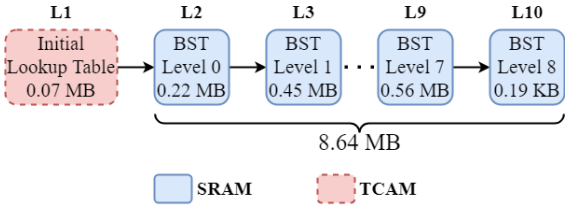
If a match is found in the look-aside TCAM, return its associated next hop. Otherwise, take the longest match across all bitmaps and generate its 25-bit hash key by bit marking.

At this step, either the next hop for a prefix match greater than 24 bits has been found or the final hash key has been created. In the latter case, use the hash key to perform an exact match lookup into the hash table to retrieve the next hop.

Algorithm 1 in Appendix A.5 contains pseudocode for RESAIL lookups. Appendix A.3.1 describes incremental updates, deletions, and insertions in RESAIL.



(a) CRAM model representation of DXR (k=16)



(b) CRAM model representation of BSIC (k=16)

Figure 6: DXR vs BSIC for IPv4 prefixes in AS65000

4 BSIC

By applying the optimization idioms to DXR [89], we create a new CRAM lookup algorithm called BSIC, capable of supporting both IPv4 and IPv6. Refer to Figure 3 for the intuition to which we now add details.

DXR Review: Inspired by [29] and [45], DXR performs binary search on range endpoints using a range table (binary search table). To reduce the depth of binary search, DXR uses an initial lookup table directly indexed by the first k bits of the address. The lookup table returns a pointer to the subsection of the range table in which binary search will be performed. DXR adds two optimizations: **(1)** Merging neighboring ranges that point to the same next hop. **(2)** Discarding right endpoints.

4.1 Applying the Optimization Idioms

We show the CRAM derivation of BSIC using the IPv4 AS65000 BGP routing table (Sep 2023). Start with the CRAM model representation of DXR in Figure 6a. As recommended by [89], set $k=16$ (D16R) for the best IPv4 results. To obtain BSIC in Figure 6b, use the idioms as follows:

1. Since DXR's initial lookup table relies on direct indexing, leaving many entries unused, we compress with TCAM (*I1*). Replacing the SRAM-based initial lookup table with a TCAM-based one reduces its memory consumption by over 3X, from 0.25 MB of SRAM to 0.07 MB of TCAM.

2. In DXR, the range table is repeatedly accessed during binary search. Since lookup tables are limited to a single access per packet in the CRAM model, the range table must be split up. We do so through memory fan-out (*I8*). By converting the range table into multiple binary search trees (BSTs) and distributing search levels across separate tables accessed at different steps, we ensure each table is visited at most once per packet. However, this greatly increases the amount of memory needed because every internal node has to store up to two pointers. In Figure 6, DXR's range table uses only 2.97 MB of SRAM while BSIC's BST levels use 8.64 MB of SRAM—a 2.9X increase. Although costly, memory fan-out is essential because duplicating the entire range table for each search level would require an infeasible amount of SRAM (26.73 MB).

Key	Value	BST Entries (for reference)
0101	Pointer to BST 1	00**
011*	Next Hop B	-
1001	Pointer to BST 2	00**, 01**, 0100, 1010, 1011
1010	Pointer to BST 3	0011

Table 3: Initial lookup table for Table 1 ($k=4$)

3. The parameter k is a strategic cut (*I4*) that balances memory usage in the initial lookup table against the number of required BST levels. Due to the high memory cost of direct indexing, DXR's SRAM-based lookup table is limited to $k<=20$. For example, if $k=24$, DXR's initial lookup table would consume 64 MB of SRAM. To effectively support IPv6, which has longer prefixes, a larger k value is required. Since TCAM can store wildcard entries without prefix expansion, BSIC's TCAM-based initial lookup table can use much larger k values, up to the underlying TCAM block width ($k=44$ for Tofino-2).

4.2 Building the Data Structures

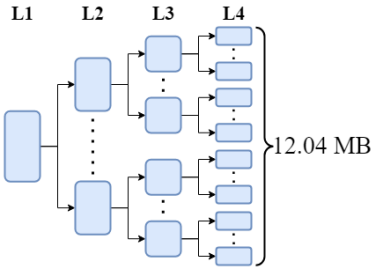
Initial Lookup Table: Given a routing database and a slice size k, populate the initial lookup table by storing all prefixes as unique k-length slices. Duplicate slices are condensed into one entry. Three cases arise when adding a prefix p of length l:

1. If $l < k$, pad p with $k-l$ wildcard (*) bits. Its associated table value is p's next hop.
2. If $l == k$, do not modify p. If there are longer prefixes that share the same k-length slice as p, its associated table value is a pointer to the corresponding BST's root node. Otherwise, its associated table value is p's next hop.
3. If $l > k$, trim p down to k bits. Its associated table value is a pointer to the corresponding BST's root node.

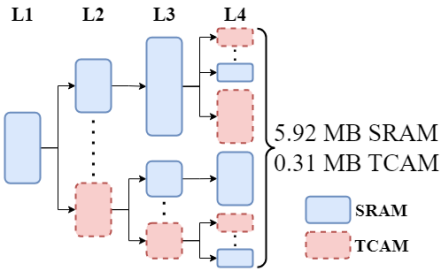
Table 3 shows an initial lookup table with k=4 created using Table 1. The maximum prefix length is 8. The Key column contains all the k-length slices while the Value column stores the associated pointers and next hops. The BST Entries column shows the prefix segments that are pointed to by the slices. Since entries 3-7 in Table 1 share the same k-length slice, they are condensed into a single key 1001 that points to BST 2.

Binary Search Trees (BSTs): To create a BST for a given lookup table entry, identify all prefixes in the database that match the entry up to the kth bit. For all such prefixes, store the remaining bits and next hops as tuples in a list. Take the list of tuples and perform the range expansion and optimizations described in [89]. We defer these details to Appendix A.4.

Use the resulting list of left endpoints to construct a BST in which every node contains four fields: pointers to the left and



(a) CRAM model representation of multibit trie (16-4-4-8)



(b) CRAM model representation of MASHUP (16-4-4-8)

Figure 7: Multibit trie vs MASHUP for IPv4 prefixes in AS65000

right child, the next hop, and the left endpoint itself. Repeat this process for all lookup table entries containing pointers.

4.3 Performing Lookups

Start in the initial lookup table by performing a longest prefix match using the first k bits of the destination address. If either a next hop is returned or a miss occurs, search terminates. If a pointer to a BST is returned, follow the pointer to the corresponding root node and form the next search key by extracting the remaining bits of the destination address. Once at a node, perform standard binary search using the search key.

Algorithm 2 in Appendix A.5 contains pseudocode for BSIC lookups. Appendix A.3.2 describes incremental updates, deletions, and insertions in BSIC.

5 MASHUP

By applying the optimization idioms to multibit tries [70], we create a new CRAM lookup algorithm called MASHUP, capable of supporting both IPv4 and IPv6. Refer to Figure 4 for the intuition to which we now add details.

Multibit Trie Review: Multibit tries are search tries that examine multiple bits per lookup, known as a stride. Reducing the number of strides decreases the number of worst-case memory accesses, but increases prefix expansion and memory usage. We assume each tree level has exactly one stride.

5.1 Applying the Optimization Idioms

We show the CRAM derivation of MASHUP using the IPv4 AS65000 BGP routing table (Sep 2023). Start with the CRAM model representation of a multibit trie in Figure 7a. We find strides 16-4-4-8 yield the best IPv4 results (§6.3). To obtain MASHUP in Figure 7b, use the idioms as follows:

1. For each trie node, we consider both compressing with TCAM (*I1*) and expanding to SRAM (*I2*). If the increase in memory due to prefix expansion [70] is less than 3X, we use SRAM. Otherwise, we use TCAM. This results in a hybrid trie with both TCAM and SRAM nodes, as seen in Figure 7b.

2. Once the trie is hybridized, apply table coalescing (*I5*) by merging partially filled nodes[1] of the same memory type into super-tables, compactly mapping them onto contiguous TCAM blocks or SRAM pages with minimal fragmentation. This requires prepending entries with a tag [66] to distinguish between logical tables. A tag of width x can coalesce 2^x logical tables into one super-table. The combination of node hybridization and table coalescing reduces SRAM usage from 12.04 MB to 5.92 MB at the cost of 0.31 MB of TCAM (Figure 7).

3. The set of strides is a parameter that serves as a strategic cut (*I4*). For a given set of strides, the trie's memory overhead is directly proportional to the number of internal pointers. A simple method for choosing strides, explained in §6.3, is to analyze the database's prefix length distribution.

We omit standard algorithms for building the MASHUP trie, as the process is identical to constructing a multibit trie, which has been extensively studied in prior work [70, 76].

5.2 Performing Lookups

Let S_i represent the i-th stride. Start in the root node by performing a lookup with the first S_0 bits of the destination address. If the current node is TCAM, perform a longest prefix match. Otherwise, perform an exact match. If a miss occurs, terminate the search. If a hit occurs, three values may be returned: a next hop, a pointer to the next node, and a unique tag. If a next hop is returned, save it. Form the lookup key for level i by extracting the next S_i bits of the destination address and prepending the current tag. Repeat the lookup process until either a leaf node is reached or a miss occurs. Upon termination, return the saved next hop.

Algorithm 3 in Appendix A.5 contains pseudocode for MASHUP lookups. Appendix A.3.3 describes incremental updates, deletions, and insertions in MASHUP.

6 Results

In this section, we introduce the databases, target implementations, and parameter values that we selected for our experiments. We use the CRAM metrics to determine the

[1] For MASHUP in §6.4, we greedily fill the largest tables with the smallest ones. This approach is easy to implement, but possibly suboptimal.

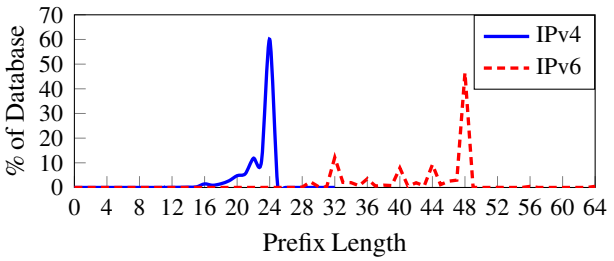


Figure 8: IPv4 and IPv6 prefix length distributions in AS65000 and AS131072, respectively, for September 2023

best IPv4 (RESAIL) and IPv6 (BSIC) algorithms *before* implementation. We then compare the resource utilizations of the best CRAM algorithms with those of state-of-the-art IP lookup schemes *after* implementation.

6.1 Databases

For IPv4, we used the AS65000 BGP routing table (Sep. 2023) [27] which has close to 930k IPv4 prefixes. For IPv6, we used the AS131072 BGP routing table (Sep. 2023) [28] which has close to 190k IPv6 prefixes. Figure 8 shows their prefix length distributions. We identify three key patterns:

P1. Prefix distributions have major and minor spikes. For IPv4, there is a major spike at length 24 and minor spikes at lengths 16, 20, and 22. For IPv6, there is a major spike at length 48 and minor spikes at lengths 28, 32, 36, 40, and 44.

P2. The majority of IPv4 prefixes are longer than 12 bits.

P3. The majority of IPv6 prefixes are longer than 28 bits.

6.2 Target Implementations

We obtained results for two different targets: an ideal RMT chip and Intel Tofino-2 (also RMT). Since RMT is a stricter variant of dRMT with additional access restrictions, we expect our RMT results to be reproducible on a dRMT chip.

Ideal RMT Chip (Simulation): We define an ideal RMT chip to be an RMT chip with Tofino-2 specifications (same memory, number of stages, etc.) [57] that can achieve 100% SRAM utilization and perform at least two dependent ALU operations per stage. The resource utilization for an ideal RMT chip is obtained through simulation by using Tofino-2 SRAM page (128x1024b) and TCAM block (44x512b) sizes. If the number of TCAM blocks or SRAM pages used by a table exceeds the amount available in a MAU (stage), the table is simply partitioned across multiple MAUs. Since Tofino-2 has 20 MAUs, results that require over 20 are considered infeasible.

Intel Tofino-2 (Implementation): The resource utilization for Tofino-2 is obtained through implementation. We implement the best CRAM algorithms using P4 and compile them with the Intel P4 compiler. P4 Insight [33] then outputs detailed resource mappings and visualizations specific to Tofino-2.

Scheme	TCAM Bits	SRAM Bits	Steps
MashUp (16-4-4-8)	0.31 MB	5.92 MB	4
BSIC (*k*=16)	0.07 MB	8.64 MB	10
RESAIL (*min_bmp*=13)	3.13 KB	8.58 MB	2

Table 4: CRAM metrics for IPv4 prefixes in AS65000

Scheme	TCAM Bits	SRAM Bits	Steps
MashUp (20-12-16-16)	0.32 MB	0.77 MB	4
BSIC (*k*=24)	0.02 MB	3.18 MB	14

Table 5: CRAM metrics for IPv6 prefixes in AS131072

6.3 Parameter Values

We choose parameter values based on observations from §6.1.

RESAIL's key parameter is *min_bmp*, the smallest bitmap available. We choose $min\_bmp = 13$ because there are so few IPv4 prefixes shorter than 13 bits **(P2)**, thus minimizing the amount of prefix expansion needed.

BSIC's key parameter is k, the initial slice size. As recommended by [89], for IPv4, we choose $k = 16$. For IPv6, we choose $k = 24$ because most IPv6 prefixes are longer than 28 bits **(P3)**. Therefore, a k value that is close to but smaller than 28 can compress over 190k prefixes into just 7k TCAM entries. We briefly explore other choices of k for IPv6 and examine potential latency-memory trade-offs in Appendix A.6.

MashUp's key parameter is its set of strides. Intuitively, we want to select strides that mirror the distribution spikes **(P1)** seen in Figure 8 because they will minimize prefix expansion. For IPv4, we choose 16-4-4-8 (spikes at 16, 20, 24). For IPv6, we choose 20-12-16-16 (spikes at 32, 48). We do not select 32 as the first stride because it is too wide—especially for the root node which may contain many entries. Therefore, we decompose 32 into separate strides of 20 and 12.

6.4 Comparisons *before* Implementation

Recall that CRAM metrics enable quick estimation of algorithm scalability *before* implementation.

We present IPv4 and IPv6 CRAM metrics in Table 4 and Table 5, derived from the optimization steps in §3.1, §4.1, and §5.1, for the IPv4 and IPv6 BGP tables, respectively.

For IPv4, RESAIL outperforms BSIC in all three CRAM metrics. Between RESAIL and MashUp, RESAIL wins in TCAM and steps but loses in SRAM. However, MashUp requires 100X more TCAM than RESAIL, whereas RESAIL requires only 1.4X more SRAM than MashUp. Therefore, we determine RESAIL to be the best CRAM IPv4 algorithm.

For IPv6, we choose between BSIC and MashUp. BSIC wins in TCAM but loses in SRAM and steps. MashUp requires 16X more TCAM than BSIC, while BSIC requires roughly 4X more SRAM and steps than MashUp. As before, we prioritize TCAM because it is more expensive and limited

Scheme	TCAM Blocks	SRAM Pages	Stages
MashUp (16-4-4-8)	235	216	10
BSIC (*k*=16)	74	558	16
RESAIL (*min_bmp*=13)	2	556	9

Table 6: Ideal RMT mapping for IPv4 prefixes in AS65000

Scheme	TCAM Blocks	SRAM Pages	Stages
MashUp (20-12-16-16)	178	47	8
BSIC (*k*=24)	15	211	14

Table 7: Ideal RMT mapping for IPv6 prefixes in AS131072

than SRAM—for example, Tofino-2 contains 19X more SRAM than TCAM. Although BSIC uses more steps than MashUp, this is due to BSIC's use of BSTs, which have a high initial step cost. Therefore, we determine BSIC to be the best CRAM IPv6 algorithm *for Tofino-2*. However, for more stage-constrained ASICs, MashUp is likely better.

To verify the validity of the CRAM metrics, we explicitly map each CRAM algorithm to an ideal RMT chip and present its resource utilization in Table 6 and Table 7. This mapping, which accounts for Tofino-2 TCAM block sizes, SRAM page sizes, and per-stage memory, is precisely the complicated process that the CRAM model seeks to relieve algorithm designers of. Comparing Table 4 with Table 6 and Table 5 with Table 7, observe that the CRAM metrics accurately predict a target algorithm's resource utilization and potential scalability.

6.5 Comparisons *after* Implementation

The previous subsection compared our three new algorithms *before* implementation using CRAM metrics. Here, we compare the best CRAM algorithms *after* implementation on Tofino-2 against the best pre-existing IPv4 and IPv6 schemes.

6.5.1 Baseline Selection

We select four single-resource baselines: SRAM-only for IPv4 and IPv6, and TCAM-only for IPv4 and IPv6.

SRAM-only for IPv4: We choose SAIL [83] as our SRAM-only IPv4 baseline due to its on-chip memory bound for short prefixes, which enables it to scale very well. Although IPv4 schemes like Poptrie [7] and DXR [89] use less memory, they require too many memory accesses and stages.

SRAM-only for IPv6: We choose Hi-BST [65] as our SRAM-only IPv6 baseline because it is the most memory-efficient IPv6 lookup algorithm to date [90]. It uses a treap data structure that maps each prefix to a unique node.

TCAM-only for IPv4 and IPv6: We choose a logical TCAM as our TCAM-only IPv4 and IPv6 baseline because, although TCAM-oriented schemes exist for reducing power consumption [48] or merging multiple FIBs [51], none focus on scaling IP lookup for a single database.

Scheme	TCAM Blocks	SRAM Pages	Stages	Target Chip
RESAIL (*min_bmp*=13)	17	750	16	Tofino-2
RESAIL (*min_bmp*=13)	2	556	9	Ideal RMT
SAIL	-	2313	33	Ideal RMT
Logical TCAM	1822	-	76	Ideal RMT
Tofino-2 Pipe Limit	480	1600	20	-

Table 8: Baseline comparison for IPv4 prefixes in AS65000

Scheme	TCAM Blocks	SRAM Pages	Stages	Target Chip
BSIC (*k*=24)	15	416	30	Tofino-2
BSIC (*k*=24)	15	211	14	Ideal RMT
Hi-BST	-	219	18	Ideal RMT
Logical TCAM	762	-	32	Ideal RMT
Tofino-2 Pipe Limit	480	1600	20	-

Table 9: Baseline comparison for IPv6 prefixes in AS131072

6.5.2 IPv4 Comparison

Table 8 compares the ideal RMT resource utilization of RESAIL with the IPv4 baselines. RESAIL requires 911X fewer TCAM blocks than the logical TCAM and approximately 4X fewer SRAM pages and stages than SAIL. Although SAIL's memory scales efficiently, its high upfront cost makes it impractical for RMT-like chips. RESAIL outperforms the logical TCAM, which only supports IPv4 databases of up to 245,760 entries—about 3.8X smaller than the current IPv4 BGP table.

Table 8 also compares RESAIL on an ideal RMT chip and on Tofino-2. While RESAIL fits on Tofino-2 for AS65000, it requires nearly 1.4X more SRAM pages and 2X more stages than on an ideal RMT chip. This is because Tofino-2 reserves bits in each SRAM word for identifying actions, limiting the maximum SRAM utilization to 50%. The increase in TCAM is due to extra ternary bitmask tables needed for extracting bits.

6.5.3 IPv6 Comparison

Table 9 compares the ideal RMT resource utilization of BSIC with the IPv6 baselines. BSIC uses less SRAM and fewer stages than Hi-BST, at the cost of 15 TCAM blocks. Both BSIC and Hi-BST support the current IPv6 BGP table, whereas the logical TCAM only supports up to 122,880 entries—about 1.6X smaller than the current IPv6 BGP table.

Table 9 also compares BSIC on an ideal RMT chip and on Tofino-2. Our ideal RMT chip assumes each stage can perform at least two dependent ALU operations. However, in practice, a Tofino-2 stage can execute only one level of ALU logic. Consequently, each BST level requires two stages: one for comparing the search key and another for performing the P4 action. This creates a repeating pattern where an SRAM-intensive stage is followed by a stage with minimal

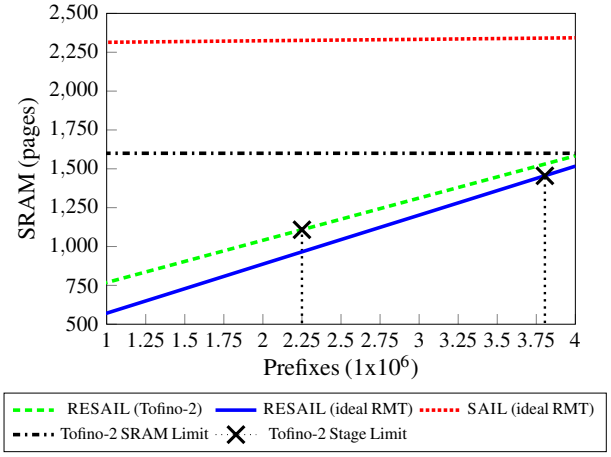


Figure 9: RESAIL vs SAIL scaling (IPv4)

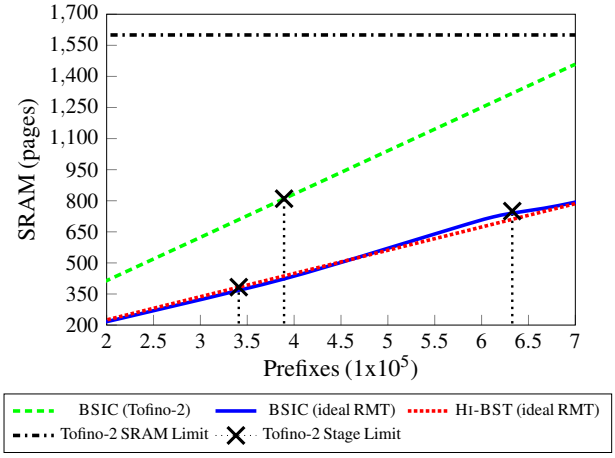


Figure 10: BSIC vs HI-BST scaling (IPv6)

SRAM usage. Although BSIC on Tofino-2 requires 30 stages (ten over the Tofino-2 pipe limit), we successfully fit BSIC for AS131072 by recirculating each packet. However, this effectively halves the number of available switch ports.

7 Scalability

While §6 presents results for *current* BGP tables, this section presents scalability analysis for RESAIL and BSIC on larger *synthetic* routing databases. We omit scalability analysis for MASHUP because it requires too much TCAM (for Tofino-2).

7.1 IPv4 Scaling

Figure 9 shows IPv4 scalability results for RESAIL and our SRAM-only IPv4 baseline, SAIL. We did not generate synthetic prefixes because the resource utilization of RESAIL and SAIL depends on the distribution of *prefix lengths* rather than the distribution of the prefixes themselves. From the perspective of memory usage, RESAIL and SAIL do not distinguish between prefixes of identical length. Therefore, we use a simple scaling model that applies a constant scaling factor to all prefix lengths. For the ideal RMT results of RESAIL and SAIL, we use the steps described in §6.2 to calculate their new utilization. For the Tofino-2 results of RESAIL, we update the corresponding P4 table sizes to reflect the larger databases.

For ideal RMT, SAIL is infeasible because its SRAM cost far exceeds the Tofino-2 SRAM limit. At any given database size, RESAIL for Tofino-2 uses more SRAM than RESAIL for ideal RMT. This is expected since Tofino-2 does not allow 100% SRAM utilization. Notably, RESAIL on an ideal RMT chip scales to around 3.8 million prefixes, 4X larger than the current IPv4 BGP table. RESAIL on Tofino-2 scales to around 2.25 million prefixes, 2.3X larger than the current routing database and significantly beyond SAIL's capacity.

7.2 IPv6 Scaling

Figure 10 shows IPv6 scalability results for BSIC and our SRAM-only IPv6 baseline, HI-BST. For BSIC, we generated synthetic prefixes because its resource utilization depends on the distribution of prefixes and sub-prefixes. To obtain worst-case scalability results, observe that the first three bits of IPv6 prefixes in AS131072 are 000—forming an *IPv6 universe*. We use different combinations of these bits to generate significantly larger synthetic databases from AS131072, an approach we call *multiverse scaling*. Multiverse scaling assumes that the distribution of all prefix lengths scales uniformly. In practice, customer scaling causes some prefixes (e.g., /48s) to scale more rapidly than others (e.g., /24s). However, this stresses only the BSTs and not the initial TCAM, unlike multiverse scaling which models worst-case results for TCAM, SRAM, and stages. For HI-BST, we use the memory calculation provided in [65].

As seen in Figure 10, both instances of BSIC are able to out-scale HI-BST. For an ideal RMT chip, HI-BST only scales to around 340k prefixes, 1.8X larger than the current IPv6 BGP Table. Even though HI-BST is the most memory efficient IPv6 lookup scheme, it requires too many stages. Comparing the two instances of BSIC, we see that BSIC for ideal RMT scales to around 630k prefixes, 3.3X larger than the current routing database. Since BSIC for Tofino-2 uses over 2X more stages, it scales to around 390k prefixes—2X the size of the current IPv6 BGP table.

8 CRAM Model Evaluation

How predictive was the CRAM model? §6.4 showed that the CRAM model accurately predicted RESAIL and BSIC as the best algorithms for IPv4 and IPv6, respectively. We now examine the CRAM metrics in more detail.

Table 10 and Table 11 show results for RESAIL and BSIC on three models: the CRAM model, an ideal RMT model, and a Tofino-2 implementation. The three models form a hierarchy

Scheme	TCAM Blocks	SRAM Pages	Steps (Stages)	Model
RESAIL (*min_bmp*=13)	1.14	549.12	2	CRAM
RESAIL (*min_bmp*=13)	2	556	9	Ideal RMT
RESAIL (*min_bmp*=13)	17	750	16	Tofino-2

Table 10: Predictive accuracy of CRAM for RESAIL (IPv4)

Scheme	TCAM Blocks	SRAM Pages	Steps (Stages)	Model
BSIC (*k*=24)	7.45	203.52	14	CRAM
BSIC (*k*=24)	15	211	14	Ideal RMT
BSIC (*k*=24)	15	416	30	Tofino-2

Table 11: Predictive accuracy of CRAM for BSIC (IPv6)

of abstractions with increasing detail. We scale the CRAM metrics found in Table 4 and Table 5 from raw bits to TCAM blocks and SRAM pages to allow for uniform comparisons. The three models can be understood as follows:

CRAM model: Using the CRAM metrics (raw bits and dependent steps), an algorithm designer can quickly predict scalability *without* seeing the product data sheet.

Ideal RMT model: This model allows for more precise predictions but requires a *basic* understanding of the data sheet, specifically the general organization of memory and stages.

Tofino-2 implementation: This is the most accurate model but also the most complex to develop. It accounts for low-level details that are hard to glean from data sheets, such as action bits and ALU operations per stage. This often requires an *expert* with intimate knowledge of the product.

Consider the predictive accuracy for RESAIL in Table 10 as we move from CRAM to ideal RMT. The TCAM and SRAM measures reflect small rounding errors due to unit conversion. However, the latency increases significantly from 2 steps to 9 stages because, unlike dRMT, RMT stages provide *both* memory and processing—to support 556 RAM pages, more stages are required even when no additional processing is needed.

Next, consider the predictive accuracy for RESAIL in Table 10 as we move from ideal RMT to Tofino-2. There is a small additive increase in TCAM blocks due to extra ternary tables required for implementing RESAIL in P4. Additionally, SRAM pages increase by a factor of 1.35, and stages increase by a factor of 1.78. As discussed earlier in § 6.5.2, this is because the maximum achievable SRAM utilization on Tofino-2 is 50%, necessitating more SRAM pages and stages.

The predictive accuracy for BSIC in Table 11 can be interpreted similarly. The key difference is that the ∼2X increase in SRAM pages and stages from ideal RMT to Tofino-2 is due to the fact that implementing 3-way branching on Tofino-2 requires two stages for each BST level.

Based on our limited experience, the CRAM model provides a useful, easily computed initial model for estimating algorithm scalability. Although its measures of space and time are off by small constant factors, this is no different from Big O notation.

9 Related Work

Models of Computation: Abstract models like RAM [23], PRAM [41], and Turing machines [62] are widely used to analyze algorithms. The RAM model abstracts sequential computers, while the PRAM model abstracts shared memory multiprocessors. Our CRAM model abstracts network processors with two types of memory, parallelism, and programmability.

Combinations of CAM and RAM: Earlier CAM and RAM combinations optimize different metrics. CoolCAM [88], cooler TCAM [49], and EaseCAM [60] all reduce power consumption. Liu [48] compacts routing tables to reduce cost, power consumption, and thermal dissipation. Luo [51] merges FIBs in virtual routers to reduce TCAM usage. Compaction and merging are orthogonal to our ideas. Other hybrid approaches [38, 69, 75, 84] target tasks such as packet classification. In summary, no existing solution optimizes scalability for IP lookup by leveraging *both* TCAM and SRAM.

10 Conclusion

Our paper introduces new algorithms for IP lookup made possible by strategically leveraging *both* CAM and RAM. These algorithms address the scaling challenges of global routing tables and are well-suited to the architectures of modern network processors. For these processors, the CRAM lens provided a fresh perspective on algorithm design. Much like the RAM [23] and PRAM [41] models, CRAM offers metrics for quickly evaluating algorithm feasibility *before* implementation. Using the CRAM model, we developed three new IP lookup algorithms: RESAIL, BSIC, and MASHUP. RESAIL and BSIC scale to much larger databases than the best existing IPv4 and IPv6 lookup schemes, respectively, while MASHUP excels in stage-constrained hardware environments.

We aim to establish CRAM's generality by applying it to other hardware architectures (§ 2.4) and network applications (§ 2.5)—helping to "cram" more packet processing power into each unit of chip area. Ultimately, our findings underscore a simple insight for networking chip vendors: *a little TCAM goes a long way. Adding small amounts of TCAM to supplement SRAM can significantly improve scalability.*

Acknowledgments

We thank our shepherd, Xiaoqi Chen, and the anonymous reviewers for their valuable feedback. We are also grateful to David Maltz and Vladimir Gurevich for their guidance. This research was supported in part by NSF grant CNS-2333587.

References

[1] The Fungible DPU™: A New Category of Microprocessor for the Data-Centric Era : Hot Chips 2020. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–25, 2020.

[2] Advanced Micro Devices, Inc. AMD Pensando Networking. https://www.amd.com/en/products/accelerators/pensando.html.

[3] Advanced Micro Devices, Inc. Vitis Networking P4. https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html.

[4] Anurag Agrawal and Changhoon Kim. Intel Tofino2 – A 12.9Tbps P4-Programmable Ethernet Switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–32, 2020.

[5] B. Agrawal and T. Sherwood. Modeling TCAM power for next generation network devices. In *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 120–129, 2006.

[6] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 29–43, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Hirochika Asai and Yasuhiro Ohara. Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup. *SIGCOMM Comput. Commun. Rev.*, 45(4):57–70, August 2015.

[8] Masanori Bando, Yi-Li Lin, and H. Jonathan Chao. FlashTrie: beyond 100-Gb/s IP route lookup using hash-based prefix-compressed trie. *IEEE/ACM Trans. Netw.*, 20(4):1262–1275, August 2012.

[9] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. *SIGCOMM Comput. Commun. Rev.*, 43(4):99–110, August 2013.

[10] A. Broder and M. Mitzenmacher. Using multiple hash functions to improve ip lookups. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1454–1463 vol.3, 2001.

[11] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485 – 509, 2003.

[12] Xiaoqi Chen. Implementing AES Encryption on Programmable Switches via Scrambled Lookup Tables. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, SPIN '20, page 8–14, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 15–29, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Xiaoqi Chen, Andrew Johnson, Mengying Pan, and David Walker. Synthesizing state machines for data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '22, page 81–88, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, Ariel Orda, and Tom Edsall. dRMT: Disaggregated Programmable Switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 1–14, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Cisco Systems, Inc. Cisco Silicon One - Network Processors for Unified Architecture - Cisco. https://www.cisco.com/site/us/en/products/networking/silicon-one/index.html.

[17] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[18] Rajdeep Das and Alex C Snoeren. Memory Management in ActiveRMT: Towards Runtime-programmable Switches. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1043–1059, New York, NY, USA, 2023. Association for Computing Machinery.

[19] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. *SIGCOMM Comput. Commun. Rev.*, 27(4):3–14, October 1997.

[20] Kevin Deierling. NVIDIA's Resource Transmutable Network Processing ASIC. In *2023 IEEE Hot Chips 35 Symposium (HCS)*, pages 1–14, 2023.

[21] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. In *Proceedings of the 2003 Conference on*

Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '03, page 201–212, New York, NY, USA, 2003. Association for Computing Machinery.

[22] Will Eatherton, George Varghese, and Zubin Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, April 2004.

[23] Matteo Frigo and Victor Luchangco. Computation-centric memory models. In *Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '98, page 240–249, New York, NY, USA, 1998. Association for Computing Machinery.

[24] Michael Galles and Francis Matus. Pensando Distributed Services Architecture. *IEEE Micro*, 41(2):43–49, 2021.

[25] Geoff Huston. AS131072 - IPv6 BGP Table Statistics. https://bgp.potaroo.net/v6/as2.0/index.html.

[26] Geoff Huston. AS65000 - BGP Table Statistics. https://bgp.potaroo.net/as2.0/.

[27] Geoff Huston. CIDR Report. https://www.cidr-report.org/as2.0/.

[28] Geoff Huston. IPv6 CIDR Report. https://www.cidr-report.org/v6/as2.0/.

[29] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98*, volume 3, pages 1240–1247 vol.3, 1998.

[30] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[31] B. Indira, K. Valarmathi, and D. Devaraj. A Trie based IP Lookup Approach for High Performance Router/Switch. In *2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, pages 1–6, 2019.

[32] Intel Corporation. Intel® Infrastructure Processing Unit (Intel® IPU). https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.

[33] Intel Corporation. Intel® P4 Insight Tool. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/p4-insight.html.

[34] Intel Corporation. Intel® P4 Suite for FPGAs. https://www.intel.com/content/www/us/en/software/programmable/p4-suite-fpga/overview.html.

[35] Intel Corporation. Intel® Tofino™ 2. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2.html.

[36] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 121–136, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Lavanya Jose, Lisa Yan, George Varghese, and Nick McKeown. Compiling Packet Programs to Reconfigurable Switches. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 103–115, Oakland, CA, May 2015. USENIX Association.

[38] Changhun Jung, Sian Kim, Rhongho Jang, David Mohaisen, and DaeHun Nyang. A Scalable and Dynamic ACL System for In-Network Defense. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1679–1693, New York, NY, USA, 2022. Association for Computing Machinery.

[39] Radhakrishna Kamath and Krishna M Sivalingam. Machine Learning based Flow Classification in DCNs using P4 Switches. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–10, 2021.

[40] Kun Suk Kim and S. Sahni. IP lookup by binary search on prefix length. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, pages 77–82 vol.1, 2003.

[41] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71(1):95–132, 1990.

[42] A. Kumar and J. Xu. Sketch Guided Sampling - Using On-Line Estimates of Flow Size for Adaptive Data Collection. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–11, 2006.

[43] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan Turner, and George Varghese. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In *Proceedings of the 3rd ACM/IEEE Symposium*

on Architecture for Networking and Communications Systems, ANCS '07, page 155–164, New York, NY, USA, 2007. Association for Computing Machinery.

[44] Karthik Lakshminarayanan, Anand Rangarajan, and Srinivasan Venkatachary. Algorithms for advanced packet classification with ternary CAMs. *SIGCOMM Comput. Commun. Rev.*, 35(4):193–204, August 2005.

[45] B. Lampson, V. Srinivasan, and G. Varghese. IP lookups using multiway and multicolumn search. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98*, volume 3, pages 1248–1256 vol.3, 1998.

[46] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: a better NetFlow for data centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, page 311–324, USA, 2016. USENIX Association.

[47] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 256–269, New York, NY, USA, 2019. Association for Computing Machinery.

[48] Huan Liu. Routing table compaction in ternary CAM. *IEEE Micro*, 22(1):58–64, 2002.

[49] Haibin Lu. Improved trie partitioning for cooler TCAMs. In *IASTED International Conference on Advances in Computer Science and Technology*. Citeseer, 2004.

[50] Daniel Luchaup, Randy Smith, Cristian Estan, and Somesh Jha. Multi-byte Regular Expression Matching with Speculation. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, page 284–303, Berlin, Heidelberg, 2009. Springer-Verlag.

[51] Layong Luo, Gaogang Xie, Steve Uhlig, Laurent Mathy, Kavé Salamatian, and Yingke Xie. Towards TCAM-based scalable virtual routers. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, page 73–84, New York, NY, USA, 2012. Association for Computing Machinery.

[52] Marvell Technology, Inc. Data Center Switches | Enabling the AI cloud with High-bandwidth, low-latency, multi-terabit network switches. https://www.marvell.com/products/data-center-switches.html.

[53] Nick McKeown. Intel's commitment to P4. https://groups.google.com/a/lists.p4.org/g/p4-announce/c/frXi_jjmawE.

[54] S. Nilsson and G. Karlsson. Fast address look-up for internet routers. In *Proceedings of the IFIP TC6/WG6.2 Fourth International Conference on Broadband Communications: The Future of Telecommunications*, BC '98, page 11–22, GBR, 1998. Chapman & Hall, Ltd.

[55] Nvidia Corporation. NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[56] Nvidia Corporation. The Spectrum Ethernet Platform. https://www.nvidia.com/en-us/networking/products/ethernet/.

[57] P4 Compiler Project. p4c/backends/tofino/bf-p4c at main · p4lang/p4c · GitHub. https://github.com/p4lang/p4c/tree/main/backends/tofino/bf-p4c, 2025.

[58] P4 Project. P4 Ecosystem – P4 – Language Consortium. https://p4.org/ecosystem/?_product_category=hardware.

[59] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Siracusano. FlowBlaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 531–548, Boston, MA, February 2019. USENIX Association.

[60] V.C. Ravikumar, R.N. Mahapatra, and Laxmi Narayan Bhuyan. EaseCAM: an energy and storage efficient TCAM-based router architecture for IP lookup. *IEEE Transactions on Computers*, 54(5):521–533, 2005.

[61] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 323–334, New York, NY, USA, 2012. Association for Computing Machinery.

[62] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1997.

[63] Alexandru Seibulescu and Mario Baldi. Leveraging P4 Flexibility to Expose Target-specific Features. In *Proceedings of the 3rd P4 Workshop in Europe*, EuroP4'20, page 36–42, New York, NY, USA, 2020. Association for Computing Machinery.

[64] D. Shah and P. Gupta. Fast updating algorithms for TCAM. *IEEE Micro*, 21(1):36–47, 2001.

[65] Tong Shen, Xian Yu, Gaogang Xie, and Dafang Zhang. High-Performance IPv6 Lookup with Real-Time Updates Using Hierarchical-Balanced Search Tree. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2018.

[66] Sandeep Sikka and George Varghese. Memory-efficient state lookups with fast updates. *SIGCOMM Comput. Commun. Rev.*, 30(4):335–347, August 2000.

[67] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 213–224, New York, NY, USA, 2003. Association for Computing Machinery.

[68] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research*, SOSR '17, page 164–176, New York, NY, USA, 2017. Association for Computing Machinery.

[69] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended TCAMs. In *11th IEEE International Conference on Network Protocols, 2003. Proceedings.*, pages 120–131, 2003.

[70] V. Srinivasan and George Varghese. Faster IP lookups using controlled prefix expansion. *SIGMETRICS Perform. Eval. Rev.*, 26(1):1–10, June 1998.

[71] Naru Sundar, Brad Burres, Yadong Li, Dave Minturn, Brian Johnson, and Nupur Jain. 9.4 An In-depth Look at the Intel IPU E2000. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 162–164, 2023.

[72] Subhash Suri, G. Varghese, and P.R. Warkhede. Multiway range trees: scalable IP lookup with fast updates. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, volume 3, pages 1610–1614 vol.3, 2001.

[73] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 329–342, New York, NY, USA, 2023. Association for Computing Machinery.

[74] David E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, September 2005.

[75] Balajee Vamanan and T. N. Vijaykumar. TreeCAM: decoupling updates and lookups in packet classification. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[76] George Varghese. *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[77] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing lookups. *SIGCOMM Comput. Commun. Rev.*, 27(4):25–36, October 1997.

[78] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. Hyperscan: a fast multi-pattern regex matcher for modern CPUs. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 631–648, USA, 2019. USENIX Association.

[79] Guorui Xie, Qing Li, Yutao Dong, Guanglin Duan, Yong Jiang, and Jingpu Duan. Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 1938–1947, 2022.

[80] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. Runtime Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 651–665, Renton, WA, April 2022. USENIX Association.

[81] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. Unleashing SmartNIC Packet Processing Performance in P4. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1028–1042, New York, NY, USA, 2023. Association for Computing Machinery.

[82] Rui Yang, Haitong Li, Kirby K. H. Smithe, Taeho R. Kim, Kye Okabe, Eric Pop, Jonathan A. Fan, and H.-S. Philip Wong. Ternary content-addressable memory with MoS2 transistors for massively parallel data search. *Nature Electronics*, 2(3):108–114, Mar 2019.

[83] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X. Liu, Qi Li, and Laurent Mathy. Guarantee IP lookup performance with FIB explosion. *SIGCOMM Comput. Commun. Rev.*, 44(4):39–50, August 2014.

[84] Ruyi Yao, Cong Luo, Xuandong Liu, Ying Wan, Bin Liu, Wenjun Li, and Yang Xu. MagicTCAM: A Multiple-TCAM Scheme for Fast TCAM Update. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11, 2021.

[85] Changhoon Yim, Bomi Lee, and Hyesook Lim. Efficient binary search for IP address lookup. *IEEE Communications Letters*, 9(7):652–654, 2005.

[86] Sophia Yoo and Xiaoqi Chen. Secure Keyed Hashing on Programmable Switches. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable Network INfrastructure*, SPIN '21, page 16–22, New York, NY, USA, 2021. Association for Computing Machinery.

[87] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with OpenSketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, page 29–42, USA, 2013. USENIX Association.

[88] F. Zane, Girija Narlikar, and A. Basu. Coolcams: power-efficient TCAMs for forwarding engines. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 1, pages 42–52 vol.1, 2003.

[89] Marko Zec, Luigi Rizzo, and Miljenko Mikuc. DXR: towards a billion routing lookups per second in software. *SIGCOMM Comput. Commun. Rev.*, 42(5):29–36, September 2012.

[90] Chunyang Zhang and Gaogang Xie. Using XorOffsetTrie for high-performance IPv6 lookup in the backbone network. *Computer Communications*, 181:438–445, 2022.

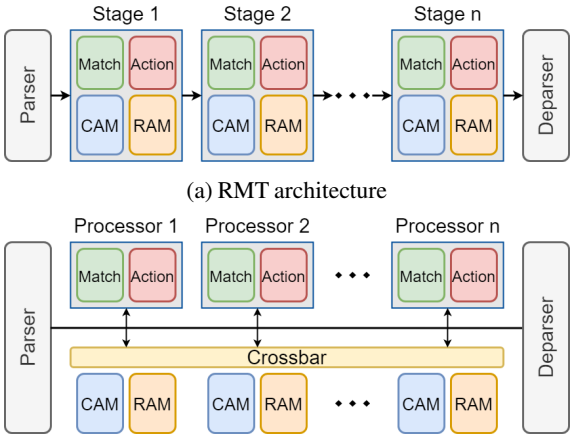
[91] Changgang Zheng, Xinpeng Hong, Damu Ding, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. In-Network Machine Learning Using Programmable Network Devices: A Survey. *IEEE Communications Surveys & Tutorials*, 26(2):1171–1200, 2024.

[92] Changgang Zheng, Zhaoqi Xiong, Thanh T. Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. IIsy: Hybrid In-Network Classification Using Programmable Switches. *IEEE/ACM Trans. Netw.*, 32(3):2555–2570, February 2024.

[93] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Liam Perreault, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Planter: Rapid Prototyping of In-Network Machine Learning Inference. *SIGCOMM Comput. Commun. Rev.*, 54(1):2–21, August 2024.

A Appendix

A.1 Packet Processing Architectures



(a) RMT architecture

(b) dRMT architecture

Figure 11: RMT vs dRMT

A.2 RMT and dRMT Implementations

Product	Architecture	Type	Sources
Intel Tofino-2	RMT	switch ASIC	[4, 18, 35]
Intel Mount Evans (E2000)	RMT	SmartNIC	[32, 71]
AMD Pensando DSC-100	RMT	SmartNIC	[2, 24, 63]
Fungible F1/S1	RMT	SmartNIC	[1]
FlowBlaze	RMT	FPGA	[59]
FlexCore	dRMT	switch ASIC	[80]
Nvidia BlueField-3	dRMT	SmartNIC	[20, 55, 81]

Table 12: Summary of known implementations

While we cannot verify all details in some cases, it is evident that TCAM, SRAM, parallelism, and programmability are present in the sources for the products listed in Table 12.

A.3 Updates, Deletions, and Insertions

A.3.1 RESAIL

Incremental updates, deletions, and insertions for RESAIL are efficient, following the same process as lookups but with modifications to the target entry. For prefixes of length *min_bmp* or greater, only two memory accesses are required (bitmap and hash table). For prefixes shorter than *min_bmp*, the operations are more costly because of prefix expansion. Update operations for prefixes longer than 24 bits are much simpler in RESAIL than in SAIL because we have eliminated pivot pushing.

A.3.2 BSIC

For BSIC, incremental updates, deletions, and insertions are costly and complex due to inherent dependencies between binary search tree levels. A separate database with additional prefix information is needed for rebuilding data structures [89]. If fast update operations are important, RESAIL and MASHUP are better choices.

A.3.3 MASHUP

Incremental updates, deletions, and insertions for MASHUP are nearly identical to lookups, except they modify the target entry. These are standard algorithms [76] for multibit tries that have been well studied. Maintaining a sorted TCAM table under these changes is non-trivial, but effective algorithms exist [64].

A.4 Range Expansion and Optimizations

Range	Next Hop
0000 - 0011	C
0100 - 0100	A
0101 - 0111	D
1000 - 1001	-
1010 - 1010	B
1011 - 1011	C
1100 - 1111	-

Table 13: Range expansion for slice 1001 (Table 3)

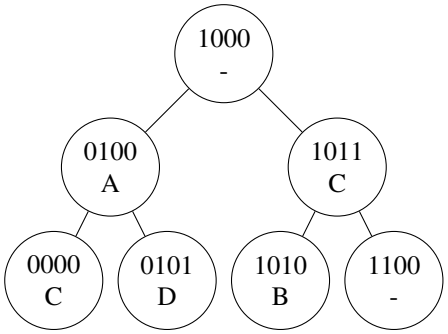


Figure 12: BST for slice 1001 (Table 3)

As in DXR [89], convert all the prefix substrings into ranges by generating their endpoint pairs. Use the endpoint pairs to create sorted, contiguous, and non-overlapping intervals that cover all possible bitstrings of the maximum length. Intervals that are added to complete the full range will "inherit" the next hop of the current lookup table entry's longest prefix match. This is necessary because it is possible for a destination address to be incorrectly directed by the initial lookup table to a BST that does not contain a legitimate match. Therefore, in the case of such a mistake, the search key will land in an interval containing the correct next hop. A simple example of range expansion for slice 1001 from Table 3 is shown in Table 13. Note that the intervals 1000-1001 and 1100-1111 were added to complete the full range. Since there are no valid longest prefix matches for slice 1001, its intervals are assigned a default value of -.

After the full range is created, merge neighboring intervals with the same next hop to reduce the number of nodes required. Discard the right endpoints as they can be inferred from the left ones. Use the remaining left endpoints to create the BST. Figure 12 shows a sample BST for slice 1001.

A.5 Pseudocode

Algorithm 1: RESAIL Lookup ($addr$, min_bmp)

```
Input  : addr, IPv4 address
Input  : min_bmp, smallest bitmap available
Output : hop, next hop
hop ← lookup_table.match(addr)
if hop ≠ None then
    return hop
for i ← 24 ; i ≥ min_bmp ; i−− do
    if B_i.match(addr ≫ (32-i)) == 1 then
        key ← (addr ≫ (32-i)) ≪ (25-i)
        key ← key + (1 ≪ (24-i))
        hop ← hash_table.match(key)
        break
return hop
```

Algorithm 2: BSIC Lookup ($addr$, k)

```
Input  : addr, IPv4 or IPv6 address
Input  : k, initial slice size
Output : hop_best, next hop
level ← 0
len ← 32 if IPv4, 64 if IPv6
hop_best, index
  ← lookup_table.match(addr ≫ (len−k))
if hop_best ≠ None then
    return hop_best
while index ≠ None
    hop, left, right, prefix ← bst_level.match(index)
    if prefix == ((addr ≪ k) ≫ k) then
        return hop
    else if prefix < ((addr ≪ k) ≫ k) then
        index ← right
        hop_best ← hop
    else
        index ← left
    level ← level + 1
return hop_best
```

A.6 Latency-memory trade-offs

A natural question is: *are there latency-memory trade-offs for CRAM algorithms that we can exploit to free up pipeline stages for other processing tasks?* Examining the basic CRAM model, there appears to be a clear trade-off: reducing dependency steps at the expense of increased memory or computation can lead to lower latency.

Algorithm 3: MASHUP Lookup ($addr$, $strides$)

```
Input  : addr, IPv4 or IPv6 address
Input  : strides, set of strides
Output : hop_best, next hop
hop_best ← default
table ← root
tag ← None
index ← 0
level ← 0
while table ≠ None
    key ← addr[index:index+strides[level]]
    hop, next_table, tag ← table.match(tag, key)
    if table.hit() then
        if hop ≠ None then
            hop_best ← hop
        index ← index + strides[level]
        level ← level + 1
        table ← next_table
    else
        break
return hop_best
```

Unfortunately, on real platforms like Tofino-2, *steps* cannot be conflated with *stages* because, in RMT, stages provide a fixed amount of both memory and computation. Therefore, large lookup tables require multiple stages. While a latency-memory trade-off exists for *steps* versus memory in our three CRAM algorithms, no corresponding trade-off exists for *stages* versus memory. Instead, there is an optimal number of stages, beyond which *both* memory and latency increase.

To see this, consider Figure 13 for BSIC (IPv6), in which the only tuning parameter is k—the width of the initial TCAM table. While a latency-memory curve exists, decreasing latency by increasing k actually *increases* the number of stages required. As k grows, the number of stages needed for the initial TCAM table rises significantly, outweighing the reduction in stages gained from decreasing BST depth. In contrast, the basic CRAM model predicts reduced latency as k increases. Unfortunately, this larger TCAM table requires more *stages* but not *steps*.

As shown in Figure 13, the optimal value of k is 24, with both smaller and larger values yielding worse results. Thus, no trade-off exists between stages and memory. This is also the reason we use $k=24$ in our experiments for IPv6.

Similarly, with MASHUP, the strides are the primary tuning parameter. Again, we did not find a useful memory-latency trade-off when considering stages. Finally, RESAIL consistently requires only two steps, with no latency-memory trade-off whatsoever in either the CRAM or ideal RMT models.

While there is no useful latency-memory trade-off for a *fixed* database size in our three algorithms, an obvious trade-off emerges as the database size *increases*: the number of stages (latency) must increase, at least to provide more memory. This is implicit in the linear trade-off curves shown in Figure 9 and Figure 10. Notably, the y-axes in these figures represent SRAM pages. However, since more SRAM requires proportionally more stages, Figure 9 and Figure 10 can be rescaled to (instead) depict stages versus database size, maintaining exactly the same curve shapes.

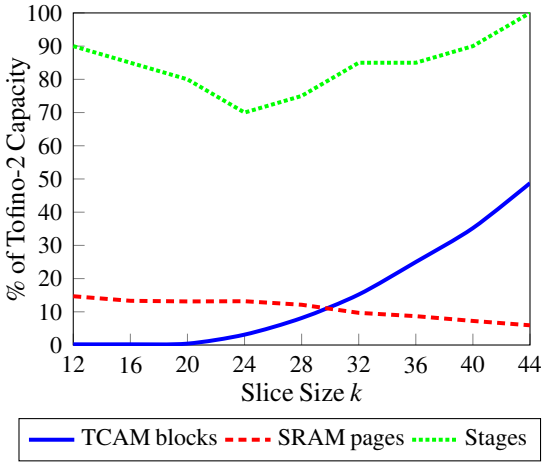


Figure 13: BSIC IPv6 latency-memory trade-off on an ideal RMT chip for AS131072 (Sep 2023)