IP Address Lookup

Masoud Sabaei

Associate professor

Department of Computer Engineering,
Amirkabir University of Technology

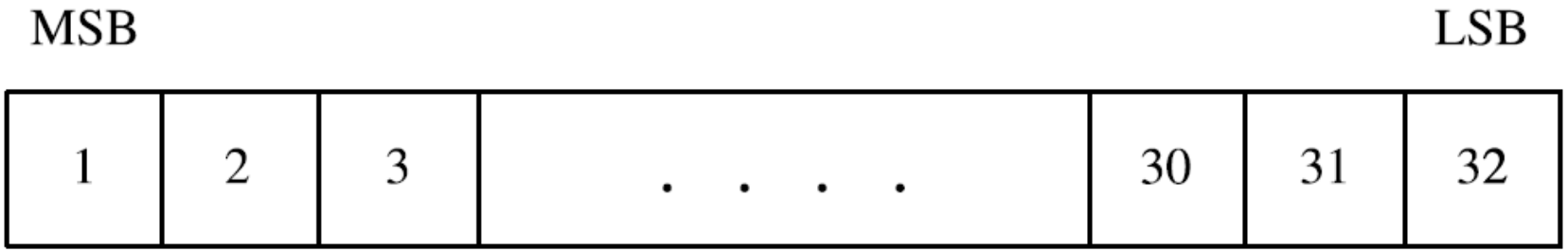
Outlines

- **Overview,**
- **Trie-based Algorithms,**
- **Hardware-based Schemes,**
- **IPV6 Lookup**

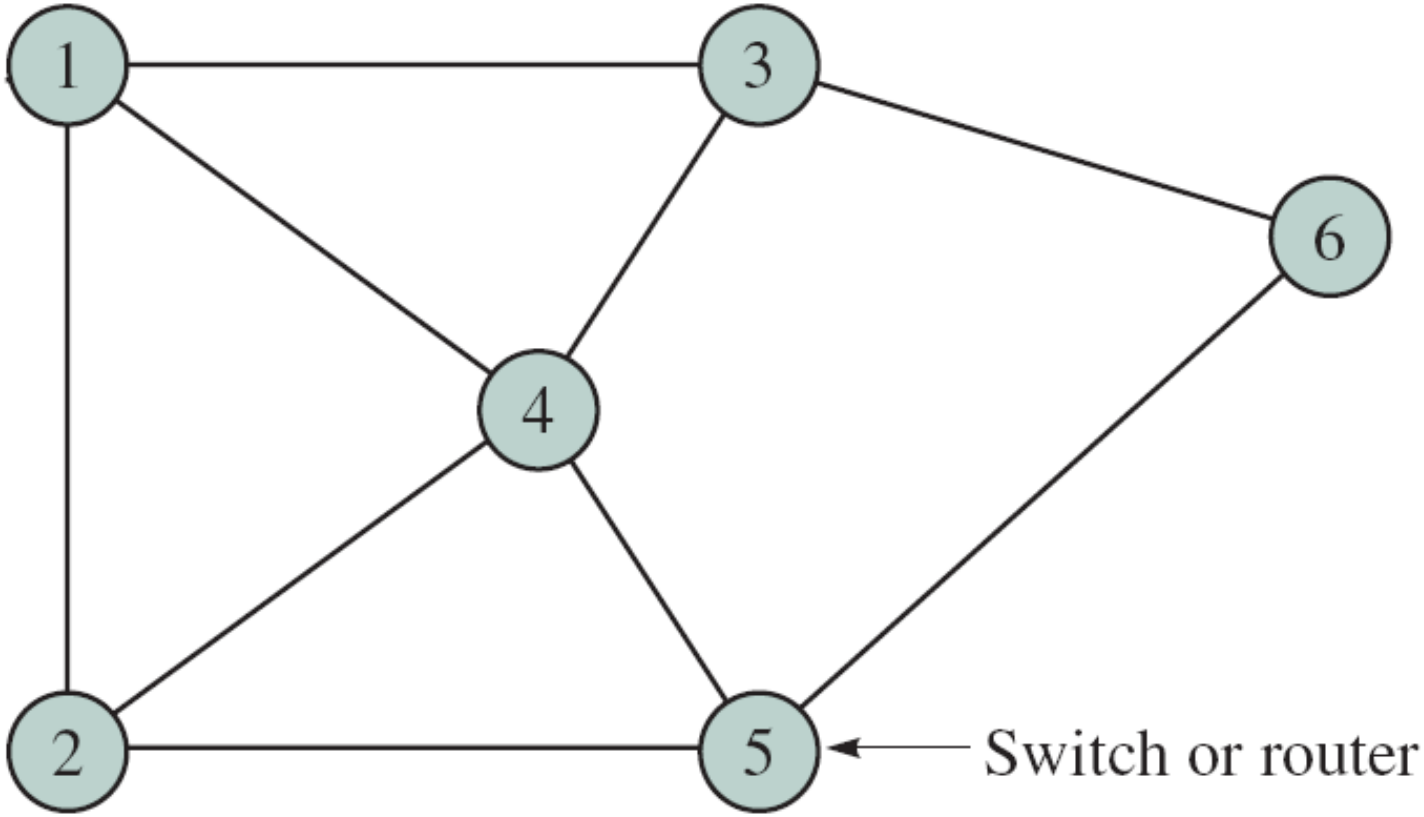
IP Routing

- Packet forwarding tasks
  - Packet header encapsulation and decapsulation
  - Updating TTL field
  - Checking for errors
  - …
  - IP route lookup -> Dominates the processing time

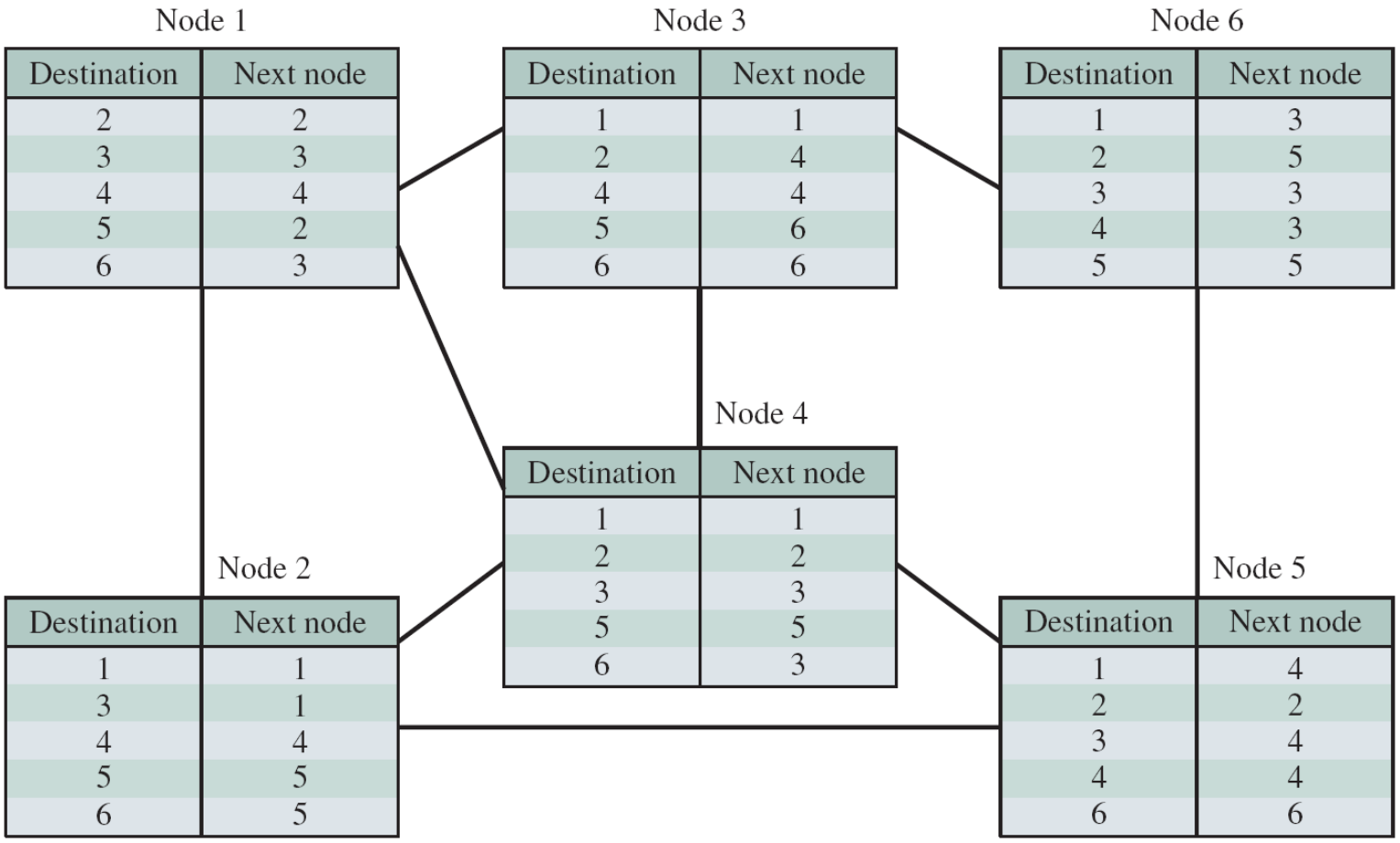
IP Address Bit Position



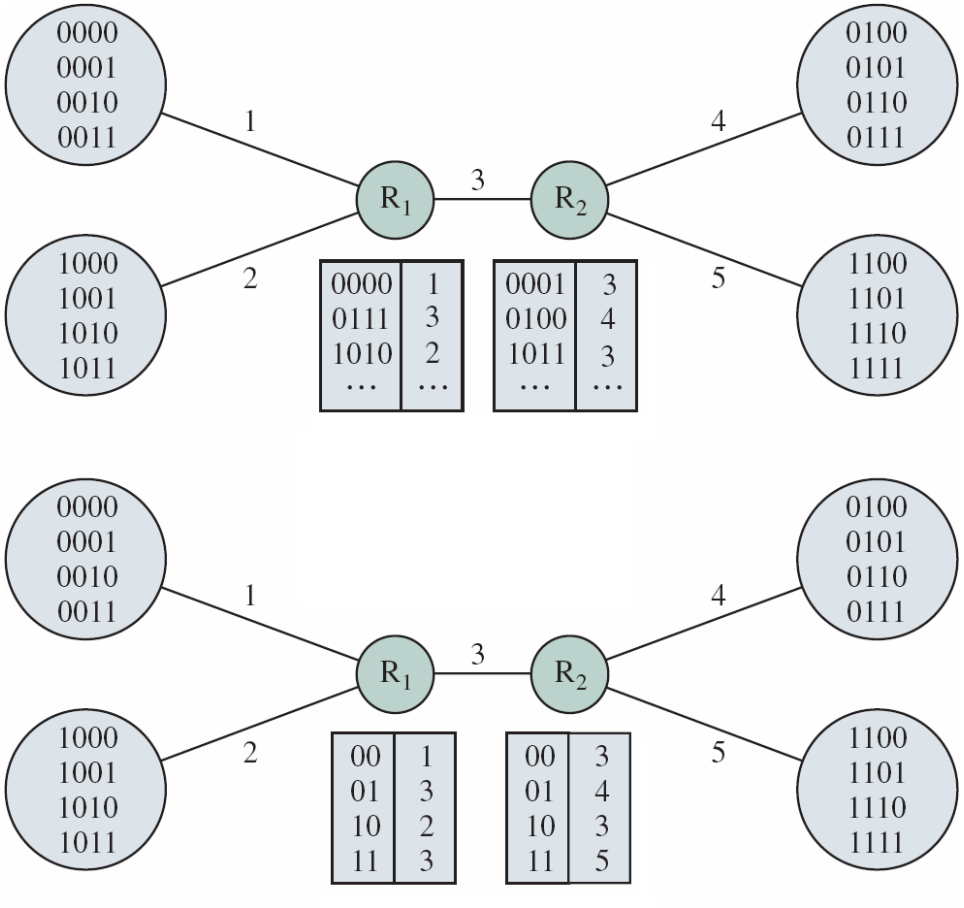
IP Address Lookup



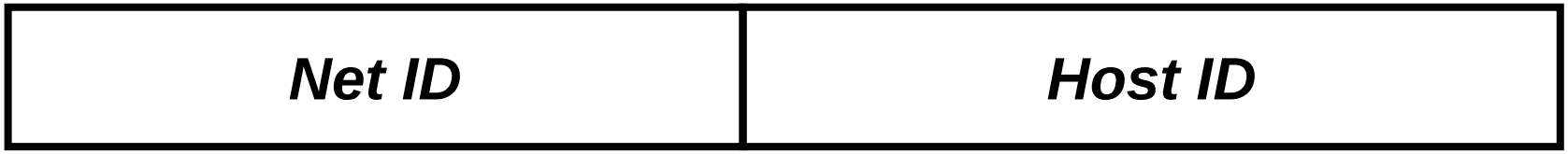
Routing Table at Each Node



Hierarchical Routing



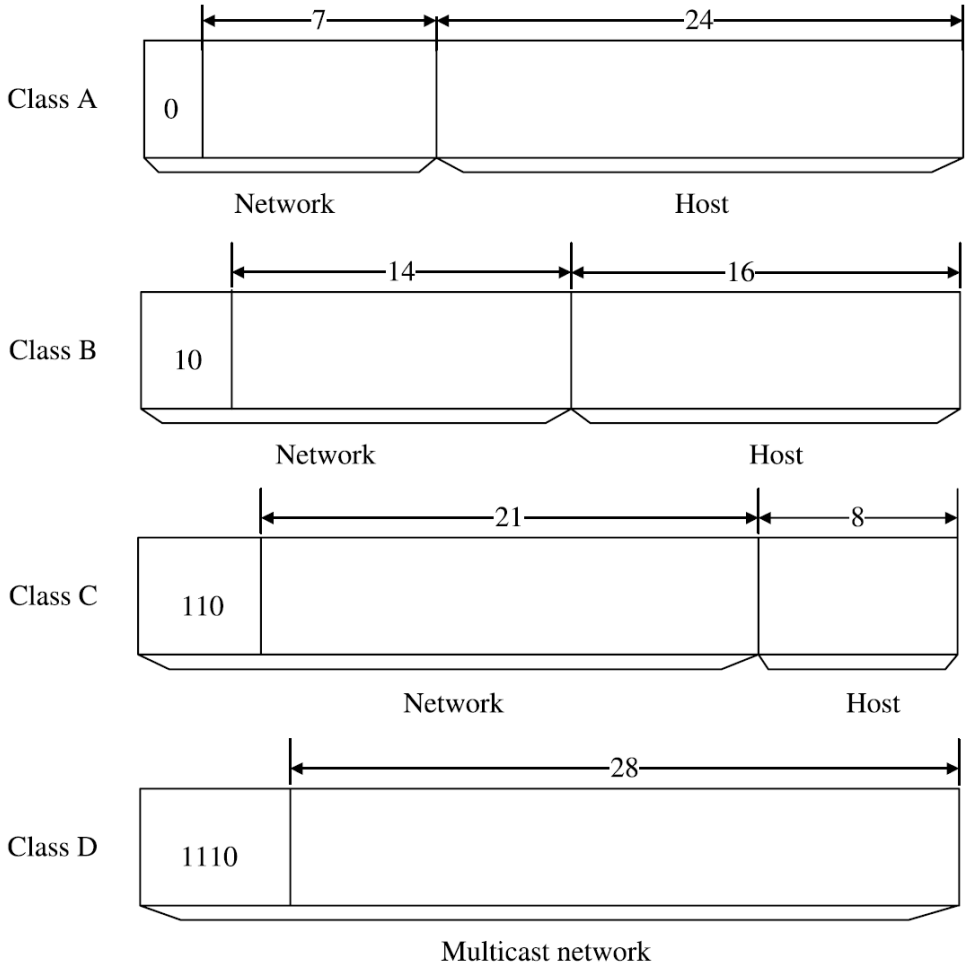
Hierarchical Routing



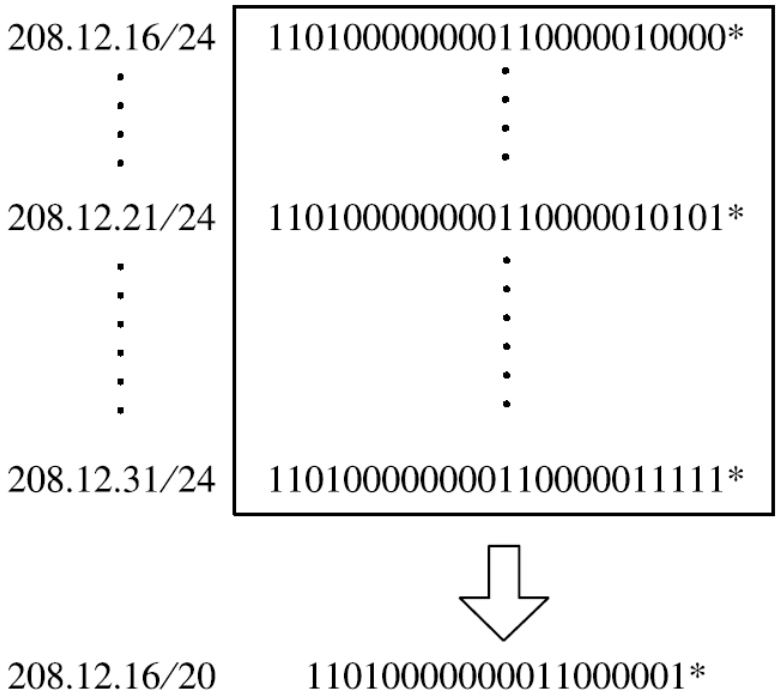
Router's Forwarding Table Structure

Destination Address Prefix	Next Hop IP Address	Output Interface
24.40.32/20	192.41.177.148	2
130.86/16	192.41.177.181	6
208.12.16/20	192.41.177.241	4
208.12.21/24	192.41.177.196	1
167.24.103/24	192.41.177.3	4

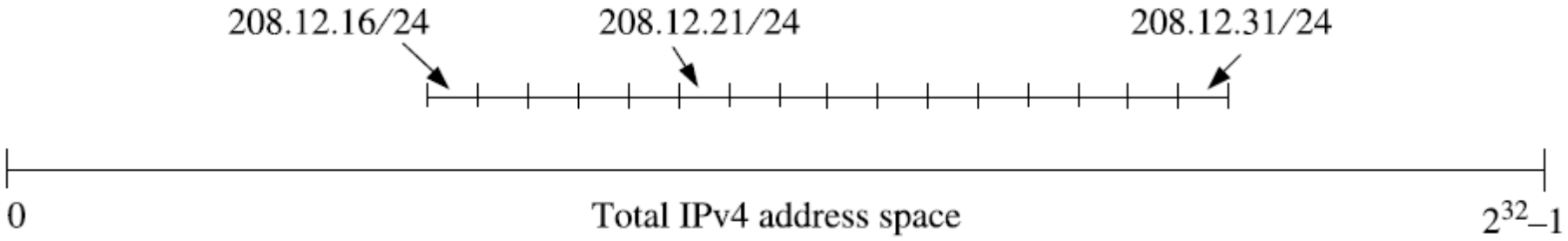
IP Routing – Classful and Classless



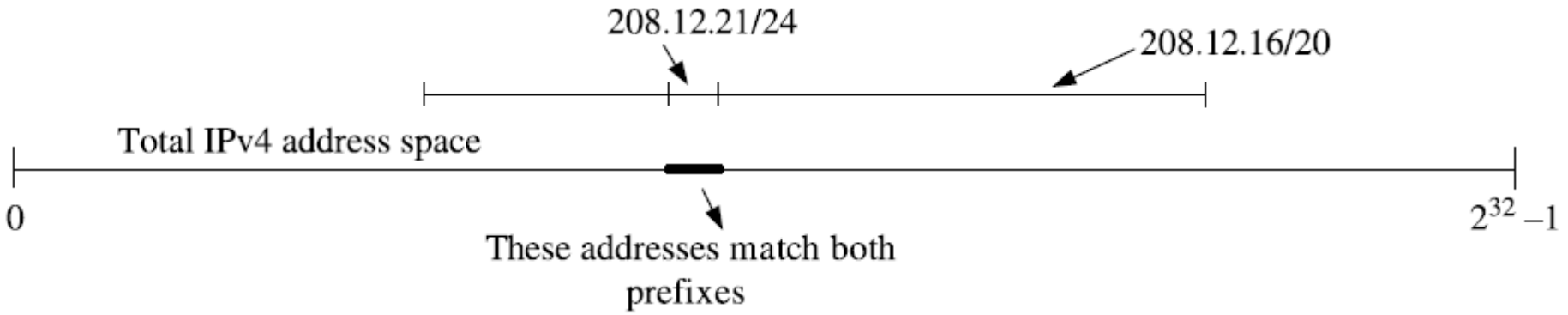
Prefix Aggregation



Prefix Ranges



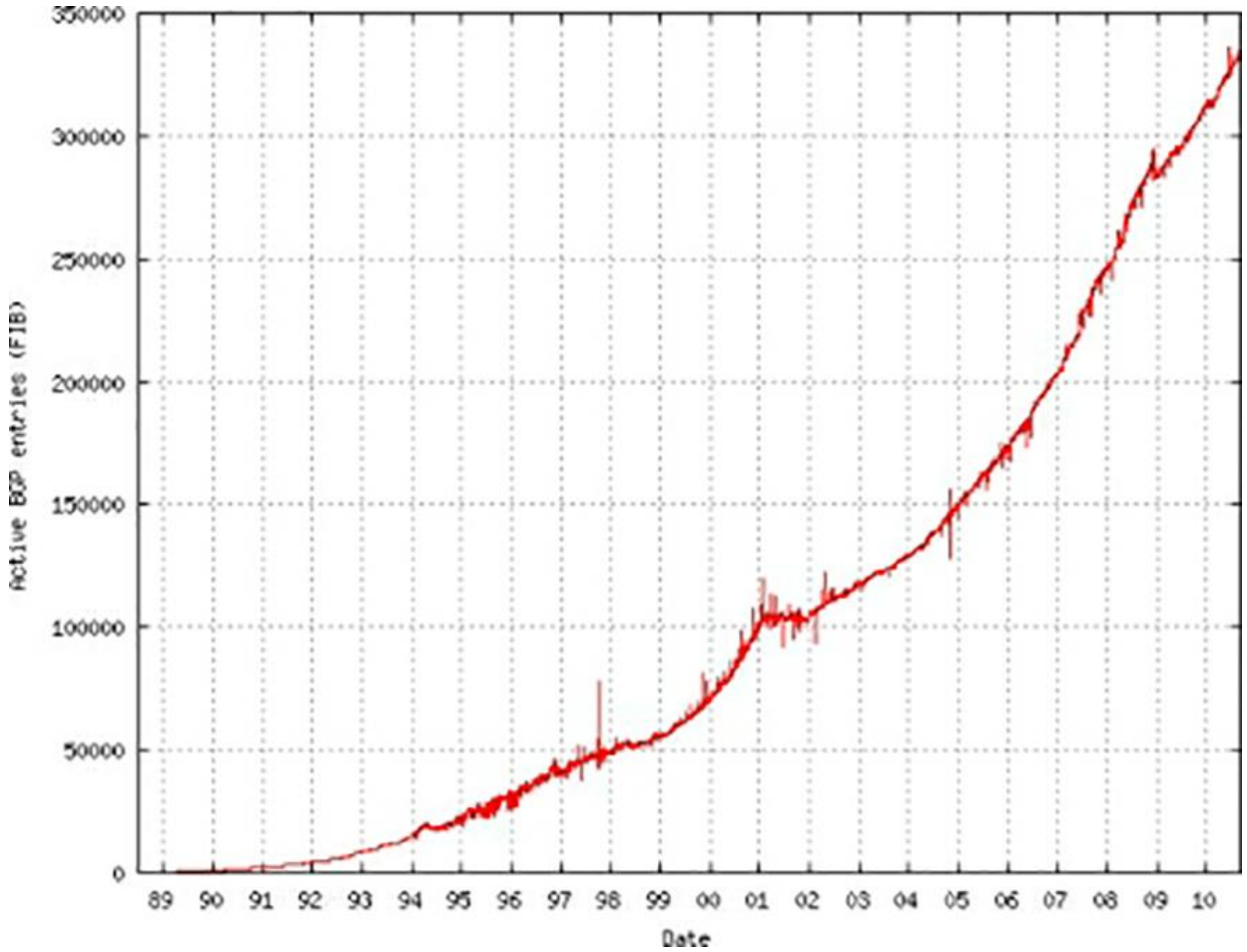
Exception Prefix



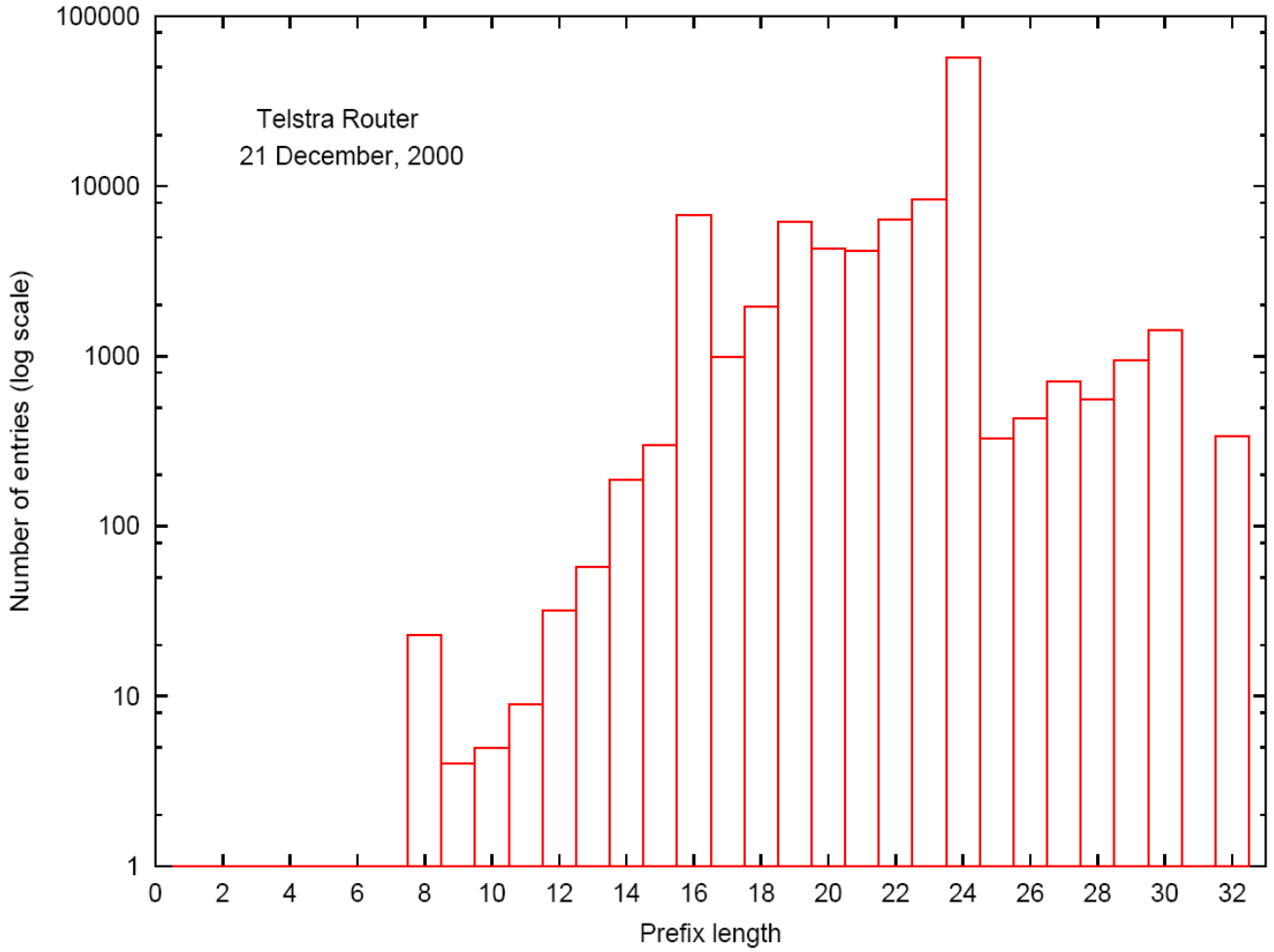
IP Routing – Classless

- Classless Interdomain Routing (CIDR)
  - Arbitrary aggregation
  - Arbitrary length for host and network fields
  - Routing entry: <prefix/length> pair
    - <12.0.54.8/32>
    - <12.0.54.0/24>
    - <12.0.0.0/16>
  - Efficient routing table size
  - Needs to find the longest match
    - Packet destination: 12.0.54.2
      - Matches: <12.0.54.0/24>, <12.0.0.0/16>
      - <12.0.54.0/24> is used
    - Makes IP route lookup a bottleneck

Table Growth of a Typical Backbone Router



Prefix Length Distribution of a Typical Backbone Router

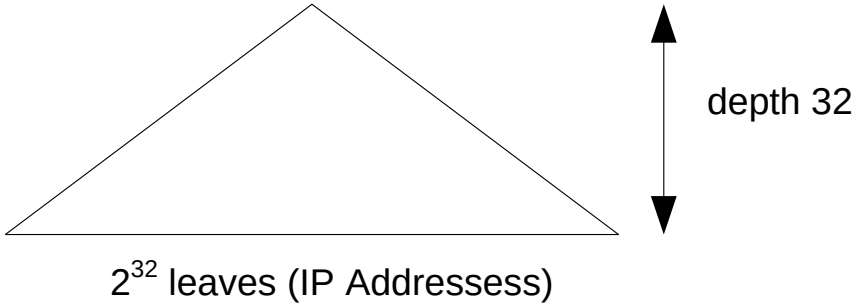


IP Address lookup design

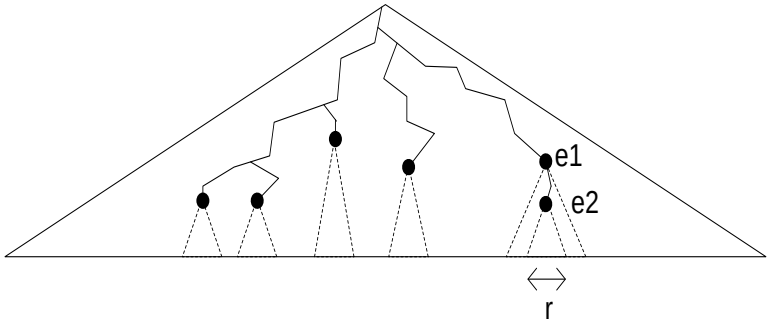
- Goals
  - Minimize time (primary goal)
    - Minimize the number of memory accesses
    - Minimize the size of the data structure
  - Minimize instructions needed
  - Aligned data structures

Address Lookup Structure

- IP address space
  - A binary tree with depth 32
  - 2^{32} leaves



- <prefix/length> pair
  - Prefix defines a path in the tree
  - Length says how deep the path goes in the tree
  - All IP addresses in the subtree are routed according that entry
  - Longest matching concept
    - Subtrees of entries e1 and e2 overlap
    - e1 is hidden by e2 for addresses in the range r



IP Address Lookup and Caching

- Using caching techniques for IP address lookup
  - Relies on locality of destination address stream
  - There is not enough locality for backbone routers
  - Not a good solution for current backbone routers

Performance Metrics for Comparison IP Address Lookup Algorithms

- Lookup Speed,
- Storage Requirement,
- Update Time,
- Scalability,
- Flexibility in Implementation.

Outlines

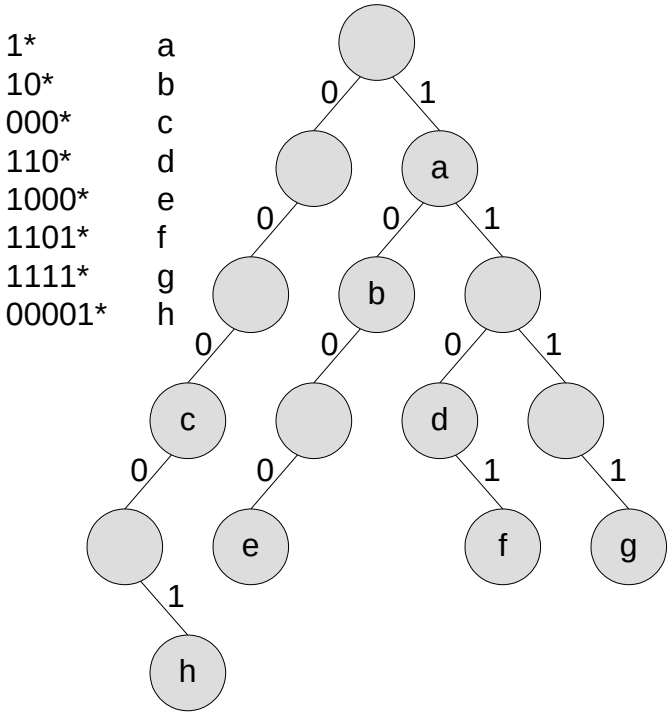
- Overview,
- **Trie-based Algorithms,**
- Hardware-based Schemes,
- IPV6 Lookup

Trie-based Algorithms

- Binary Trie,
- Path Compressed Trie,
- Multi-Bit Trie
- Level Compression,
- Tree Bitmap Algorithm
- Tree-Based Pipelined Search,
- Binary Search on Prefix Range.

Binary Trie

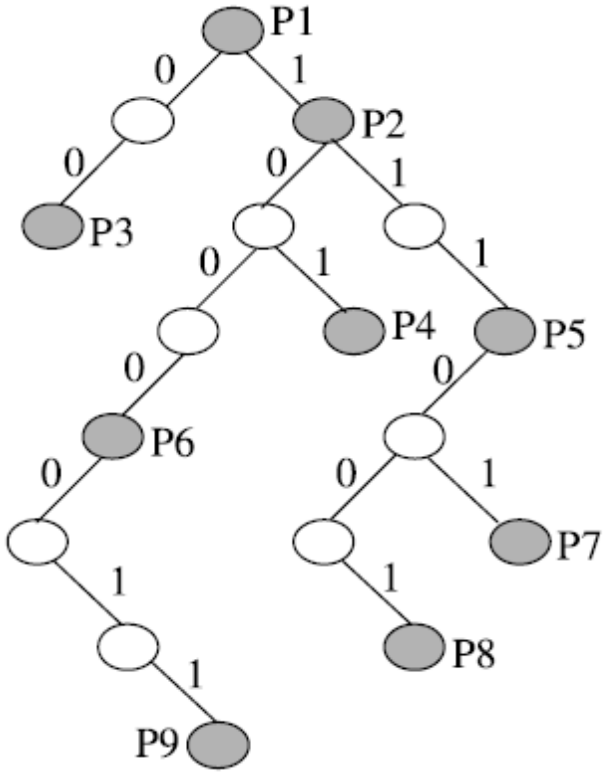
- Represents prefixes of different lengths
- 1-bit trie
  - Left link: 0
  - Right link: 1
  - Search
    - Start from root, move to left or right if the current bit of the address is 0 or 1 respectively
    - If a node containing a prefix mark (*) is seen, store it somewhere as the longest match up to now
  - Addition
    - Follow the path and create new nodes if needed and finally mark the last node as a prefix
  - Deletion
    - Follow the path and delete the last node and its parents until a marked node or a node with another child is seen



Data Structure of a 1-bit Binary Trie

Prefix database

P1	*
P2	1*
P3	00*
P4	101*
P5	111*
P6	1000*
P7	11101*
P8	111001*
P9	1000011*



Next hop information (If a prefix node)	
Left pointer	Right pointer

Data structure of a node in the binary trie

Performance of Binary Trie

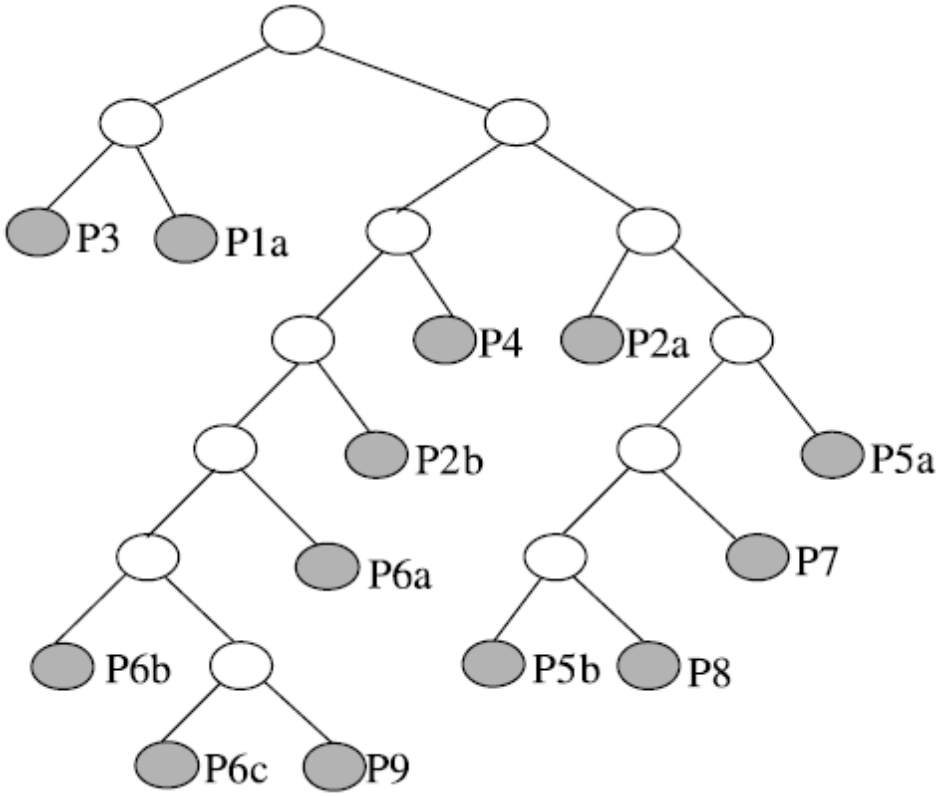
- The Number of Memory Accesses in The Worst Case is 32 for IPv4.
- To Add a Prefix to The Trie, In The Worst Case It Needs to Add 32 Nodes. In This Case, The Storing Complexity is $32N \cdot S$,
- The Lookup Complexity is $O(W)$,
- The Storagecomplexity is $O(NW)$

Variants of Binary Tries

Disjoint-prefix binary trie

Prefix database

P1	*
P2	1*
P3	00*
P4	101*
P5	111*
P6	1000*
P7	11101*
P8	111001*
P9	1000011*



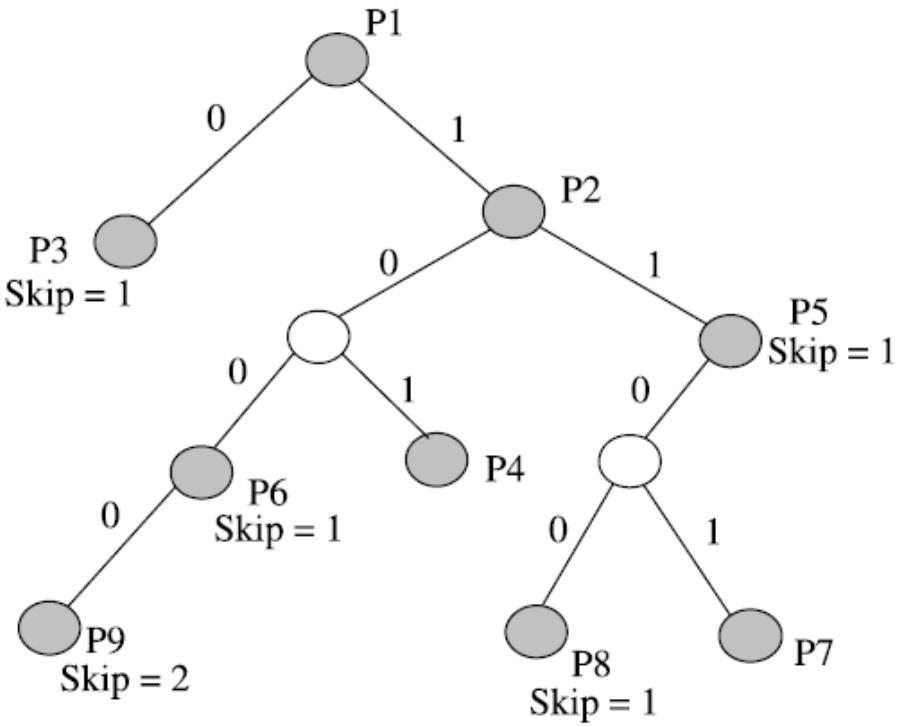
Path-Compressed Trie

Prefix database

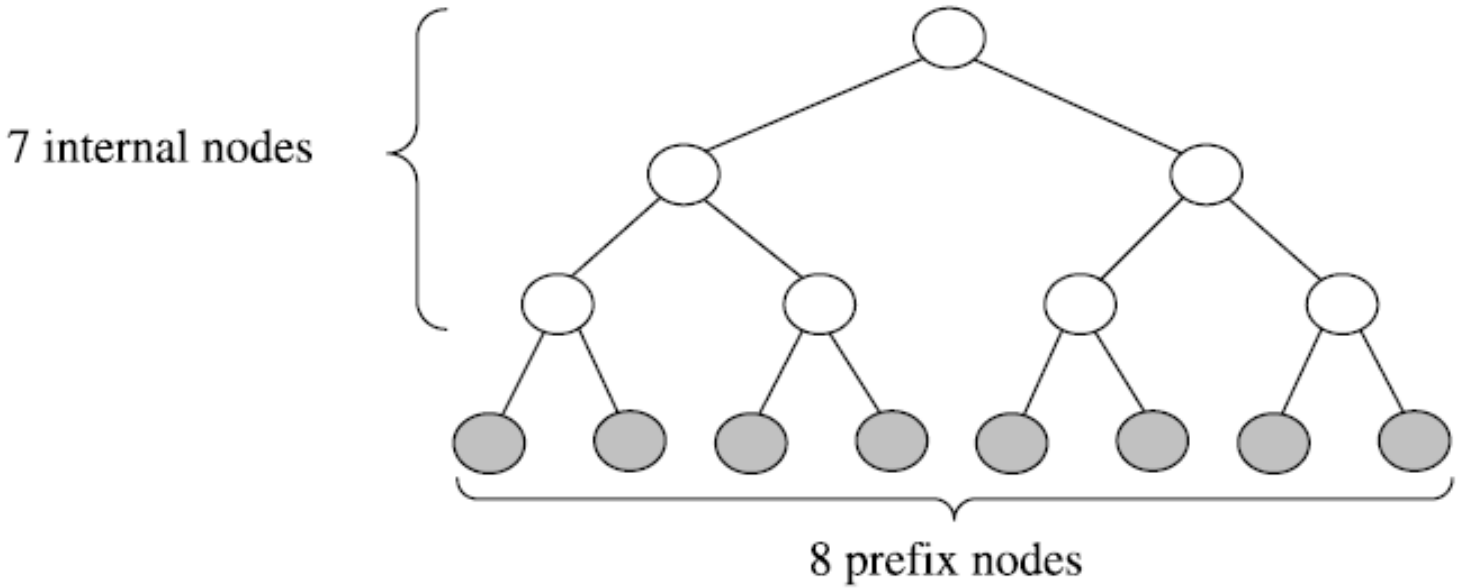
P1	*
P2	1*
P3	00*
P4	101*
P5	111*
P6	1000*
P7	11101*
P8	111001*
P9	1000011*

Next hop information (If a prefix node)	
Skip value	Segment
Left pointer	Right pointer

Data structure of a node in the path-compressed trie



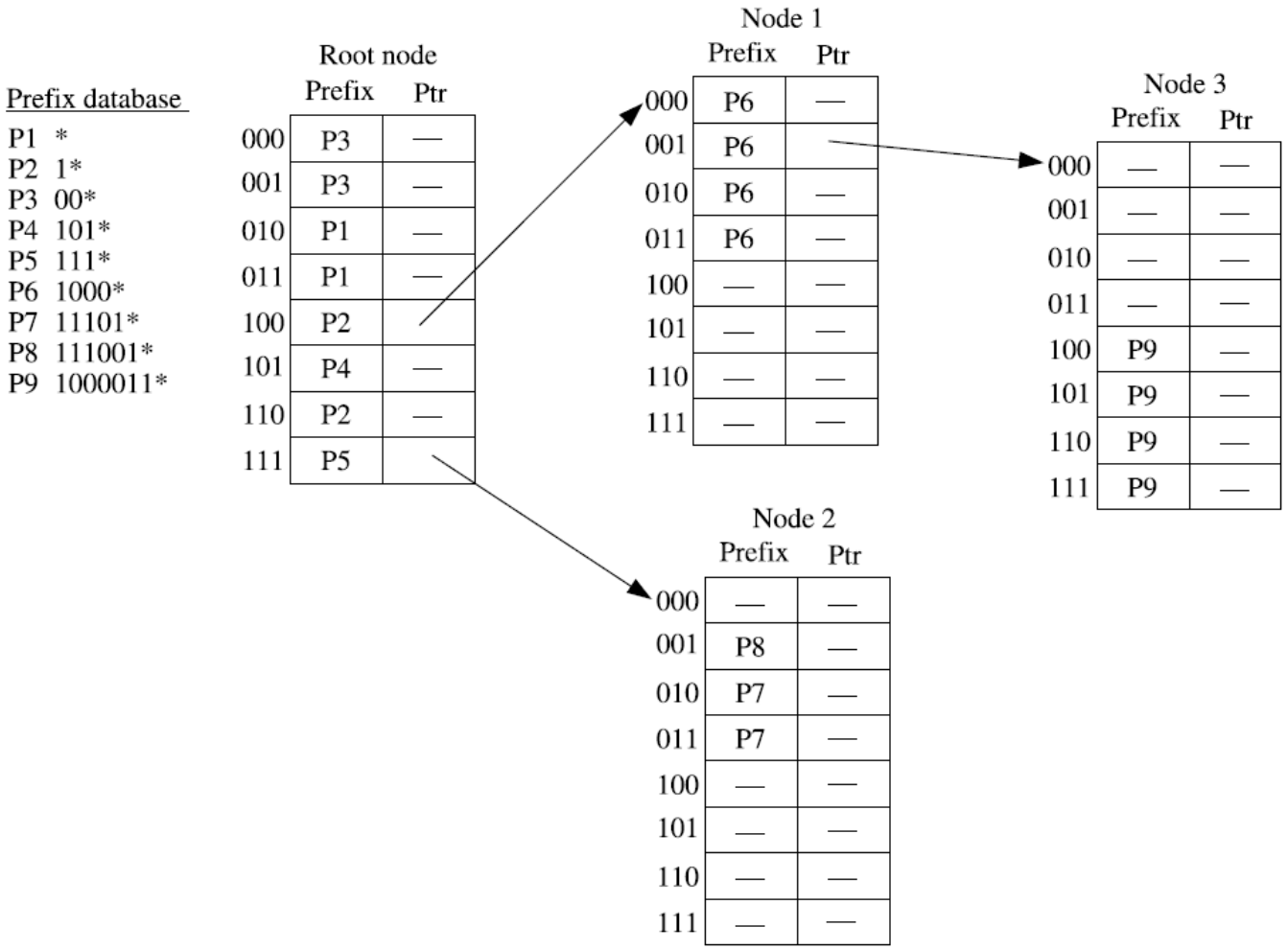
Example of Path-Compressed Trie with N Leaves



Performance of Path-compressed Tire

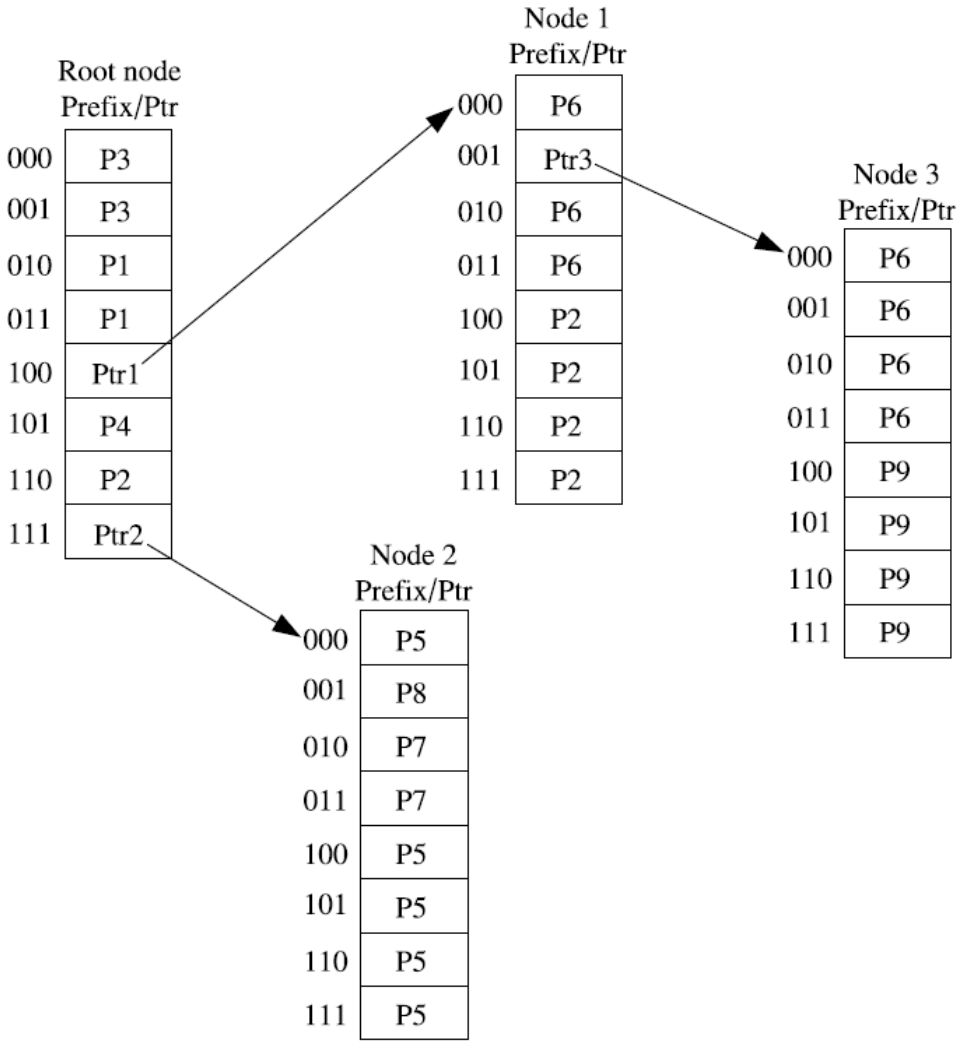
- The Lookup and Update (the worst case) Complexity is *O(W)*,
- The Space complexity is *O(N)*

Multi-Bit Trie



Multi-Bit Trie

Example With Each Entry a Prefix or a Pointer to Save Memory Space

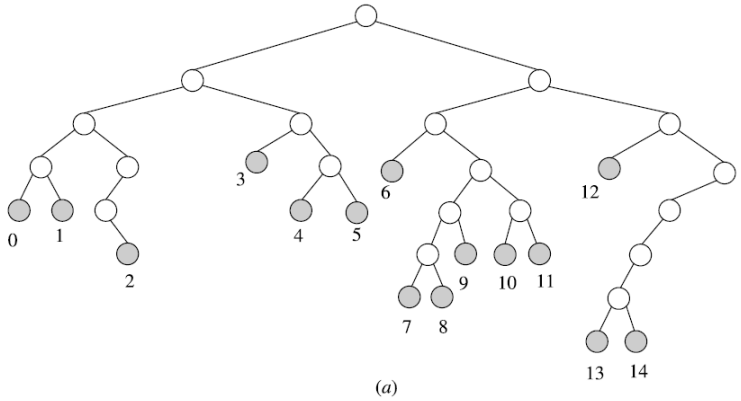


Performance of Multi-Bit Trie

- The Lookup Complexity is *O(W/k)*,
- The Update Complexity is $O(W/k+2^k)$,
- The Space Complexity is $O((2^k*N*W)/k)$

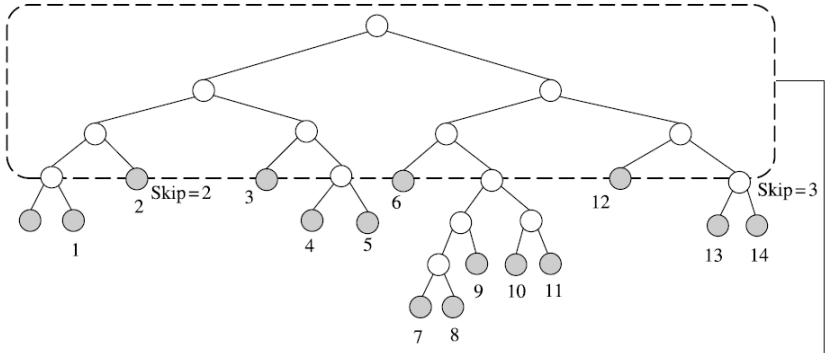
Level Compression Trie

(a) One-bit trie;



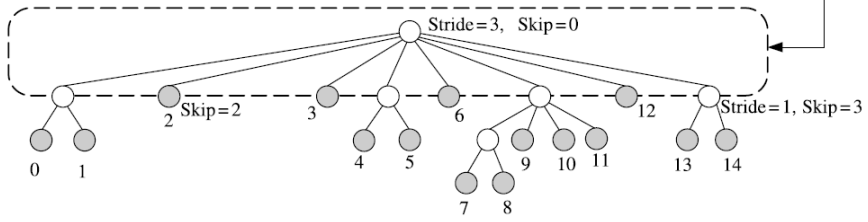
(a)

(b) Path-compressed trie;



(b)

(c) LC-trie.

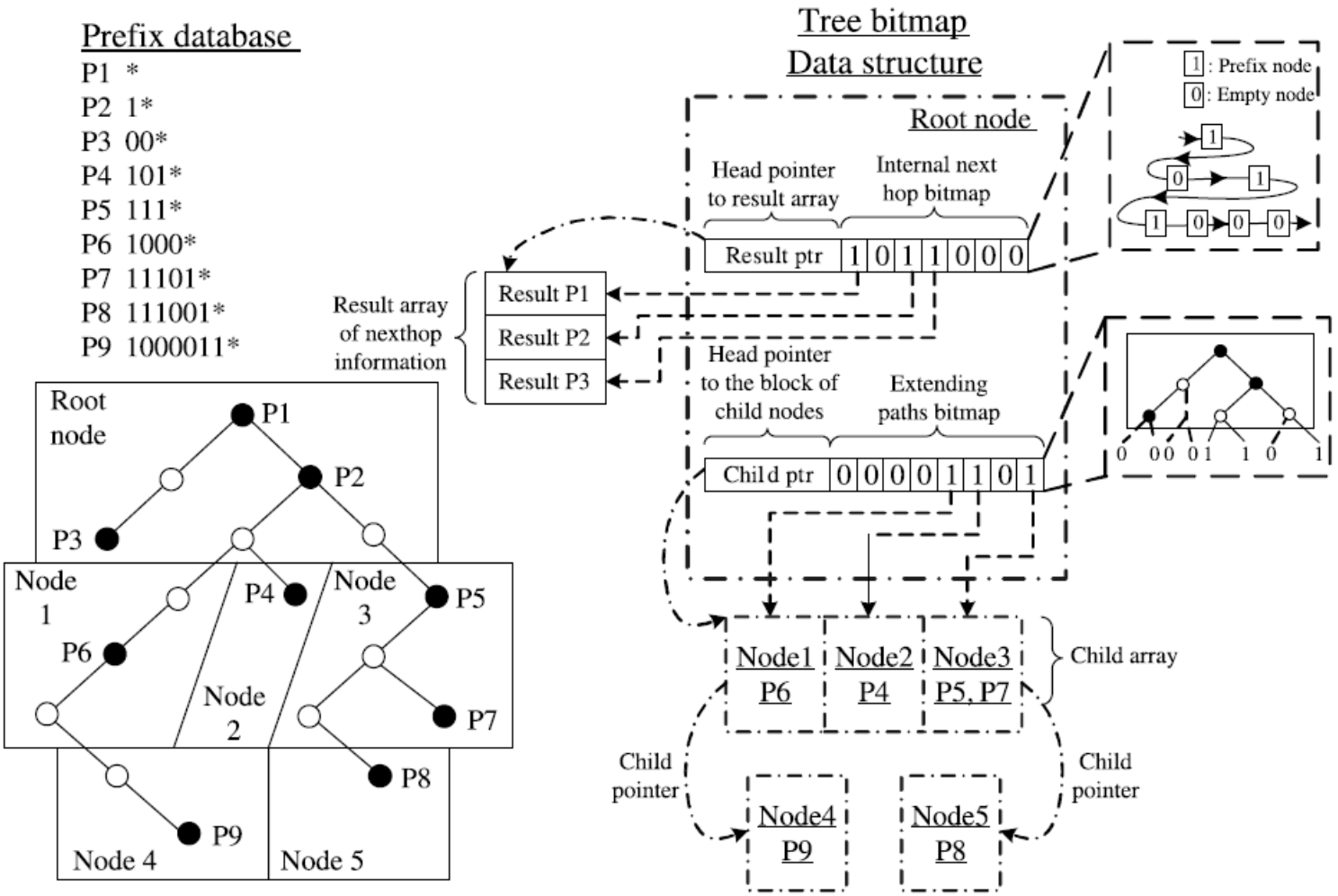


(c)

Performance of Level Compression Trie

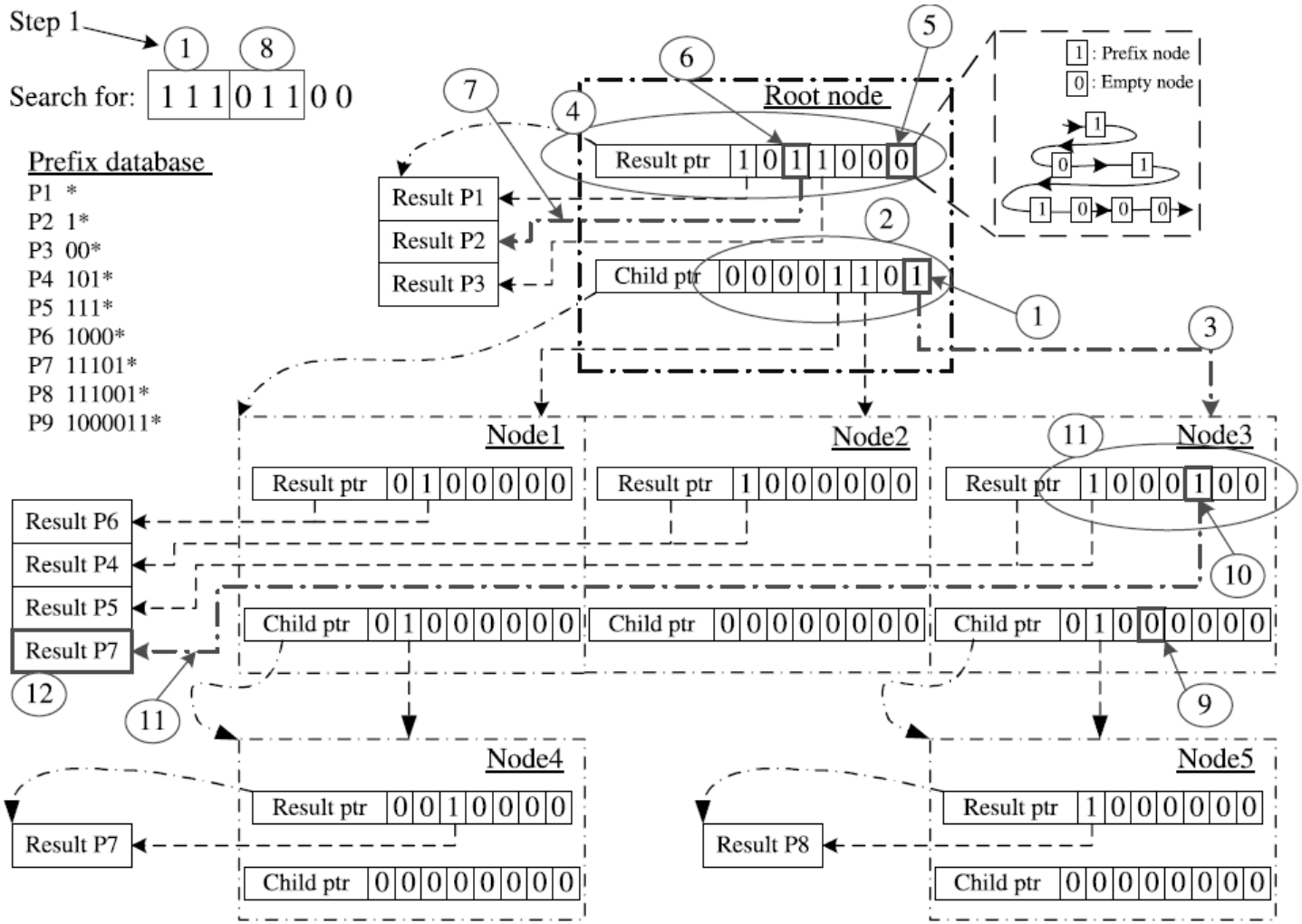
- The Lookup Complexity is *O(W/k)*,
- The Update Complexity is $O(W/k+2^k)$,
- The Space Complexity is $O((2^k*N*W)/k)$

Tree Bitmap Algorithm



Tree Bitmap Algorithm

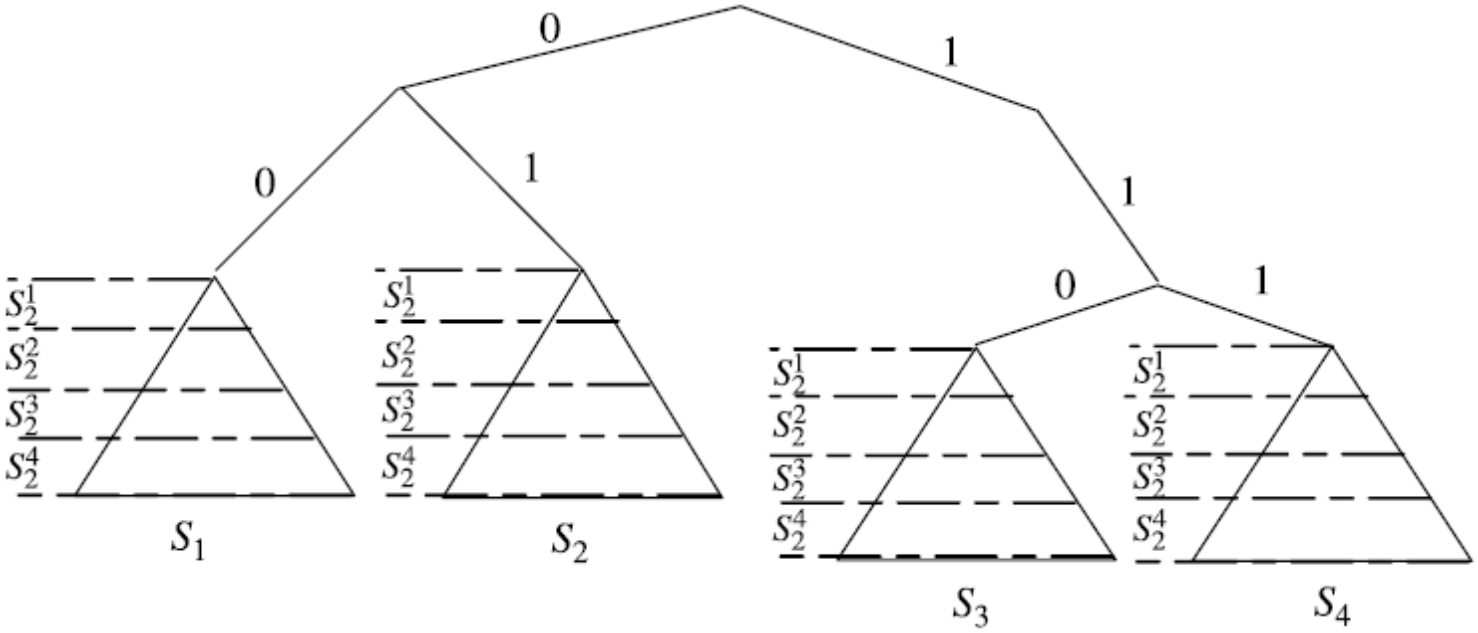
Example of route lookup



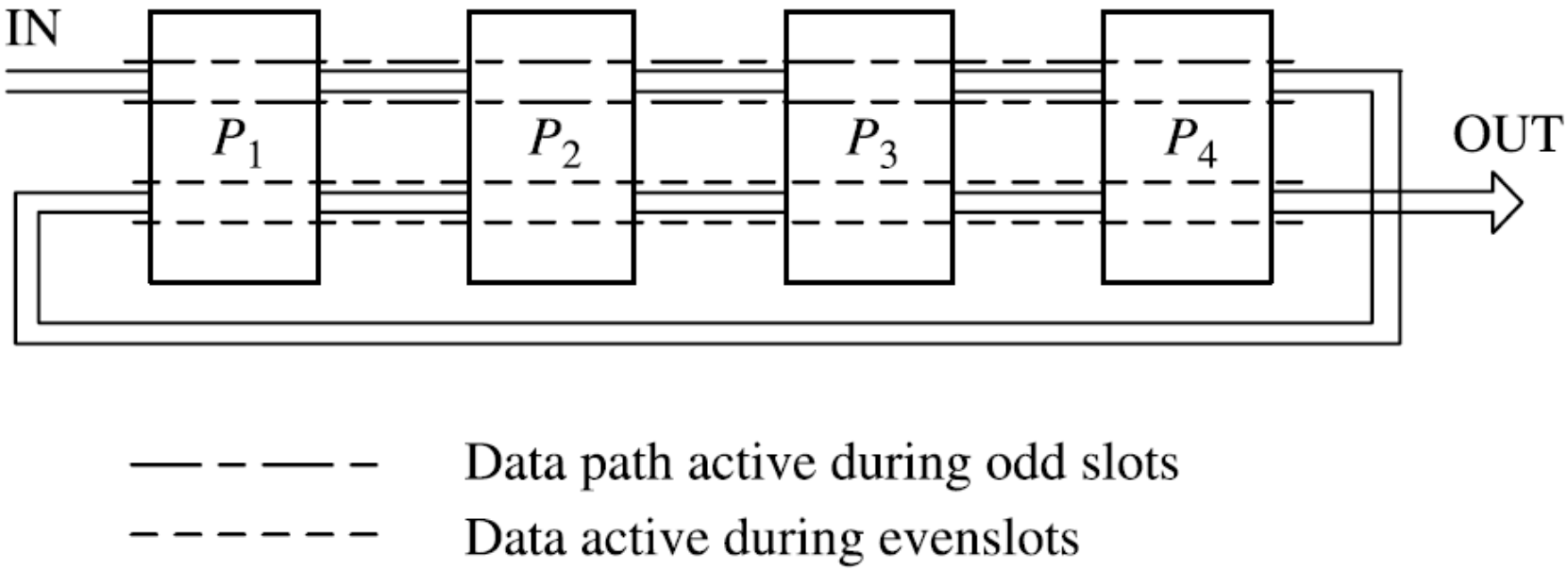
Performance of Tree Bitmap

- The Lookup Complexity (the worst case) is *O(W/k)*,
- The Memory Space Complexity is $O((2^k * N * W)/k)$

Tree-Based Pipelined Search



Tree-Based Pipelined Search

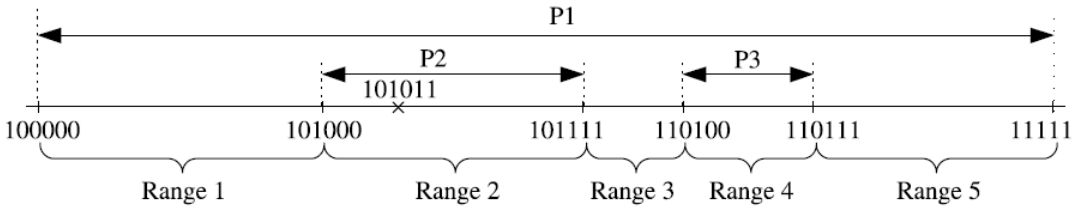


Random ring pipeline architecture with two data paths

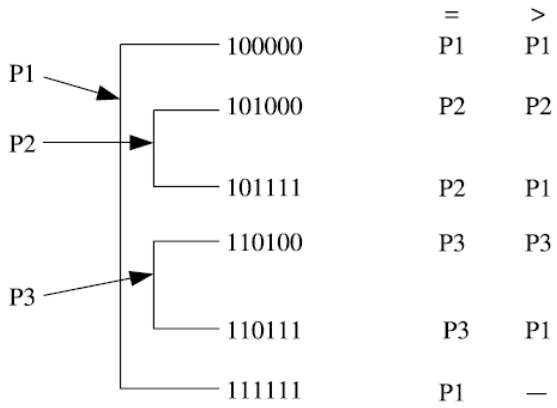
Binary Search on Prefix Range

Prefix database	Range
P1: 1*	100000 –111111
P2: 101*	101000 –101111
P3: 1101*	110100 –110111

(*a*)



(*b*)



(*c*)

Performance of Binary Search on Prefix Lengths

- The Lookup Complexity (the worst case) is $O(\log_k 2N)$,
- The Update Complexity is $O(N)$
- The Memory Space Complexity is $O(N)$

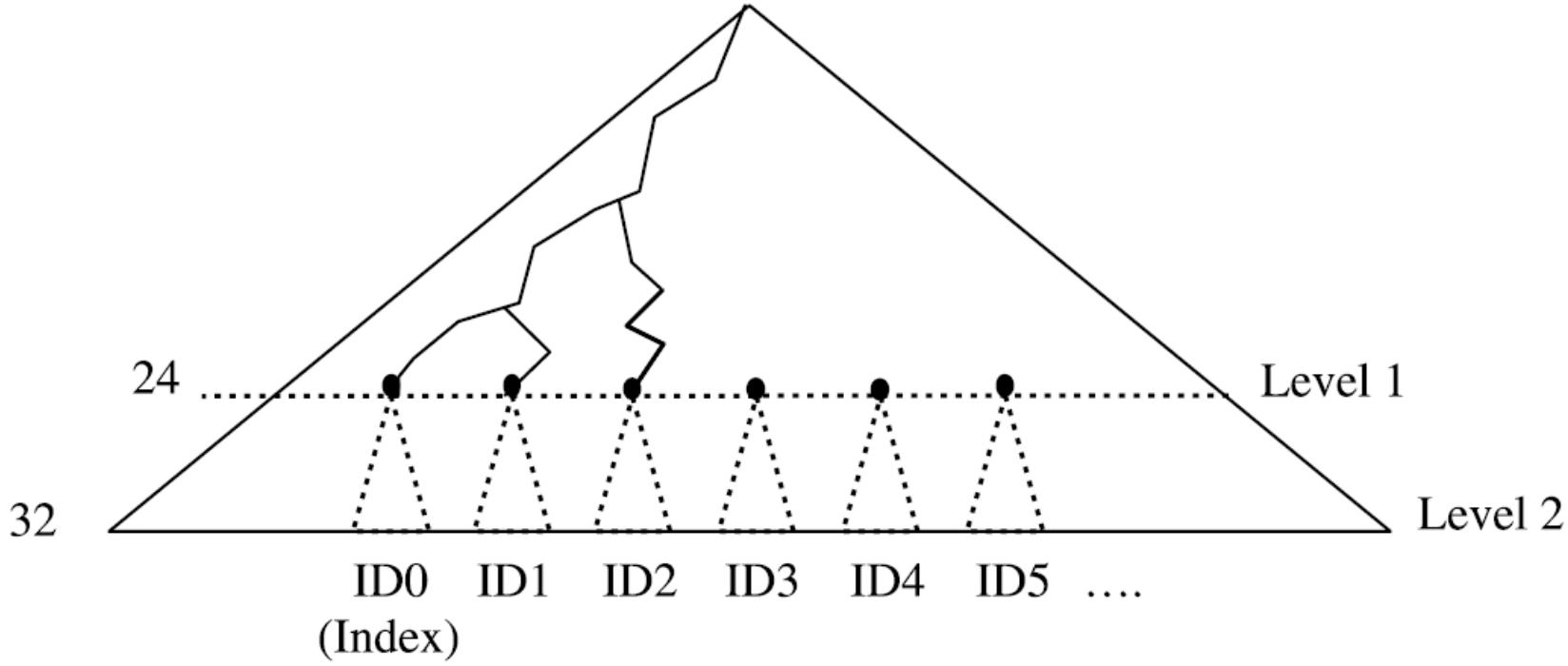
Outlines

- Overview,
- Trie-based Algorithms,
- **Hardware-based Schemes,**
- IPV6 Lookup

Hardware-based Algorithms

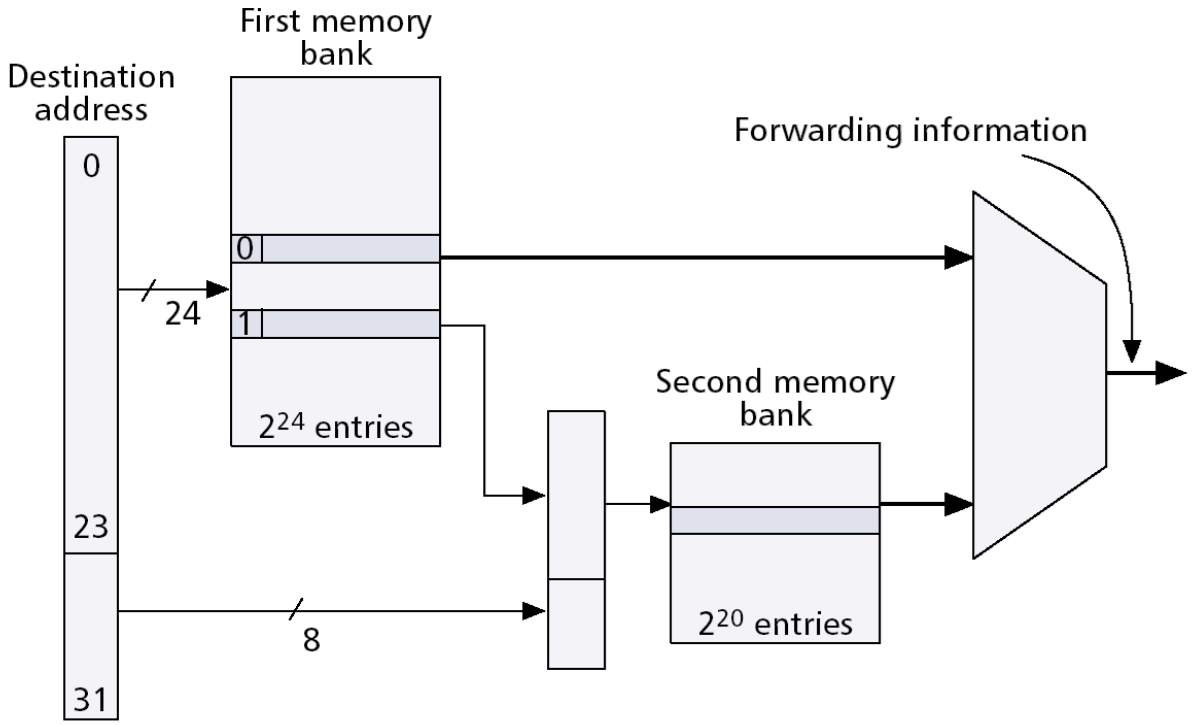
- DIR-24-8-BASIC Scheme
- DIR-Based Scheme with Bitmap Compression (BC-16-16)
- Ternary CAM for Route Lookup
- The Algorithms for ReducingTCAM Entries
- ReducingTCAM Power – CoolCAMs
- TCAM-Based Distributed Parallel Lookup

DIR-24-8 BASIC Scheme



DIR-24-8 implementation

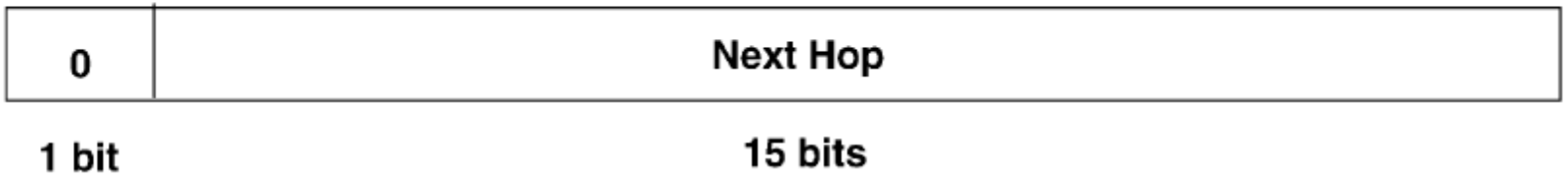
- Gupta et al.
- Two levels
  - First memory bank: 24 bits of address
  - Second memory bank: 8 bits of address



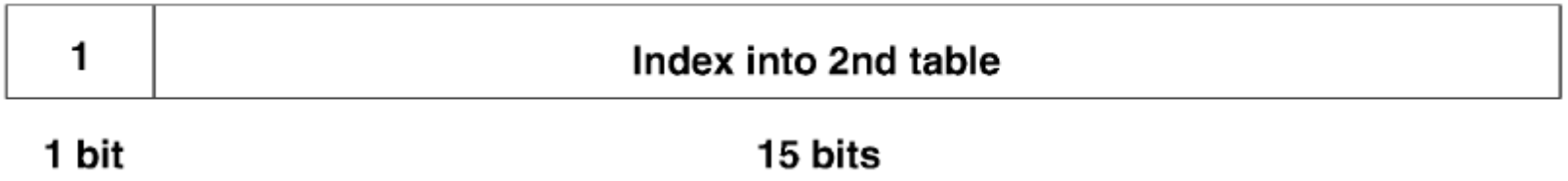
DIR-24-8 implementation

TBL24 entry format.

If longest route with this 24-bit prefix is < 25 bits long :

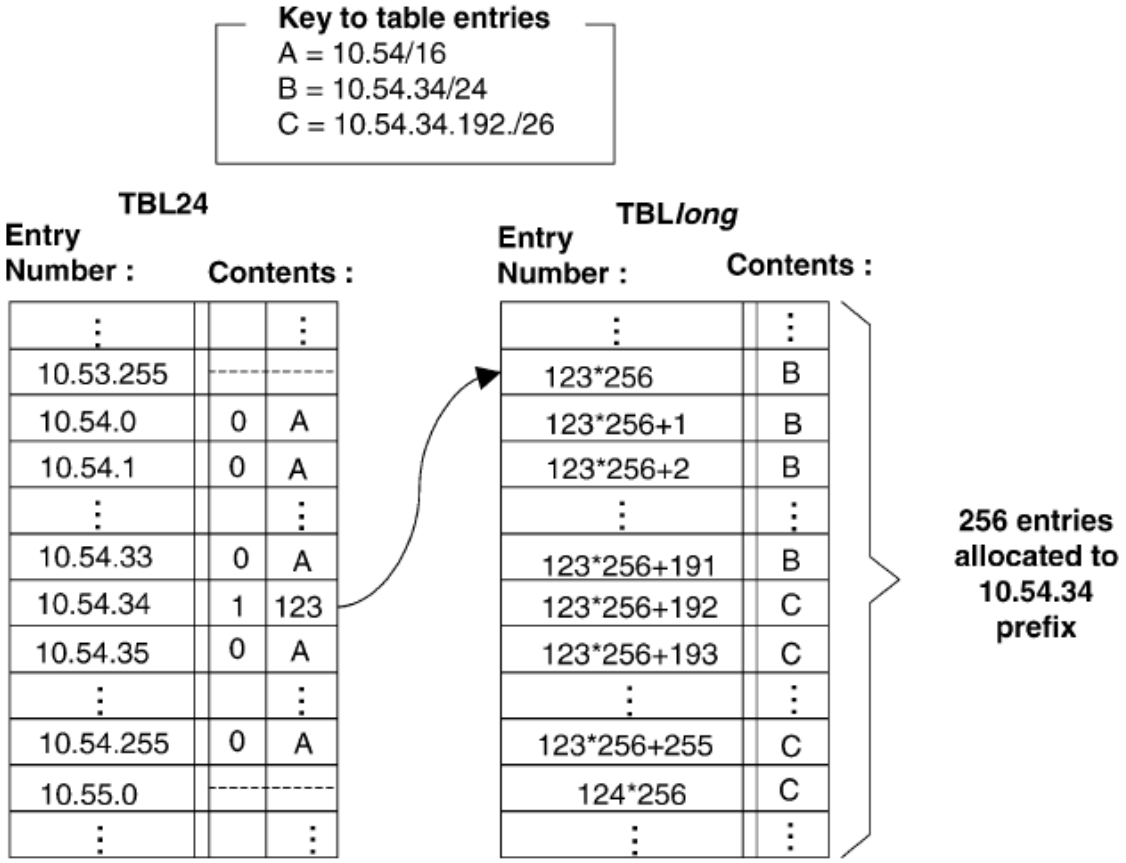


If longest route with this 24-bit prefix is > 24 bits long :



DIR-24-8 implementation

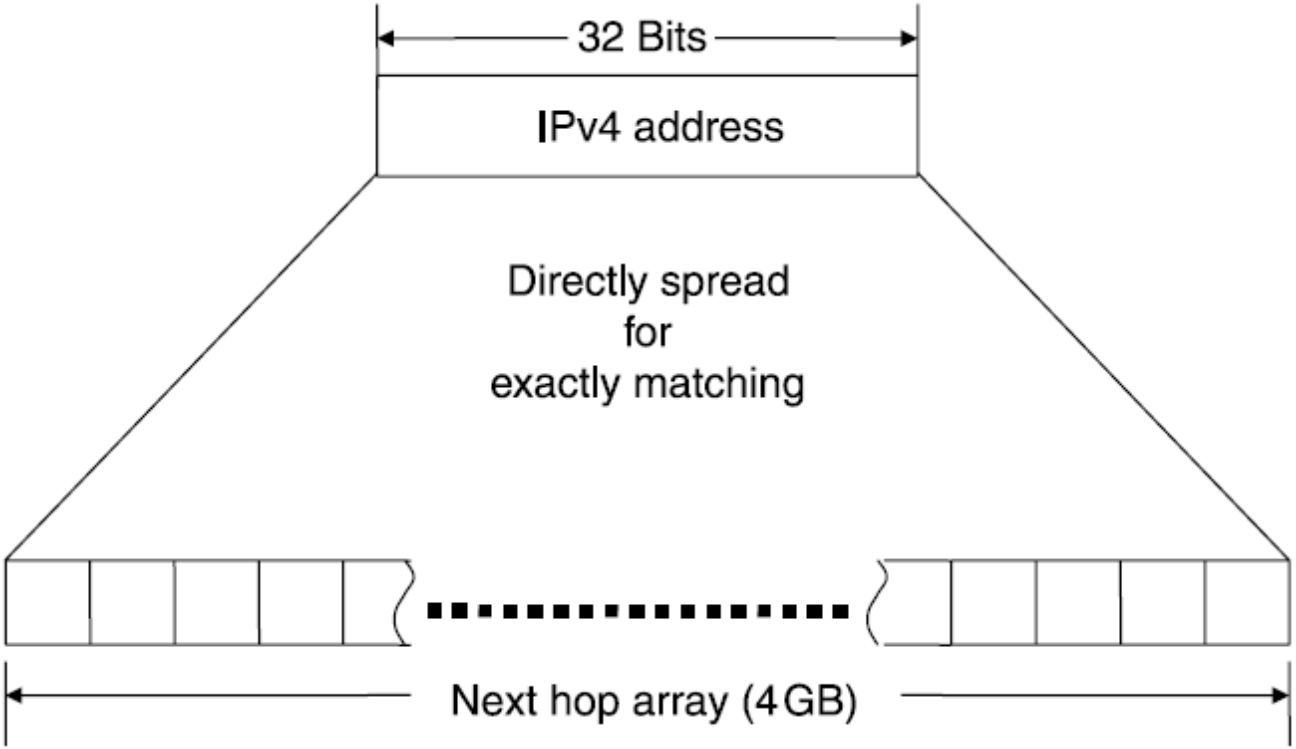
Example of two tables containing three routes.



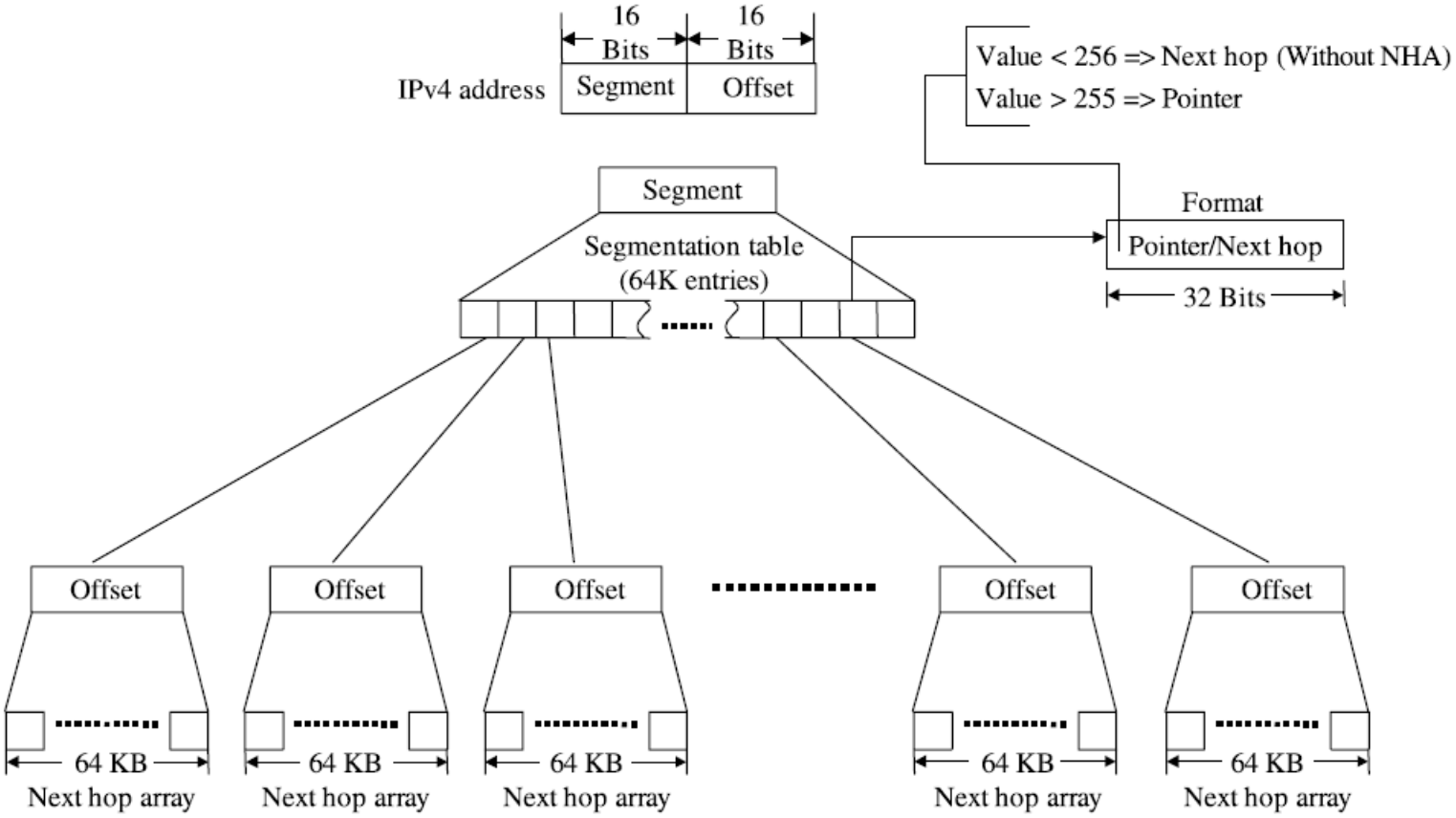
DIR-24-8 implementation

- Performance
  - Two pipelined memory accesses per lookup
  - DRAM delay of 50ns => 20 mlps
  - 33 Mbytes of DRAM
- Drawbacks
  - High memory usage
  - Many memory places may need to change for an update

DIR-Based Scheme with Bitmap Compression



DIR-Based Scheme with Bitmap Compression



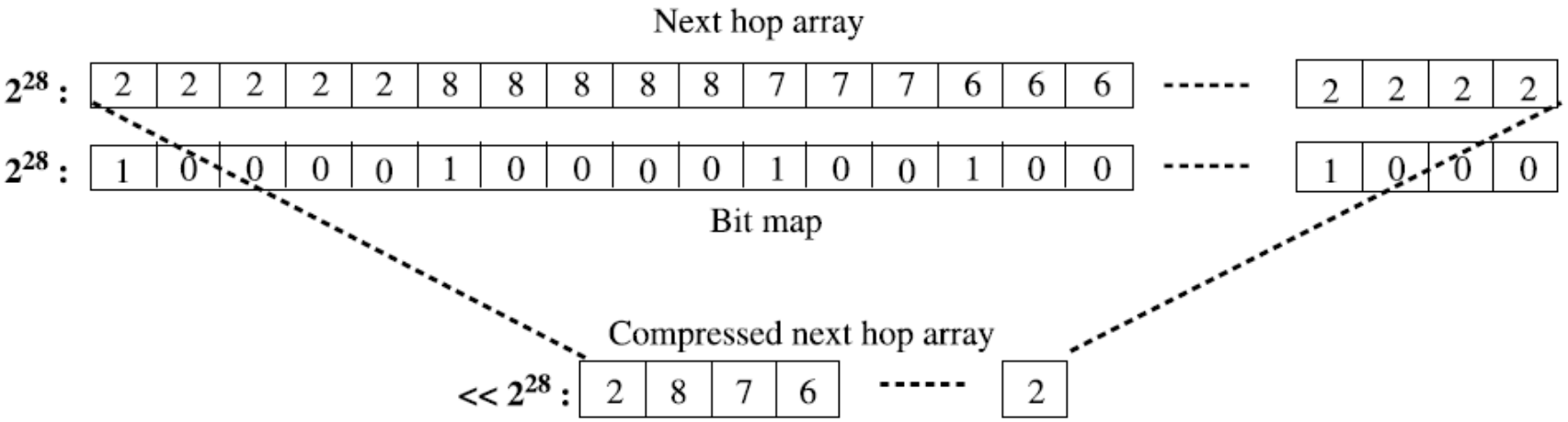
DIR-Based Scheme with Bitmap Compression

Segment table entry format.

F (1 bit)	*Pointer/Next hop (27 bits)*	*Offset length k (4 bits)*

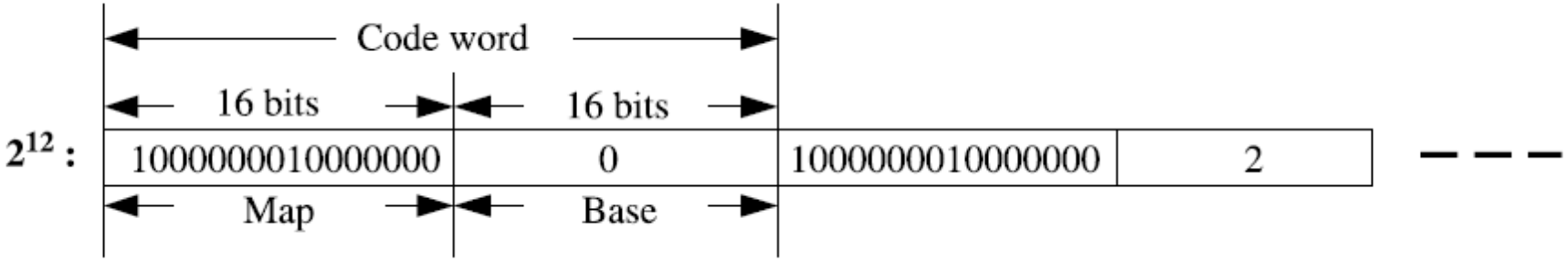
DIR-Based Scheme with Bitmap Compression

Compressed next hop array (CNHA)

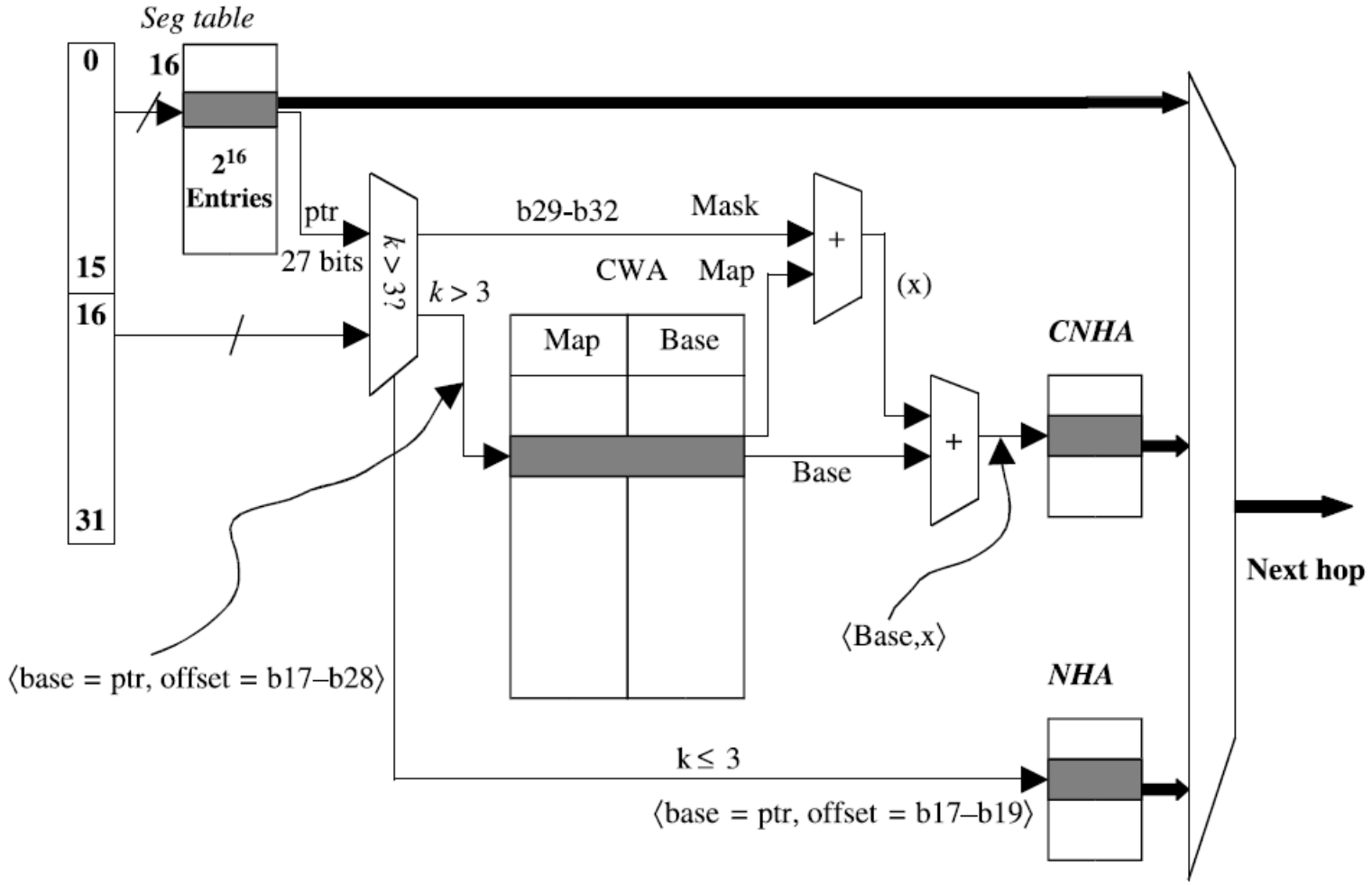


DIR-Based Scheme with Bitmap Compression

Code word array (CWA).



DIR-Based Scheme with Bitmap Compression

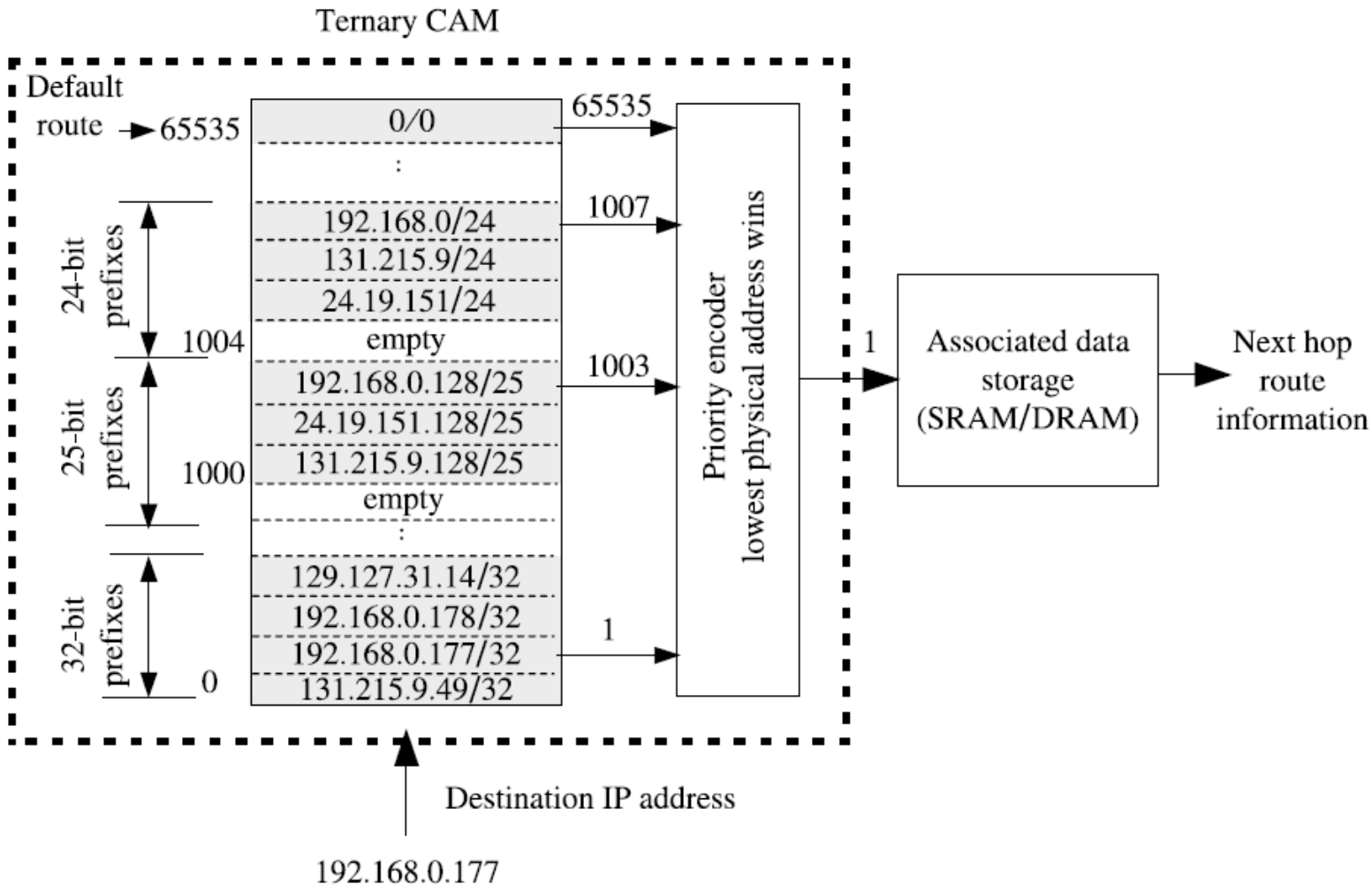


DIR-Based Scheme with Bitmap Compression

Performance.

- BC-16-16 needs only a tiny amount of SRAM and can be easily implemented in hardware.
- A large forwarding table with 40,000 routing entries can be compacted to a forwarding table of 450–470 kbytes.
- Most of the address lookups can be done by one memory access.
- In the worst case, the number of memory accesses for a lookup is three.
- When implemented in a pipeline in hardware, the proposed mechanism can achieve one route lookup every memory access.
- This mechanism furnishes approximately 100M route lookups per second with current 10 ns SRAM.

Ternary CAM for Route Lookup



Ternary CAM for Route Lookup

Performance.

- TCAM returns the result of the longest matching prefix lookup within only one memory access, which is independent of the width of the search key.
- The commercially available TCAM chips can integrate 18 M-bit (configurable to 256 k × 36-bit entries) into a single chip working at 133 MHz, which means it can perform up to 133 million lookups per second.
- However, the TCAM approach has the disadvantage of high cost-to-density ratio and high-power consumption (10–15Watts/chip).

Two Algorithms for ReducingTCAM Entries

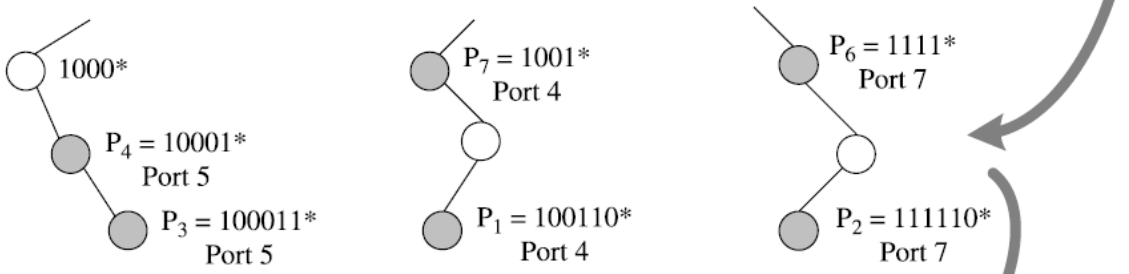
- ***Prefix Pruning.***
- ***Mask Extension.***

Two Algorithms for ReducingTCAM Entries

Prefix Pruning

#	Prefix	Mask	Next hop port
P_1	10011000	11111100	4
P_2	11111100	11111100	7
P_3	10001100	11111100	5
P_4	10001000	11111000	5
P_5	11010000	11110000	4
P_6	11110000	11110000	7
P_7	10010000	11110000	4

#	Prefix	Next hop port
P_1	100110*	4
P_2	111110*	7
P_3	100011*	5
P_4	10001*	5
P_5	1101*	4
P_6	1111*	7
P_7	1001*	4

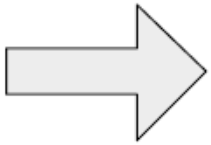


#	Prefix	Mask	Next hop port
P_3 & P_4	10001000	11111000	5
P_1 & P_7	10010000	11110000	4
P_2 & P_6	11110000	11110000	7
P_5	11010000	11110000	4

Two Algorithms for ReducingTCAM Entries

Mask Extension

#	Prefix	Mask	Next hop port
P_1	10011100	11111100	7
P_2	10001100	11111100	7
P_3	11011100	11111100	7
P_4	10001000	11111000	5
P_5	11010000	11110000	4
P_6	11110000	11110000	7
P_7	10010000	11110000	4



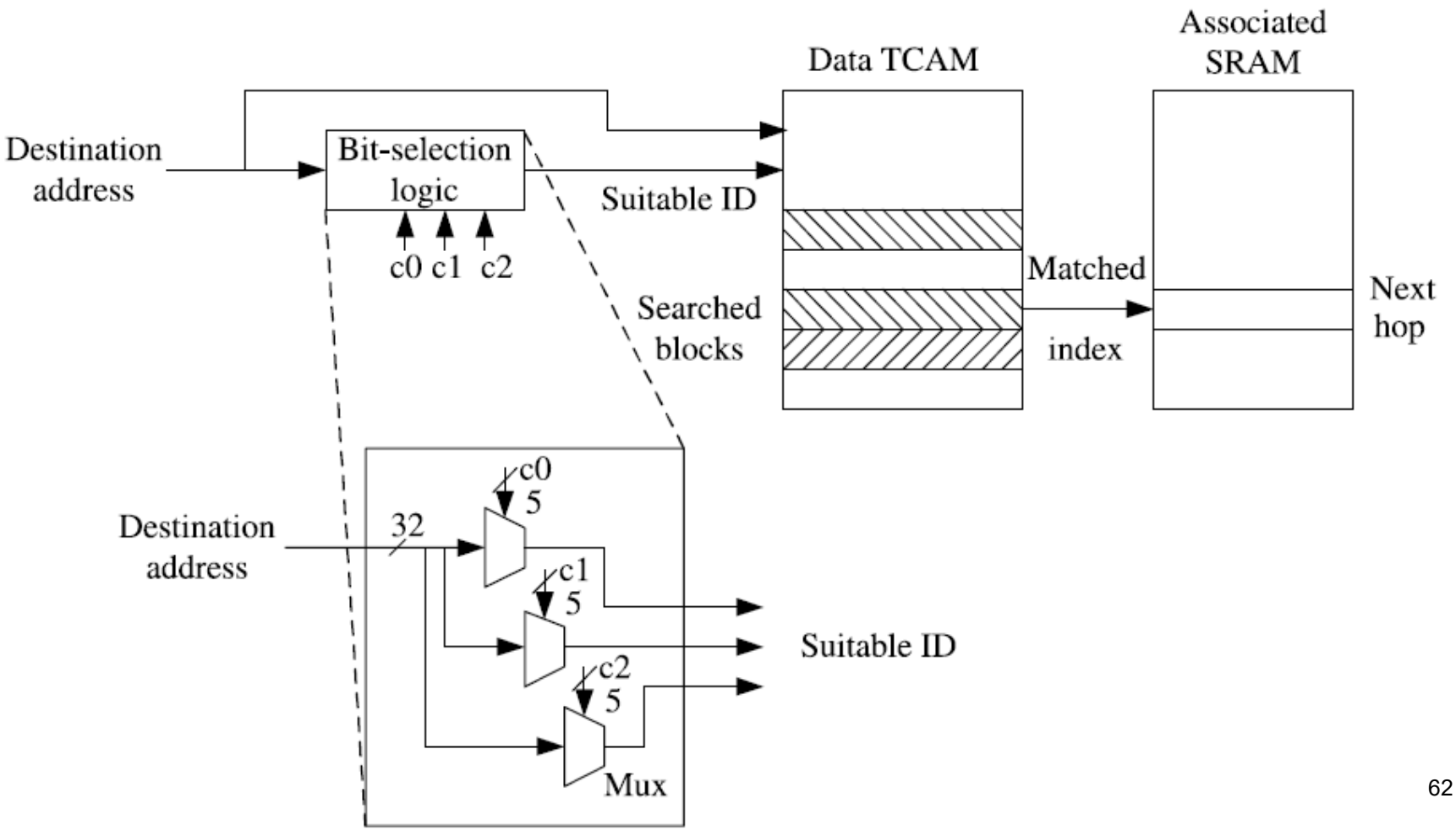
#	Prefix	Mask	Next hop port
P_1 & P_2	10001100	11101100	7
P_1 & P_3	10011100	10111100	7
P_4	10001000	11111000	5
P_5 & P_7	10010000	10110000	4
P_6	11110000	11110000	7

Two Algorithms for ReducingTCAM Entries

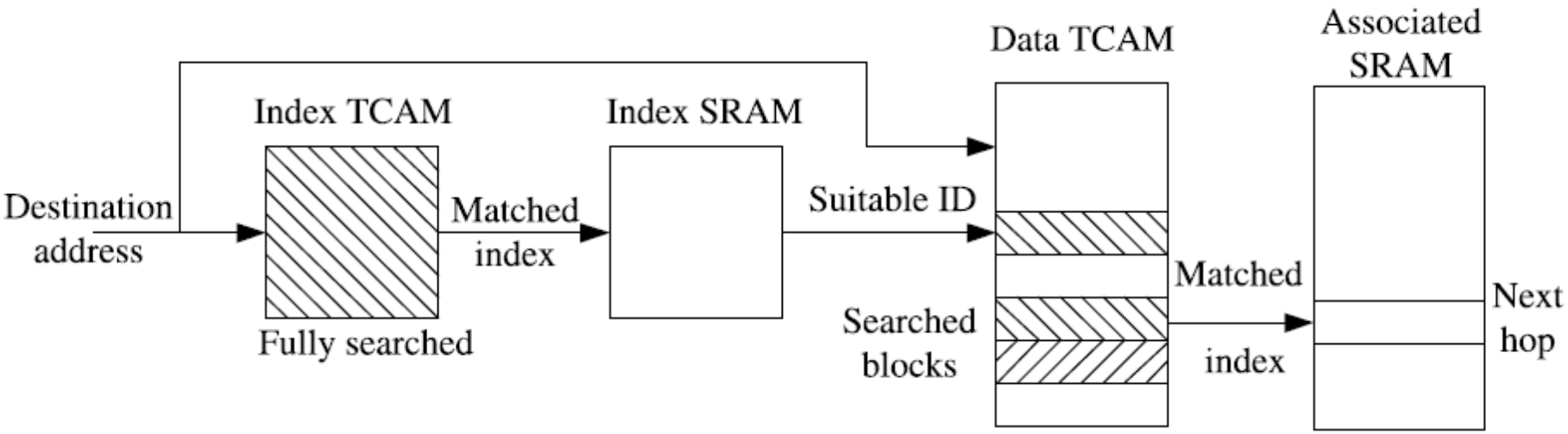
Performance.

- Real-life forwarding tables can be represented by much fewer TCAM (ternary match) entries, typically 50–60 percent of the original size by using the above two techniques. Prefix pruning would cause no change in prefix update complexity.
- Mask extension increases update complexity. Many prefixes are associated with others after mask extension, which results in the obstacles of performing incremental updates.

ReducingTCAM Power – CoolCAMs
Bit-Selection Architecture



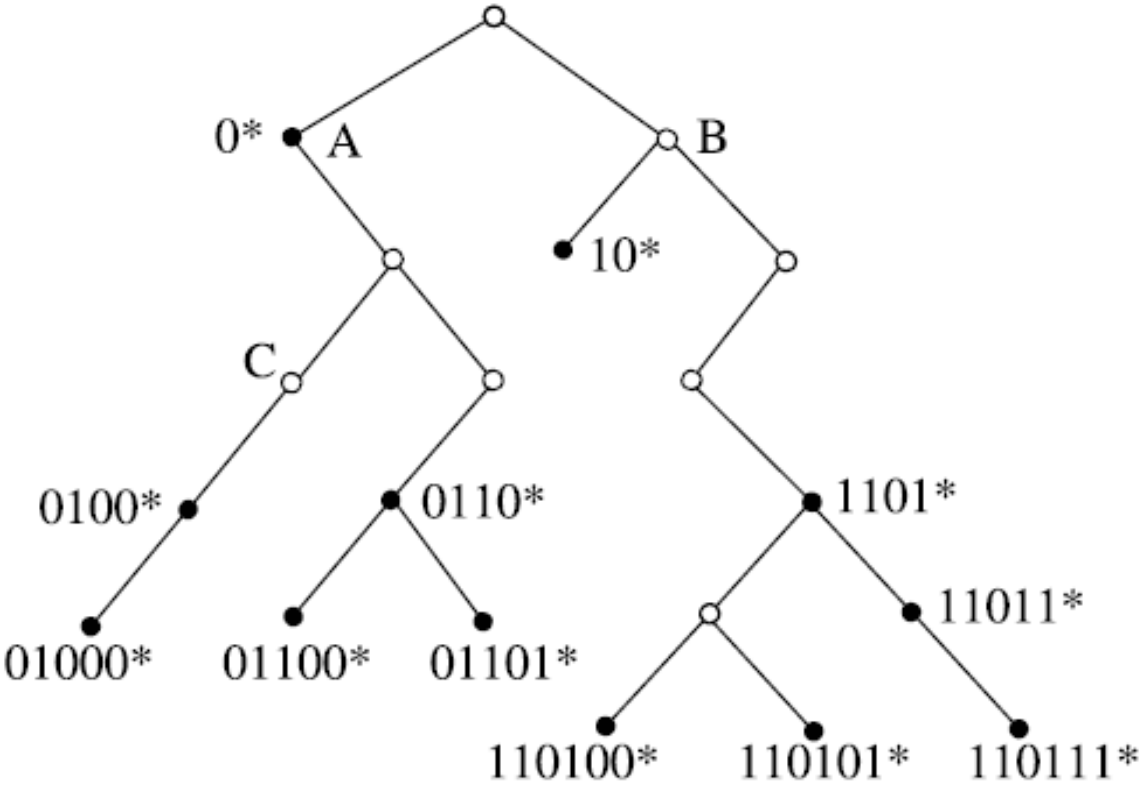
ReducingTCAM Power – CoolCAMs
Trie-Based Table Partitioning



ReducingTCAM Power – CoolCAMs
Trie-Based Table Partitioning

0*
0100*
01000*
0110*
01100*
01101*
10*
1101*
110100*
110101*
11011*
110111*

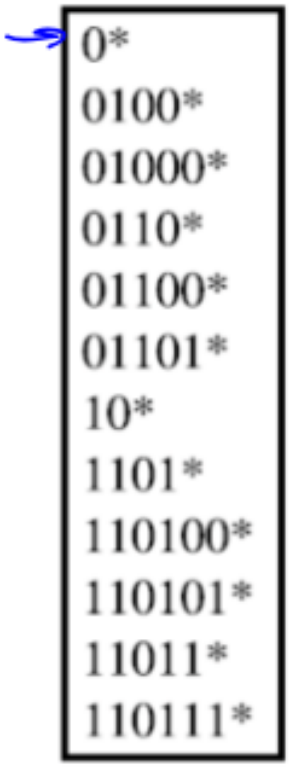
(*a*)



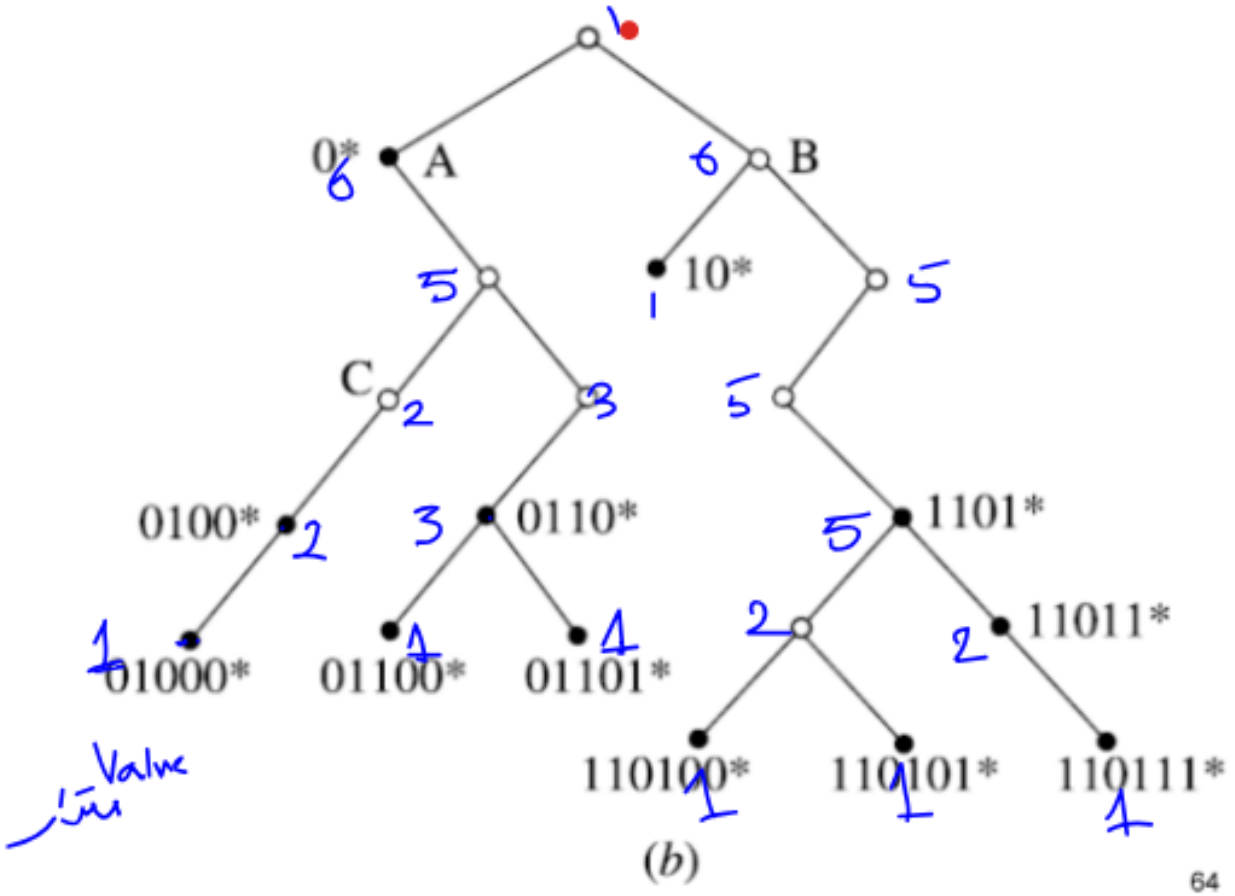
(*b*)

ReducingTCAM Power – CoolCAMs

Trie-Based Table Partitioning

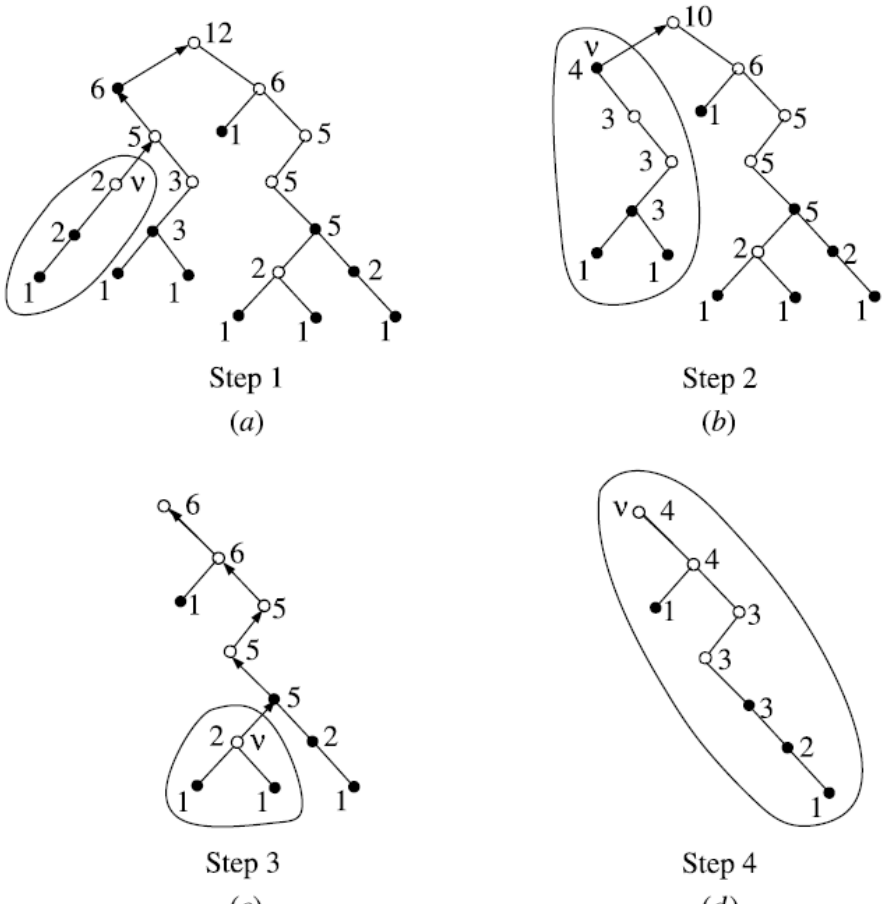


(a)



(b)

Subtree-Split Algorithm



Four iterations of the subtree-split algorithm (with parameter *b* set to 4) applied to the 1-bit trie from previous figure.

Playback

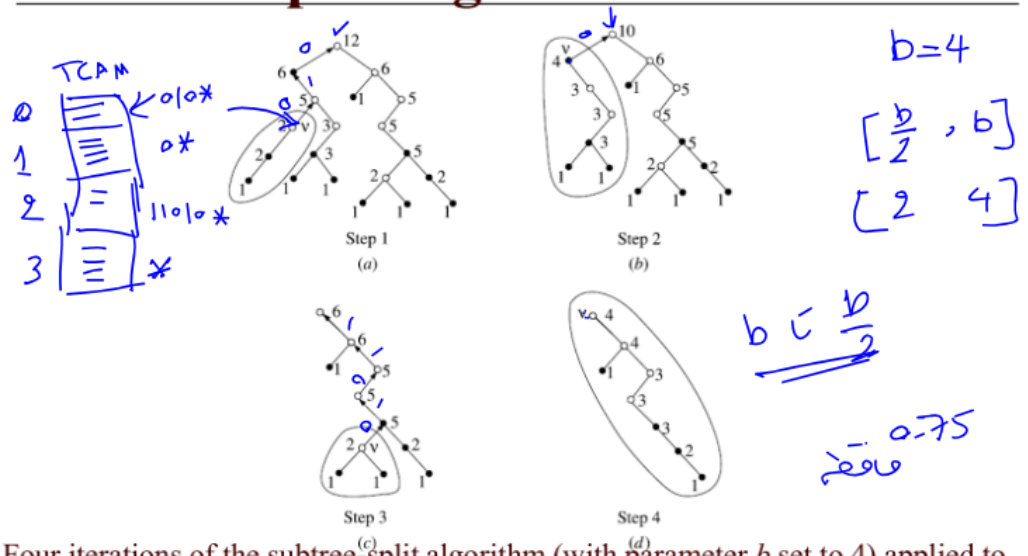
blue.aut.ac.ir/playback/presentation/2.3/486897e308544bf53250e3a499da8d22e49e3609-1633245501086

Gmail | university | programming | design | movies | books | linux | machine learning | eng | python | Google Calendar -... | google translate - ...ع | All Bookmarks

ReducingTCAM Power – CoolCAMs

Trie-Based Table Partitioning

Subtree-Split Algorithm



b=4

$[\frac{b}{2}, b]$

[2 4]

$b \leq \frac{b}{2}$

حدود 0.75

Four iterations of the subtree-split algorithm (with parameter *b* set to 4) applied to the 1-bit trie from previous figure.

-9:31 1.25x

Type here to search

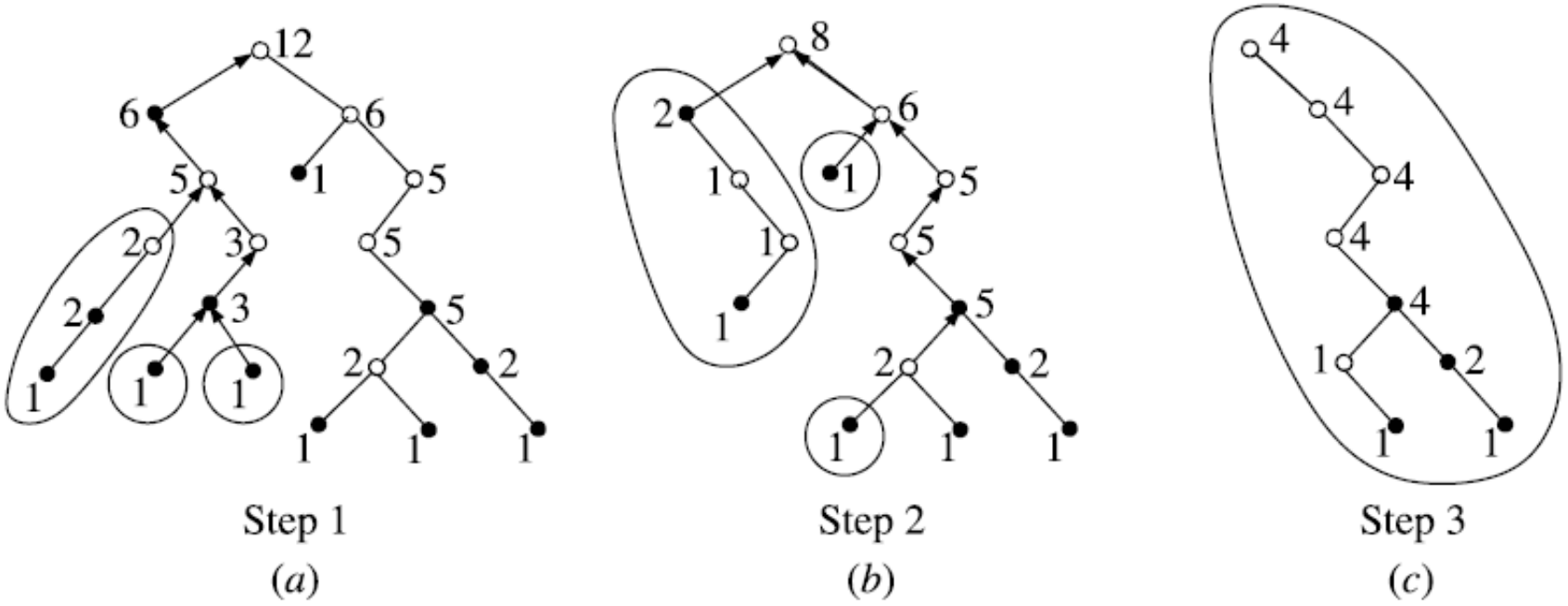
ENG 11:36 PM 11/22/2023

Subtree-Split Algorithm

Four Resulting Buckets from the Subtree-Split Algorithm

Index	Bucket Prefixes	Bucket Size	Covering Prefix
010*	0100*, 01000*	2	0*
0*	0*, 0110*, 01100*, 01101*	4	0*
11010*	110100*, 110101*	2	1101*
*	10*, 1101*, 11011*, 110111*	4	—

post-order split algorithm



Three iterations of the post-order split algorithm (with parameter *b* set to 4) applied to the 1-bit trie from previous figure.

post-order split algorithm

Three Resulting Buckets from the Post-Order Split Algorithm

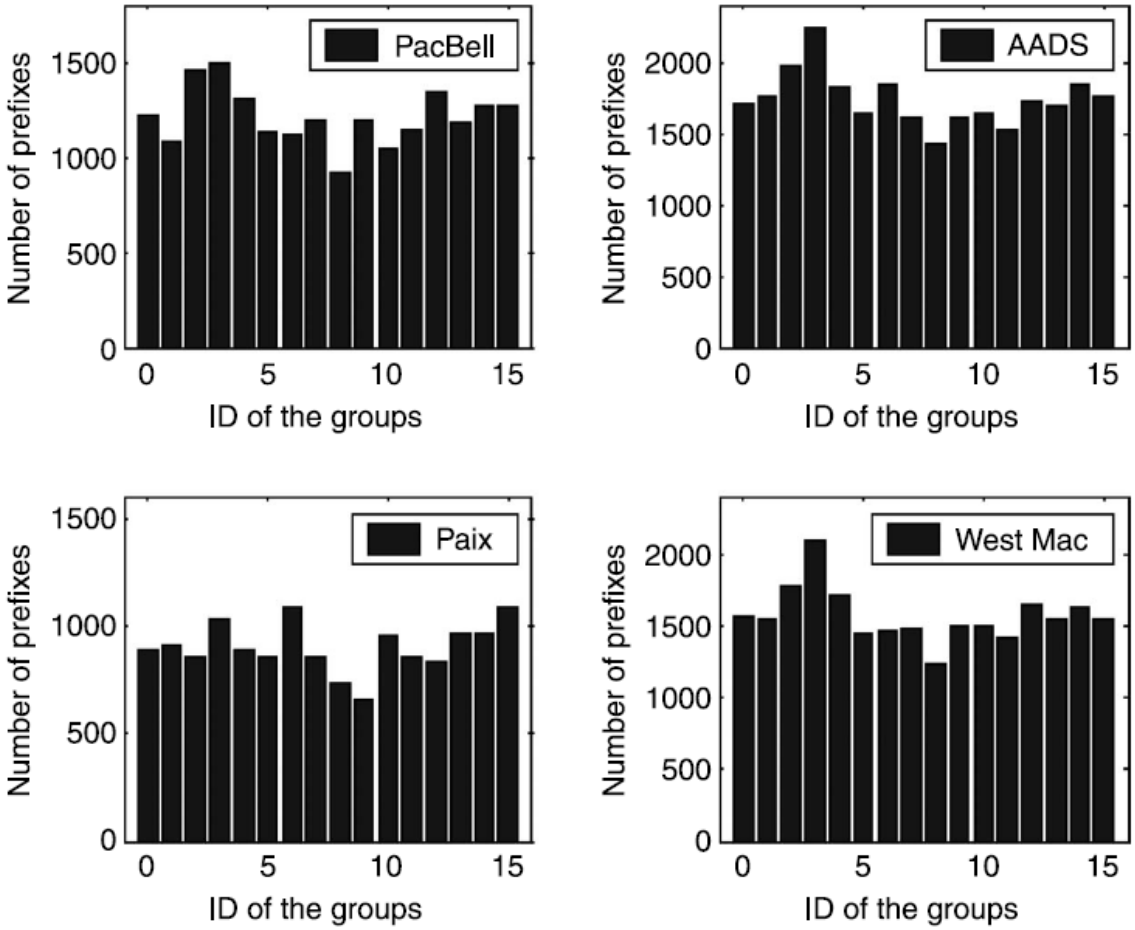
i	$Index_i$	Bucket Prefixes	Size	Covering Prefix
1	010*, 01100*, 01101*	0100*, 01000*, 01100*, 01101*	4	0*, 01100*, 01101*
2	0*, 10*, 110100*	0*, 0110*, 10*, 110100*	4	0*, 10*, 110100*
3	1*	110101*, 1101*, 11011*, 110111*	4	—

ReducingTCAM Power – CoolCAMs
Trie-Based Table Partitioning

Performance.

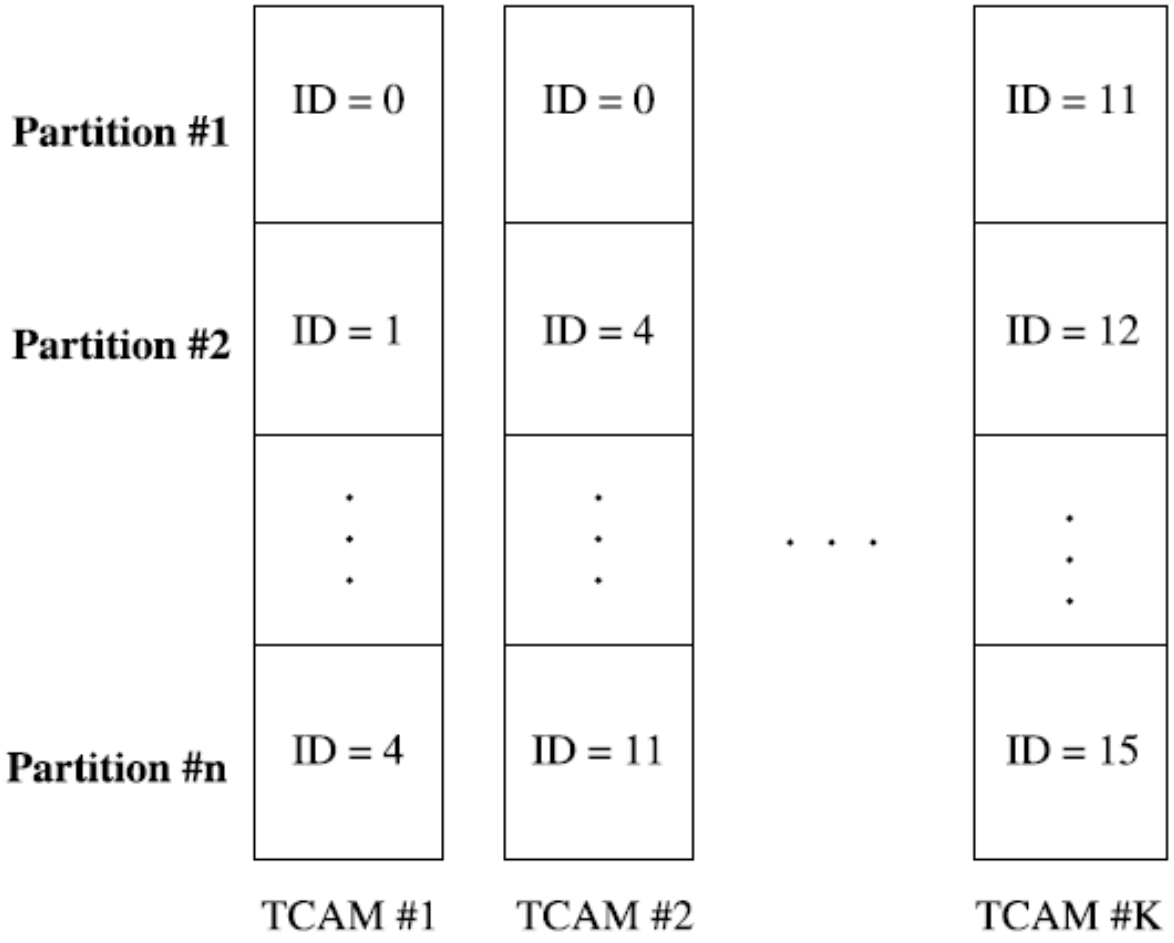
- The complexity for the post-order traversal is *O(N)*. Updating the counts of nodes all the way to root when a subtree is carved out gives a complexity of *O(NW/b)*. This is where *W* is the maximum prefix length and *O(N/b)* is the number of subtrees carved out.
- The total work for laying out the forwarding table in the TCAM buckets is *O(N)*. This makes the total complexity for subtree-split *O(N + NW/b)*. It can be proved that the total running time for post-order split is also *O(N + NW/b)*.
- The drawback of subtree-split is that the smallest and largest bucket sizes vary by as much as a factor of 2.

TCAM-Based Distributed Parallel Lookup



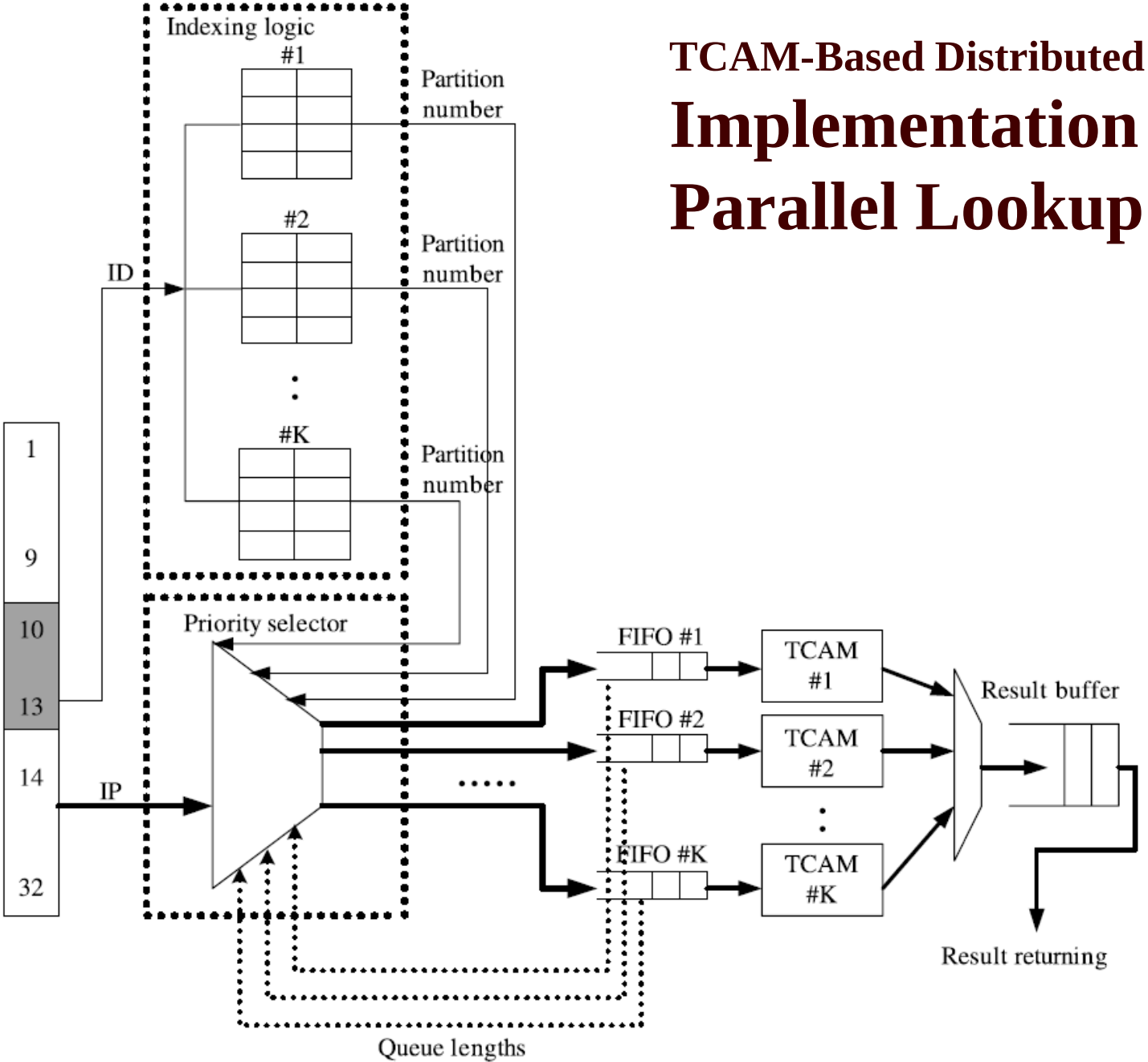
Prefixes Distribution Among ID Groups in Four Real-life Forwarding Tables

TCAM-Based Distributed Parallel Lookup Example of the TCAM Organization



TCAM-Based Distributed Parallel Lookup

Implementation of the Parallel Lookup Scheme



TCAM-Based Distributed Parallel Lookup

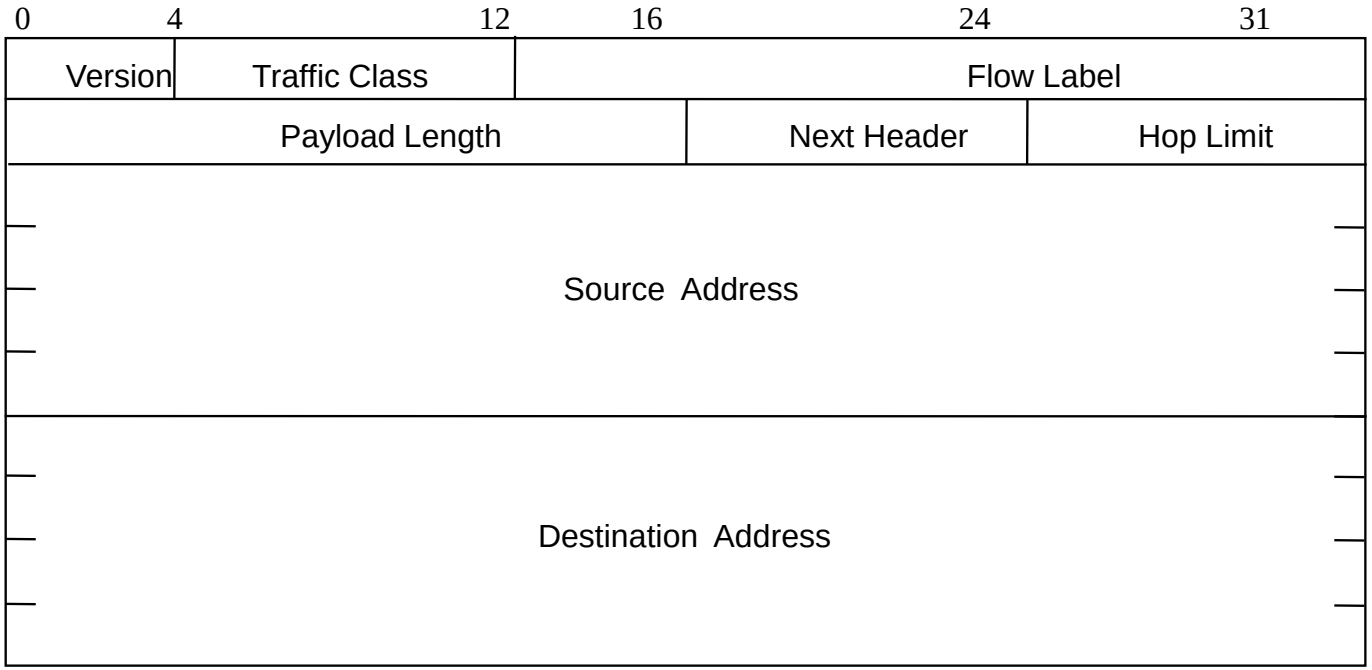
Performance.

- The lookup throughput can be significantly improved with chip level parallelism.
- The scheme can achieve a peak lookup throughput of 533 million packets per second (mpps) using four 133MHz TCAM chips and having 25 percent more TCAM entries than the original forwarding table. This performance can readily support a line rate of 160 Gbps.
- The disadvantage is that route update can be quite complicated.

Outlines

- Overview,
- Trie-based Algorithms,
- Hardware-based Schemes,
- **IPV6 Lookup**

IPv6 Lookup

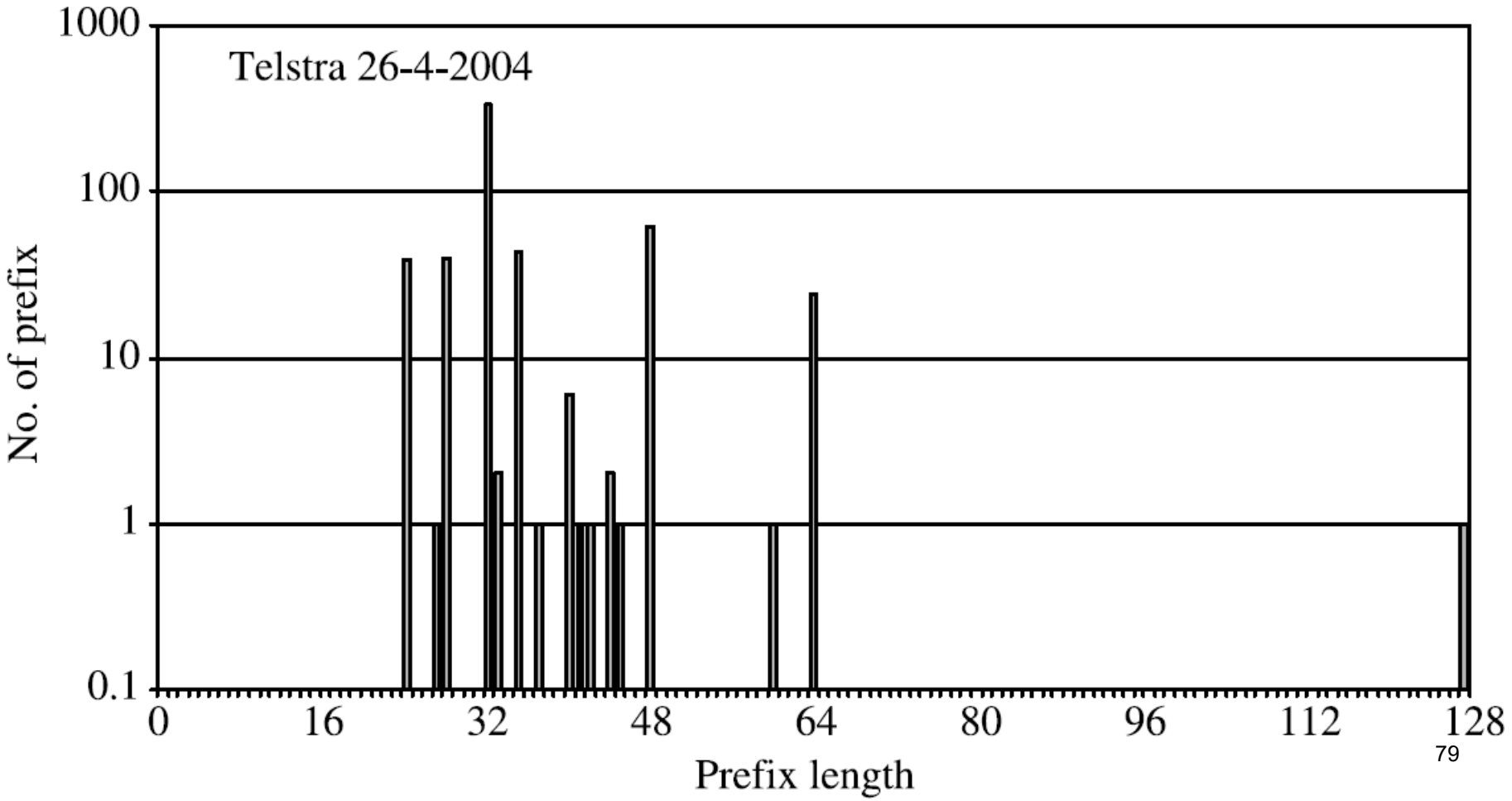


IPv6 Header Format

IPv6 Lookup

(1) It is obvious but important to note that there is no prefix with lengths between 64 bits and 128 bits (excluding 64 bits and 128 bits);
(2) The majority of the prefixes should be the ‘/48s’ and ‘/64s’ the secondary majority. Other prefixes would be distinctly fewer than the ‘/48s’ and ‘/64s’;
(3) Specifically, the number of ‘/128s’ should be tiny, which would be similar to the ratio of the ‘/32s’ in the case of IPv4.

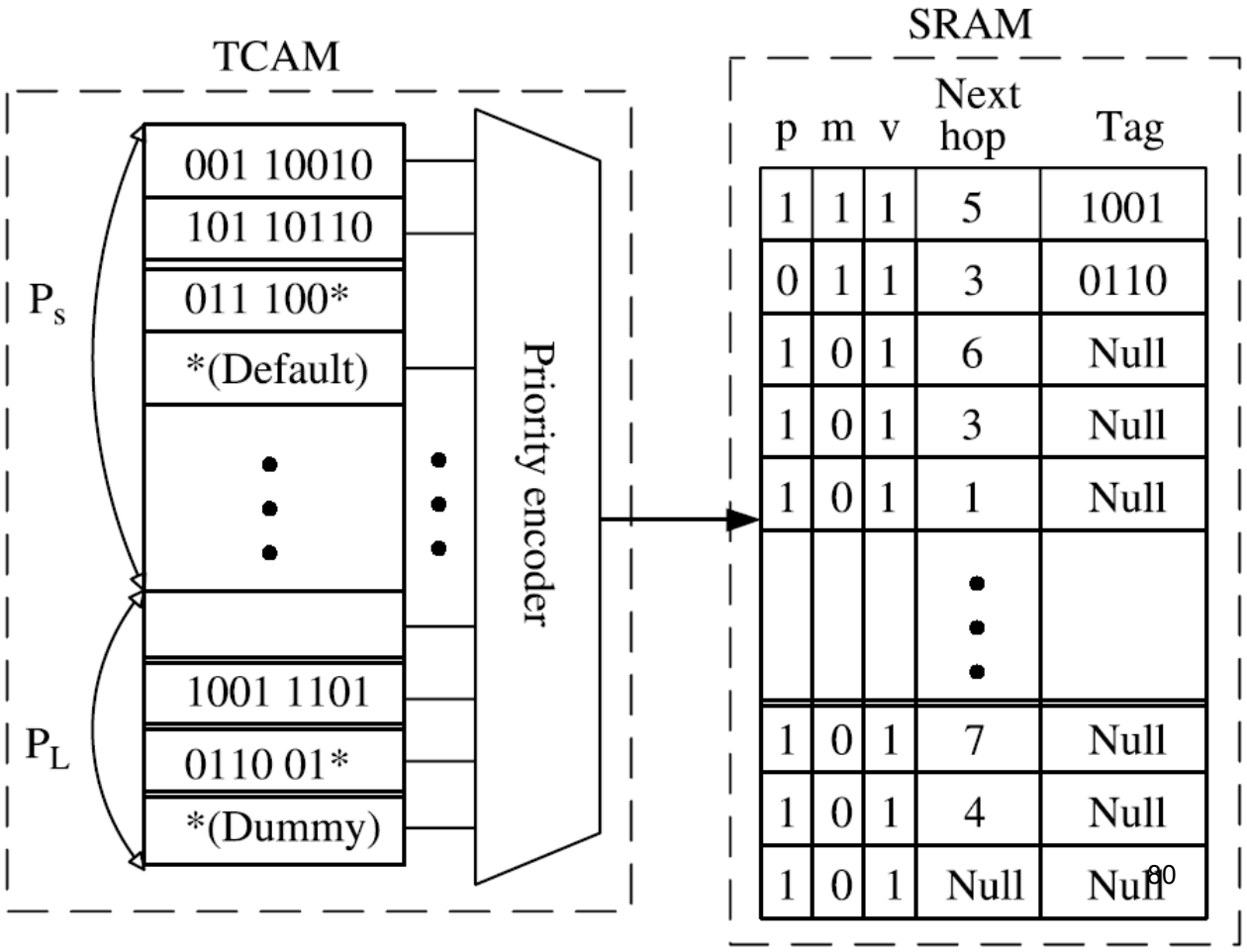
IPv6 Address Lookup
Prefix Length Distribution of an IPv6 Routing Table



IPv6 Address Lookup
Two-Level Routing Table Organization in TCAM

Prefix	Next hop
001 10010 1101	7
101 10110 01*	4
001100 10*	5
011 100*	6
10110*	3
*(Default)	1

Sample routing table
(assuming 12-bit address)

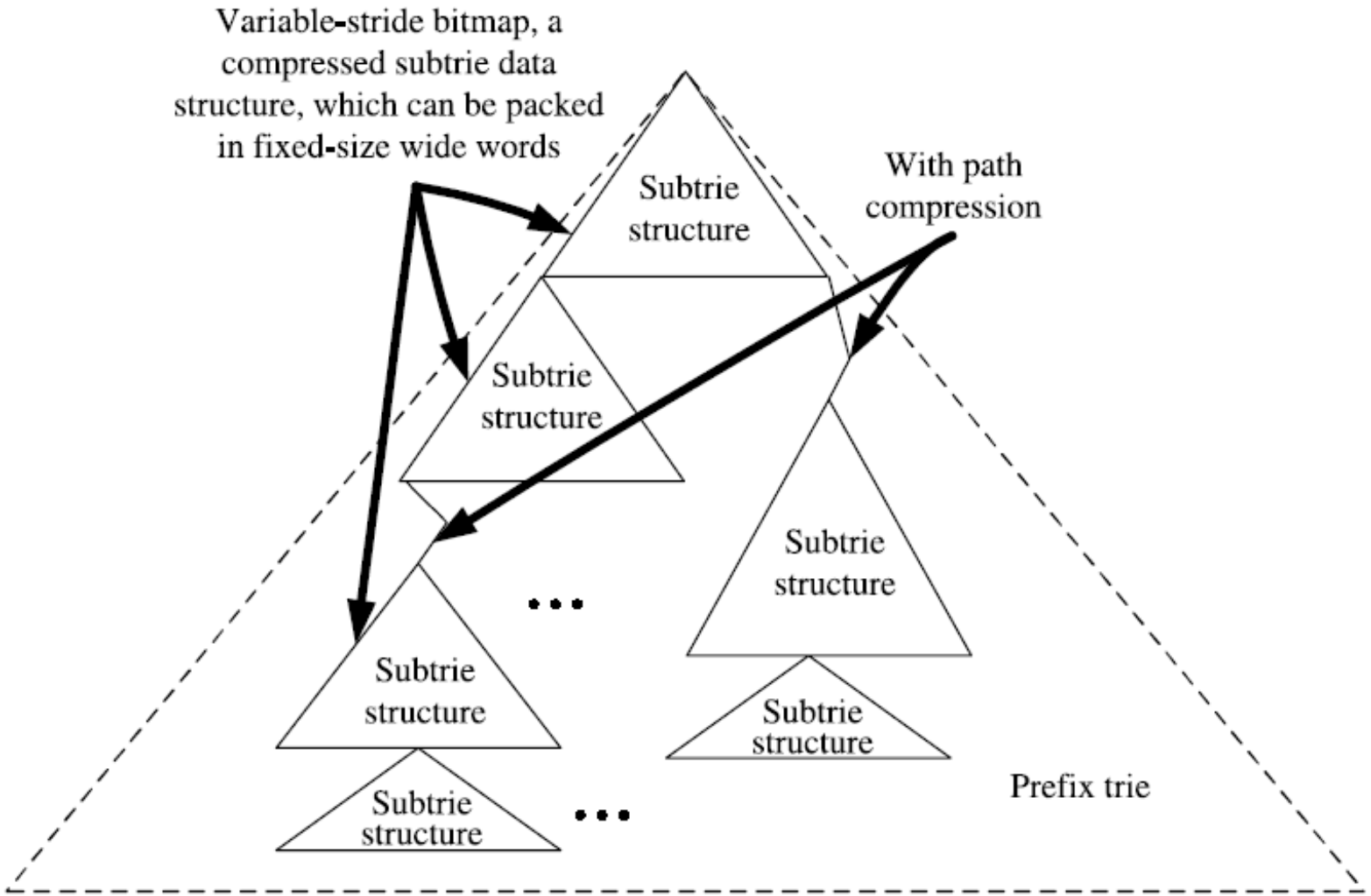


IPv6 Address Lookup
Two-Level Routing Table Organization in TCAM

Performance.

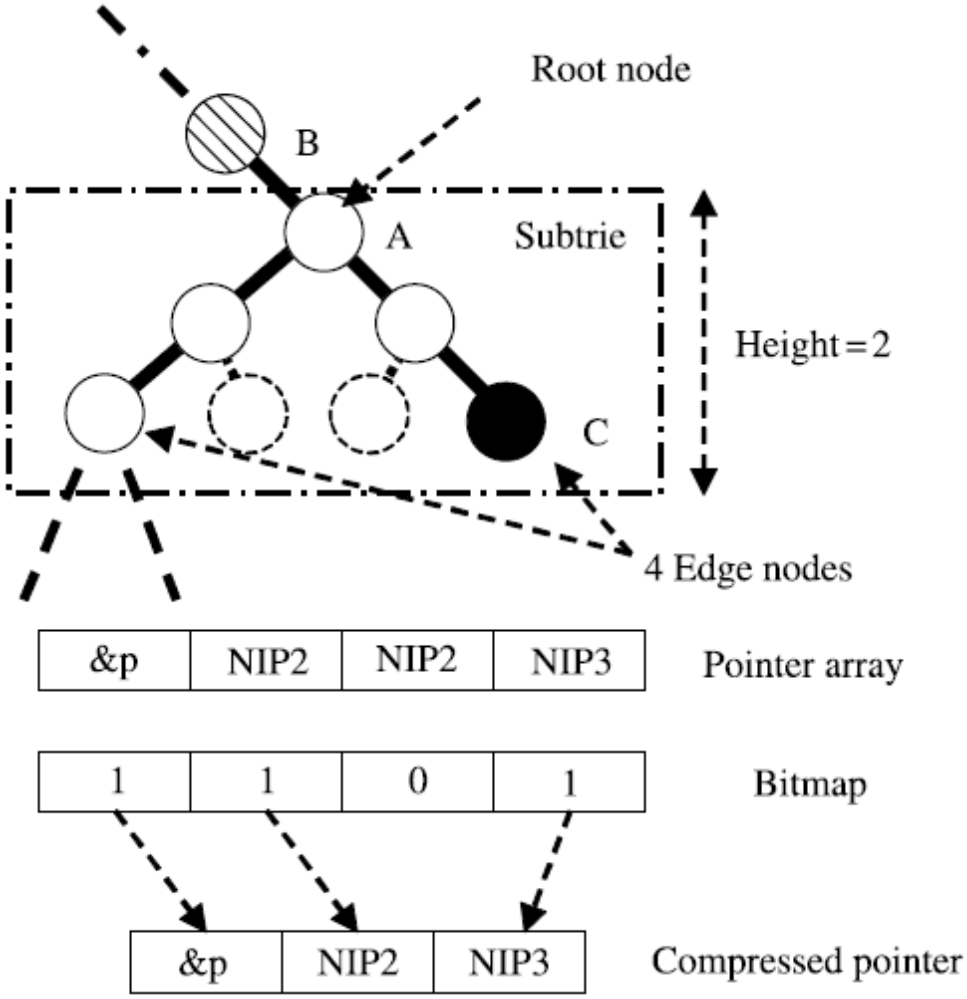
- Let the total number of prefixes be N and the fraction of prefixes with no more than 72 bits be S. Hence, the number of prefixes in group G_S is SN and the number of prefixes in G_L is $(1 - S)N$. In the basic scheme, a 144-bit word is used to store a prefix regardless of the actual length of the prefix. The total TCAM space used is $144N$ bits. For the folded approach, let the average size of a subgroup of long prefixes in G_L be β. The number of markers required is equal to $(1 - S)N/\beta$.
- The total TCAM space required is $72(SN + (1 - S)N/\beta) + 72(1 - S)N = 72N(1 + (1 - S)/\beta)$ bits.
- Let R_S be the ratio of the space used by the folded scheme over the space used by the basic scheme. We have $R_S = (1 + (1 - S)/\beta)/2$.

IPv6 Lookup via
Variable-Stride Path and Bitmap Compression



IPv6 Lookup via Variable-Stride Path and Bitmap Compression

Pointer Arrary (PA) and corresponding compressed PA

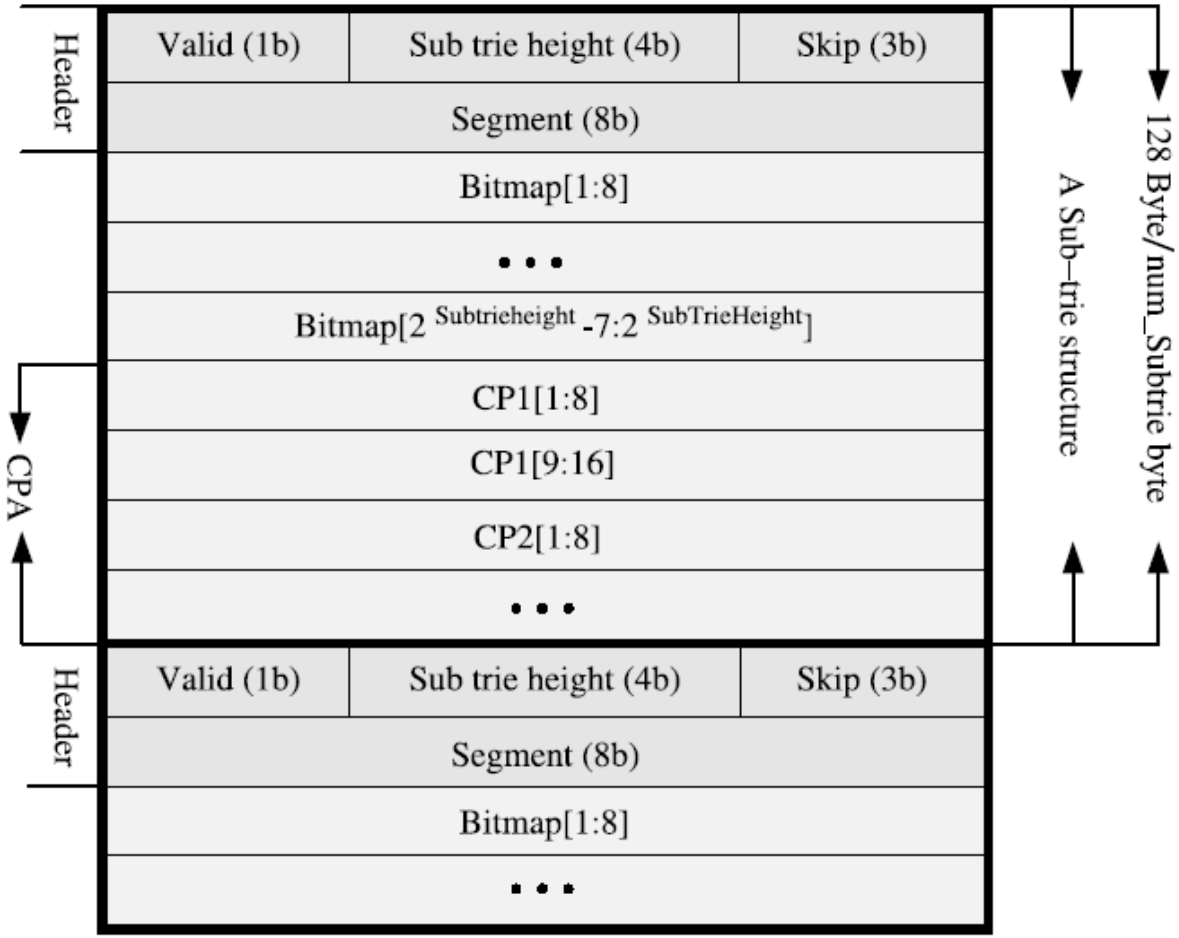


Route	Next hop
B	NIP2
C	NIP3

Route Table

IPv6 Lookup via Variable-Stride Path and Bitmap Compression

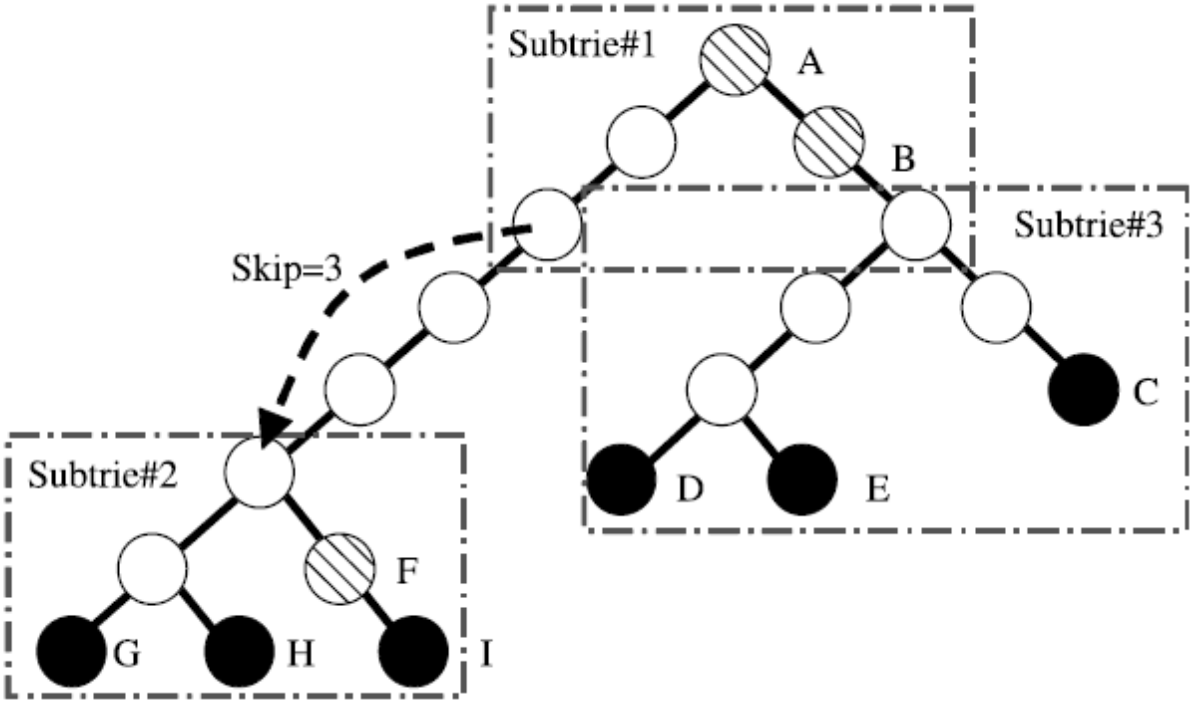
Data structure of a word frame



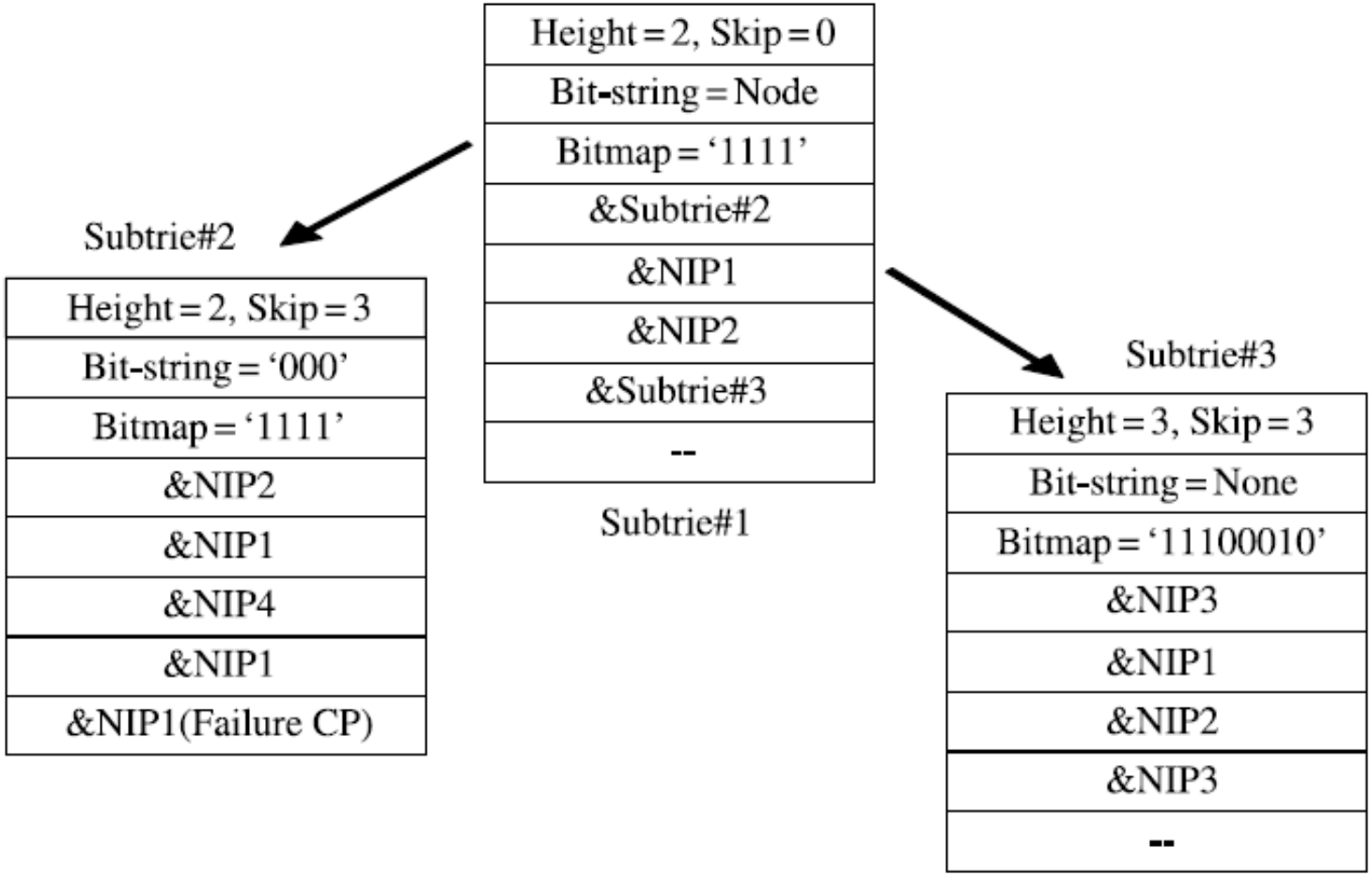
Example of variable-strike trie with path and bitmap compression trie. Forwarding table and corresponding binary trie

Route	Prefix	Next hop
A	*	NIP1
B	1*	NIP2
C	1111*	NIP3
D	11000*	NIP3
E	11001*	NIP1
F	000001*	NIP4
G	0000000	NIP2
H	0000001	NIP1
I	0000011	NIP1

Route table

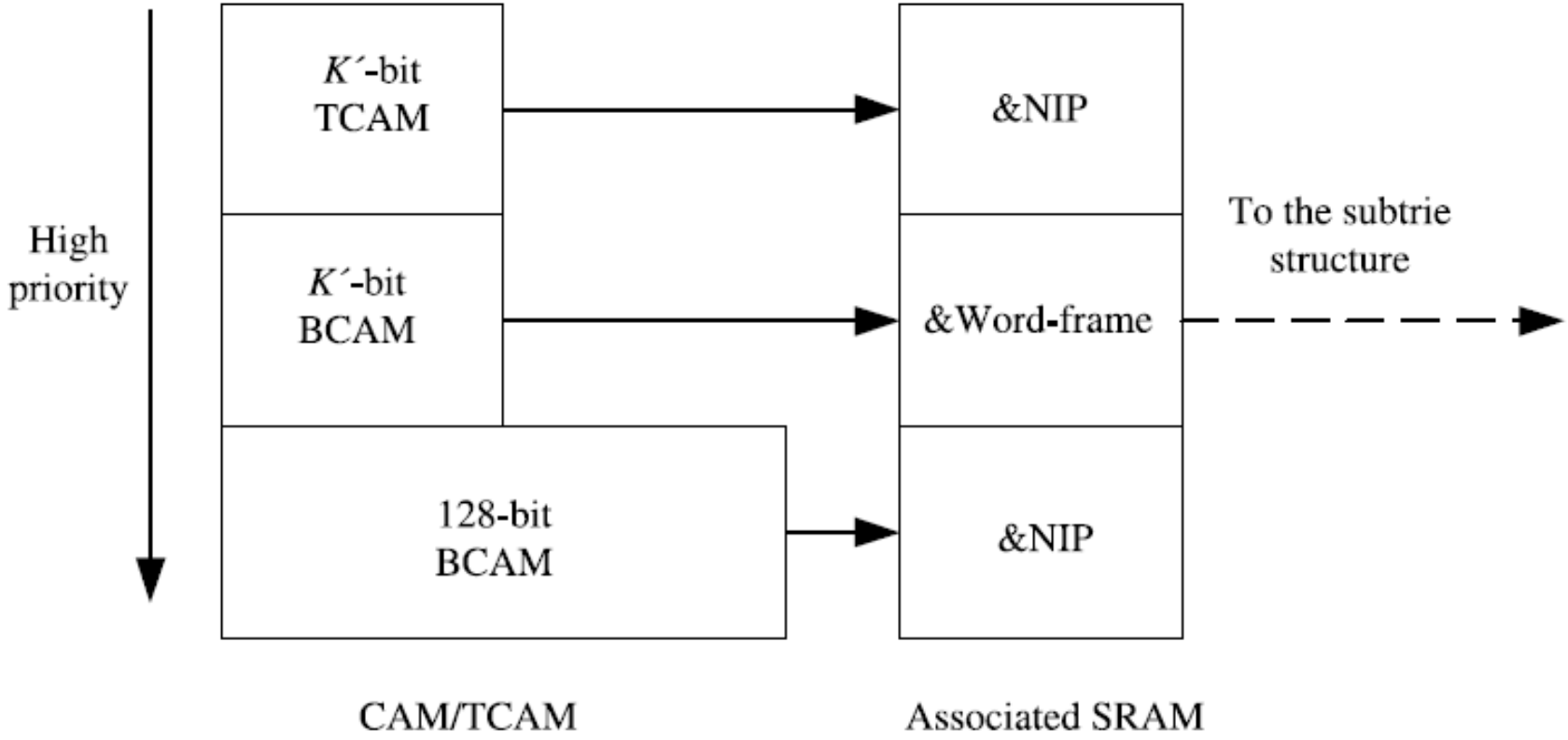


Example of variable-strike trie with path and bitmap compression trie. Corresponding data structure



IPv6 Lookup via Variable-Stride Path and Bitmap Compression

CAM organization for lookup



IPv6 Lookup via Variable-Stride Path and Bitmap Compression

Performance.

- The experimental results showthat for an IPv6 forwarding table containing over 130k prefixes, generated by an IPv6 generator, the scheme can perform 22 million lookups per second even in the worst case with only 440 kbytes of SRAM and no more than 3 kbytes of TCAM. This means that it can support 10 Gbps route lookup for back-to-back 60-byte packets using on-chip memories.

Conclusion

Complexity Comparison of Different Route Lookup Schemes

Case	Scheme	Worst-case Lookup	Storage	Update
I	1-bit trie	$O(W)$	$O(NW)$	$O(W)$
	PC-trie	$O(W)$	$O(N)$	$O(W)$
II	k-bit trie	$O(W/k)$	$O(2^k NW/k)$	$O(W/k + 2^k)$
	LC-trie	$O(W/k)$	$O(2^k NW/k)$	$O(W/k + 2^k)$
	Lulea	$O(W/k)$	$O(2^k NW/k)$	$O(W/k + 2^k)$
III	Binary search on prefix lengths	$O(\log_2 W)$	$O(N \log_2 W)$	$O(\log_2 W)$
IV	k-way range search	$O(\log_k N)$	$O(N)$	$O(N)$
V	TCAM	$O(1)$	$O(N)$	—