Clustered Linked List Forest For IPv6 Lookup

Oğuzhan ERDEM[1] and Aydın CARUS[2]
[1]Department of Electrical and Electronics Engineering
[2]Department of Computer Engineering
Trakya University, TURKEY
{ogerdem,   }@trakya.edu.tr

Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work

Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work

Background

- **IP Lookup**
  - Core functions of Internet routers
    - Match the destination IP address to the prefixes in the routing table
  - Longest Prefix Matching (LPM)
    - Multiple matched prefixes possible
    - Only the longest matched prefix will be used
  - Example
    - IP address “114.110.88.212”
    - Routing table:

Routing Prefixes	Next-Hop Forwarding
114.110.*.*.	Port 0
114.110.88.*.	Port 1

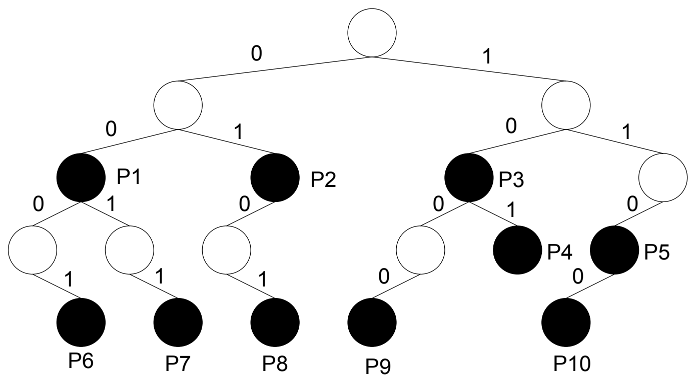
    - Matches both, but the second match is used

Tree based IP Lookup

- **Binary Trie (BT)**
  - Simple data structure
  - Easy update process
  - Large number of memory accesses

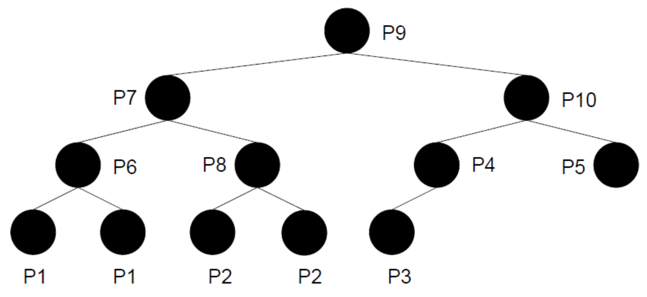
- **Binary Search Tree (BST)**
  - Each node stores prefix values
  - Better memory performance
  - Complex pre-processing
    - Prefix expansion
    - Sorting requirement
  - Hard incremental updates

Prefix	Next Hop
00*	P1
01*	P2
10*	P3
101*	P4
110*	P5
0001*	P6
0011*	P7
0101*	P8
1000*	P9
1100*	P10



Binary Trie

Next Hop	Prefix
0000	P1
0010	P1
0100	P2
0110	P2
1001	P3
1010	P4
1101	P5
0001	P6
0011	P7
0101	P8
1000	P9
1100	P10



Binary Search Tree

Performance Metrics

- **Throughput**
  - Support for higher link data rates
    - 100 Gbps network interfaces and links
- **Memory usage**
  - Rapid growth of routing table size
    - Estimated as ~500K entries in 2015
  - Transition from IPv4 (32-bit) to IPv6 (128-bit)
    - IPv4 address exhaustion
  - State-of-the-art designs cannot support large routing tables due to the available on-chip memory and the number of I/O pins of FPGAs
- **Update Capability**
  - Types of updates – Insert, delete and modify
  - How fast an update operation can be performed

Agenda

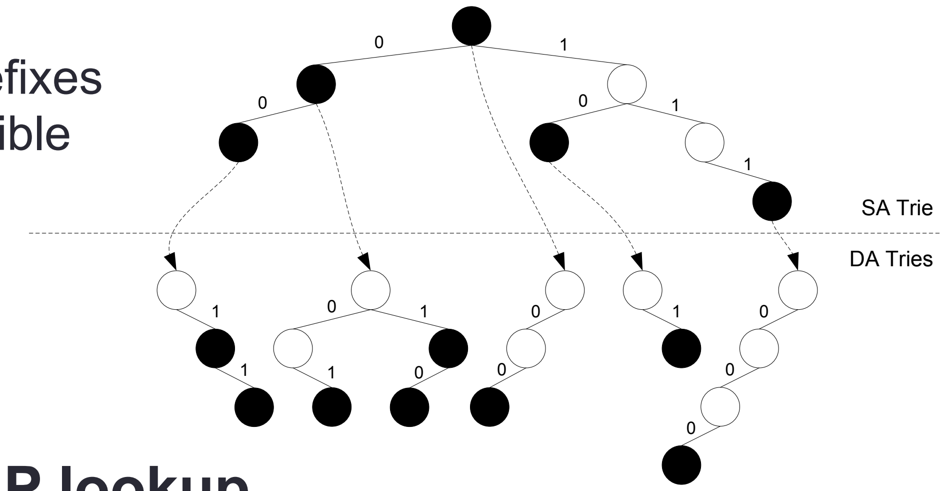
- ➢Background
- ➢Prior Work
- ➢Our Solutions
- ➢Implementation Results
- ➢Conclusions & Future Work

Hardware based solutions

- **Linear pattern search in TCAM**
  - One lookup per cycle
  - Expensive, power-hungry, low density, large access time, need for a priority encoder
- **Hash based solutions**
  - Simple and fast
  - Large number of different hash tables and functions for each prefix lengths, hash collisions.
- **Binary bit traversal in pipelined tries**
  - Good throughput performance and support quick prefix updates
  - Inefficient memory usage
- **Binary value search in pipelined trees**
  - Good memory performance
  - Complex pre-processing and updates

Hierarchical Search Structures

- **Hierarchical trie for packet classification**
  - Hierarchically connected binary tries
  - One big SA trie from SA prefixes
  - Many small DA tries using DA prefixes
  - Hardware implementation unfeasible
    - Backtracking problem



- **Hierarchical structures for IP lookup**
  - Ony DA is used
  - Split prefixes into two part
  - Hierarchically connected data structures
  - Substantial memory saving
    - Bit strings shared by multiple prefixes are stored only once
    - No backtracking : Fixed size strings naturally pairwise disjoint

Agenda

- ➢Background
- ➢Prior Work
- ➢Our Solutions
- ➢Implementation Results
- ➢Conclusions & Future Work

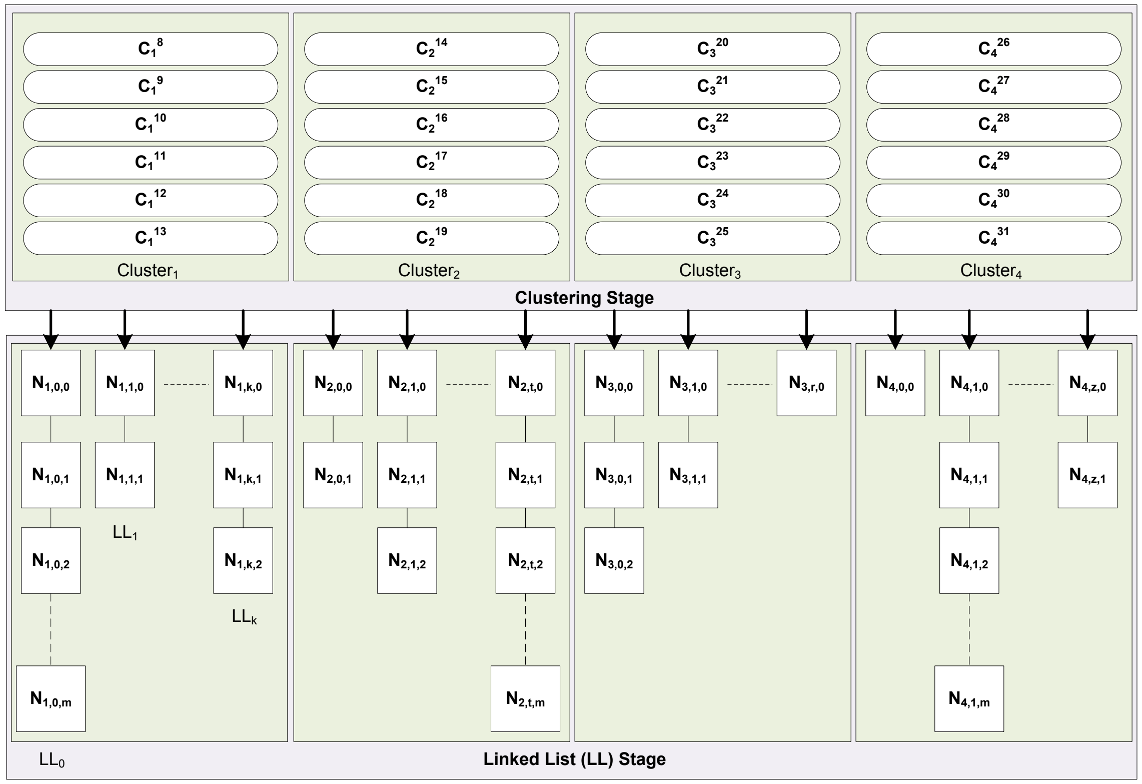
Our Solutions

- **Clustered Linked List Forest (CLLF) structure**
  - Hierarchical two stage data structure
  - Reduces the memory requirement
  - Simplifies the table updates

- **Linear pipelined SRAM-based architecture on FPGAs**
  - Supports up to 712K IPv6 prefixes
  - Sustained throughput of 432 Million lookups per second (138 Gbps for the minimum packet size of 40 Bytes)

Clustered Linked List Forest (CLLF)

- **Hierarchical two stage data structure**
  - A prefix P can be expressed as the concatenation of two substrings P_x and P_y such that $P = P_x P_y$.
  - **Stage 1 :** Clustering stage (P_x) **Stage 2 :** Linked List (LL) stage (P_y)



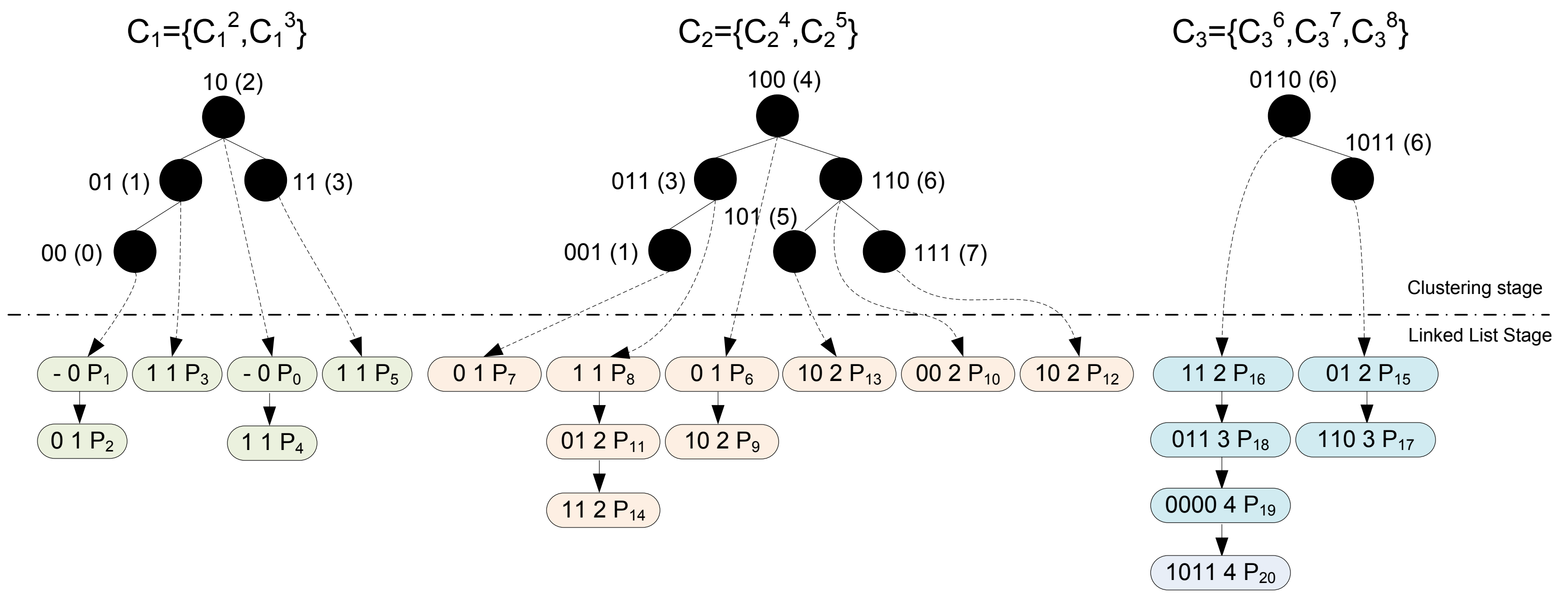
Clustering

- **Length-based**
  - *64* clusters in IPv6 the worst case
    - Each cluster is indexed starting from 1 to *n* ($\boldsymbol{C_i}$ represents i^{th} cluster)
    - Reduce the number of clusters by merging
- **Parameters**
  - *Block size (BS) :* The number of distinct lengths involved in a cluster
    - C_i^L **:** Each distinct length (L) block of prefixes in a C_i
    - Ex: If a cluster C_i includes the prefixes of lengths from *8 to 11,* then $BS(C_i)$=4 and C_i^9 represents a bunch of prefixes with length *9.*
  - *Initial prefix stride (IPS) :* The length of substring P_x ($|P_x|$) *in* $P = P_x\ P_y$
    - Ex*:* If $IPS\ (C_i)= 4$, then a prefix $P = 110010111^*$ in C_i can be represented using two substrings $P_x= 1100$ and $P_y= 10111^*$

Data Structure

- **Stage 1: Clustering stage**
  - Each group has a *Binary Search Tree (BST)* using P_x of prefixes.
- **Stage 2: Linked List (LL) stage**
  - Each node of BST connects to a *linked list data structure.*
  - Each node in a LL contains different $\mathbf{P_y}$ substring with its length.
  - $BS(C_1)$=2, $BS(C_2)$=2, $BS(C_3)$=3, $IPS(C_1)$=2, $IPS(C_2)$=3, $IPS(C_3)$=4.

	Prefix	Next Hop
P_0	10*	1
P_1	00*	2
P_2	000*	2
P_3	011*	8
P_4	101*	6
P_5	111*	5
P_6	1000*	8
P_7	0010*	6
P_8	0111*	7
P_9	10010*	10
P_{10}	11000*	9
P_{11}	01101*	8
P_{12}	11110*	10
P_{13}	10110*	3
P_{14}	01111*	1
P_{15}	101101*	5
P_{16}	011011*	1
P_{17}	1011110*	3
P_{18}	0110011*	7
P_{19}	01100000*	9
P_{20}	01101011*	4



IP Lookup

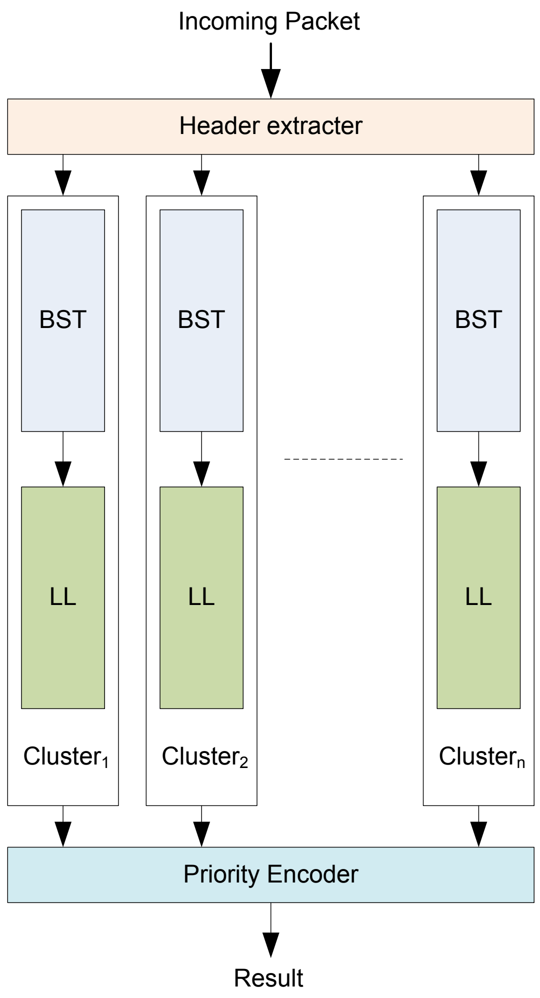
- **Split IP address into two: *IP_x* and *IP_y***
  - Based on the *IPS* value of corresponding cluster
  - Search IP_x in Stage 1 (BST) and IP_y in Stage 2 (Linear search).
- **Parallel searches in clusters**
  - If a match is found in BST, proceed to the connected LL.
  - At each LL node, an IP_y sub-address is compared with P_y sub-prefix.
  - Update search result if longer match than the previous match.
  - Search terminates when all the nodes in a LL is traversed
  - Outputs from all clusters are given to the priority encoder that chooses the longest prefix.

Optimization

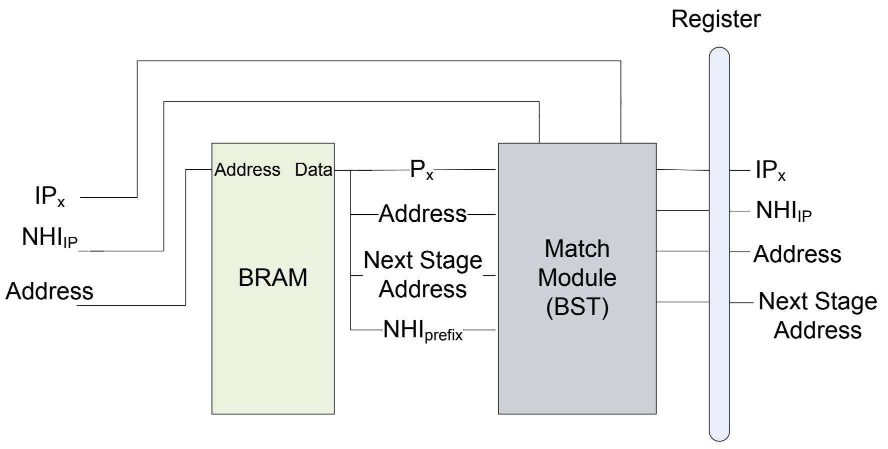
- **Node size optimizations**
  - BST
    - The nodes of BST can be enhanced to store P_y substrings
    - P_{tree}: A limit for the number of P_y substrings per tree node
  - LL
    - LL nodes can be enhanced to store **more than one** P_y substrings
    - P_{LL}: Upper bound for the number of P_y prefixes per LL node
- **Clustering optimizations**
  - *BS*
    - Defines the number and size of clusters
    - Larges *BS* results in less number of pipelines in hardware implementation
  - *IPS*
    - Determines the number of nodes in BST and LL.
    - Smaller *IPS* reduces the number of BST nodes but increase the number of LL nodes and implicitly the delay.

Architecture

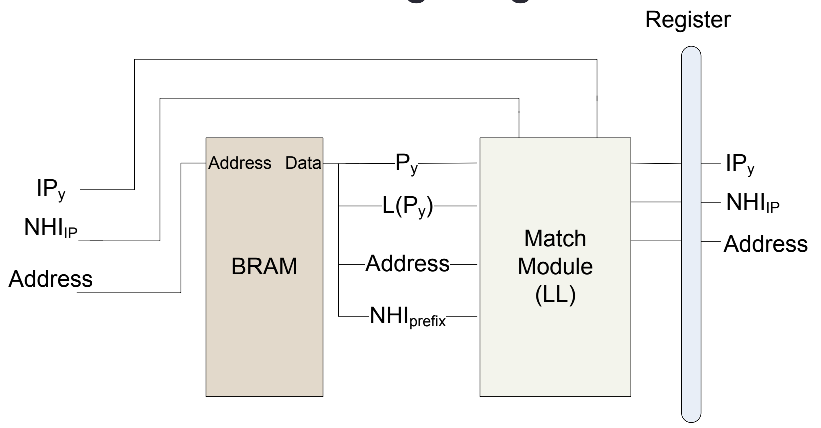
- Overall



- 1 stage



Clustering stage



LL stage

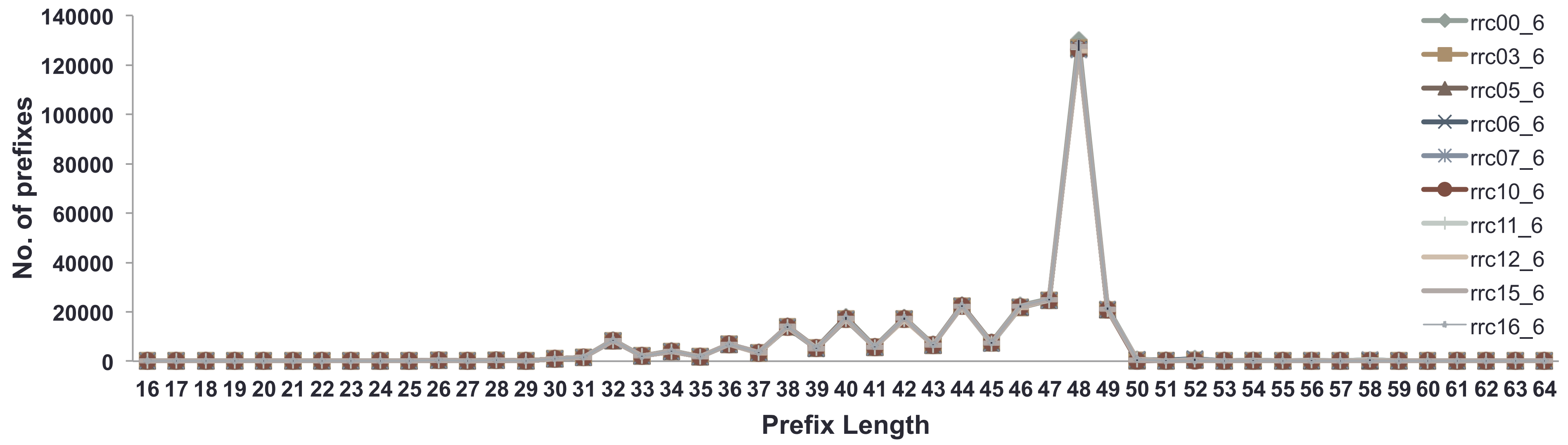
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work

Results

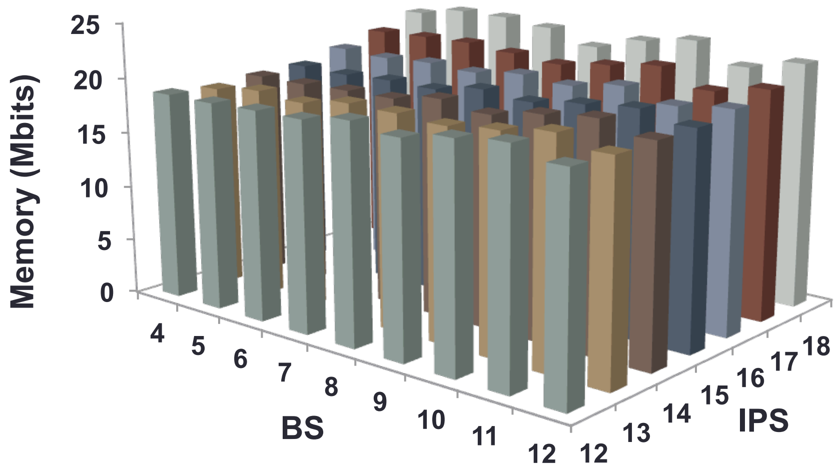
- **Experimental setup**
  - Ten IPv4 core routing tables from Project - RIS on 06/03/2010.
  - Ten synthetic IPv6 routing tables generated using IPv4 core RTs

IPv4	rrc00	rrc03	rrc05	rrc06	rrc07	rrc10	rrc11	rrc12	rrc15	rrc16
IPv6	rrc00_6	rrc03_6	rrc05_6	rrc06_6	rrc07_6	rrc10_6	rrc11_6	rrc12_6	rrc15_6	rrc16_6
# prefixes	332118	321617	322997	321577	322557	319952	323668	320016	323987	328296

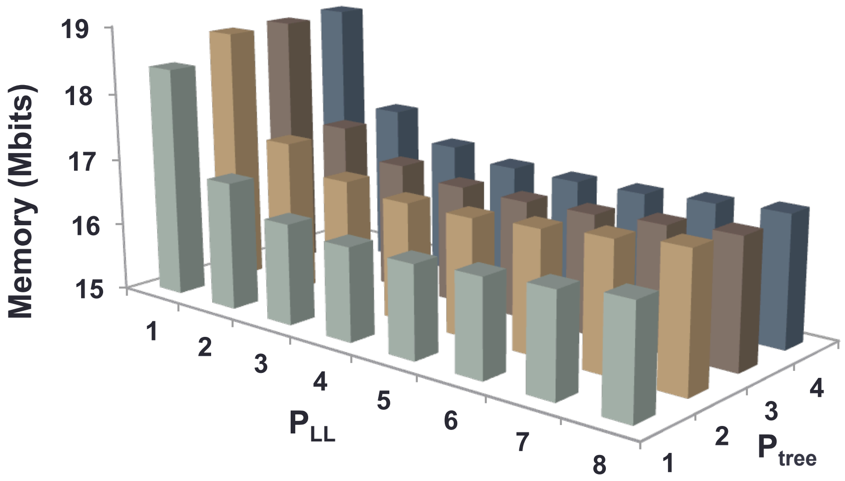


Memory Requirement

- For various *BS* and *IPS* (P_{LL}=P_{tree}=1)



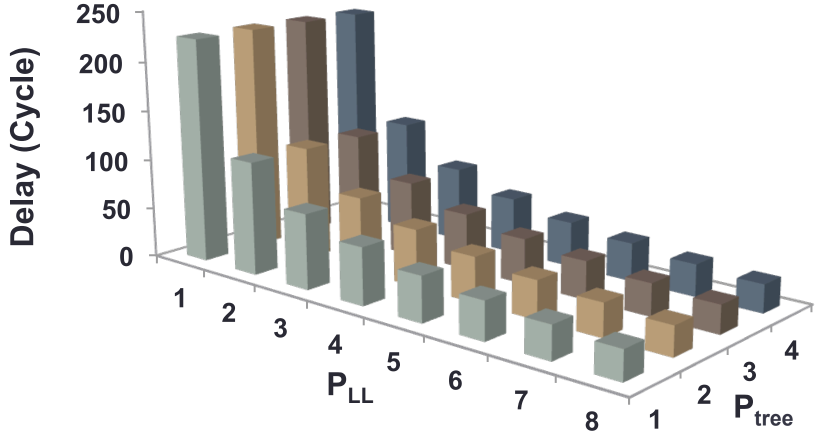
- For various P_{tree} and P_{LL} (*BS*=4, *IPS*=13)



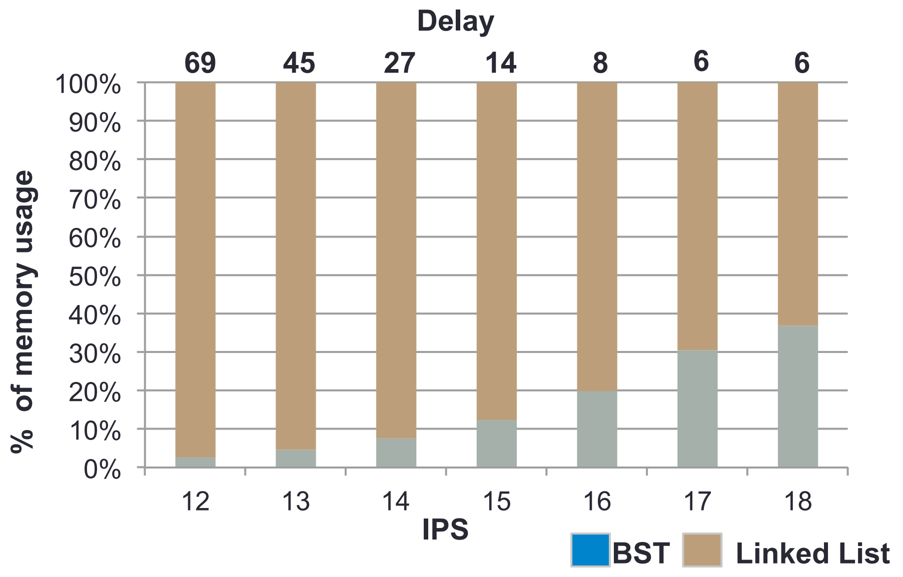
	rrc00_6	rrc03_6	rrc05_6	rrc06_6	rrc07_6	rrc10_6	rrc11_6	rrc12_6	rrc15_6	rrc16_6
BT	341	330	332	331	332	329	347	329	334	337
BTPC	72	69	70	69	70	69	72	70	70	71
DBPC	45	45	45	45	46	45	48	45	45	46
BST	35.5	34.4	34.6	34.4	34.5	34.3	34.6	34.3	34.7	35.2
CTF	27.9	27.3	27.0	29.2	27.2	27.1	27.1	26.9	28.0	27.2
CLLF	***16.4***	***15.8***	***15.9***	***15.9***	***16.0***	***15.7***	***16.0***	***15.7***	***15.9***	***16.1***

Delay

- The delay for various P_{tree} and P_{LL} (*BS*=4, *IPS*=13)



- The % of memory by BST and LL for various *IPS* (*BS*=4, P_{LL}=5, P_{tree}=1)



	rrc00_6	rrc03_6	rrc05_6	rrc06_6	rrc07_6	rrc10_6	rrc11_6	rrc12_6	rrc15_6	rrc16_6
CLLF	45	44	45	44	45	45	44	45	46	45

- Depth (cycles) (*BS*=4, *IPS*=13, P_{LL}=5, P_{tree}=1)

Performance Comparison

- **FPGA implementation**
  - Implemented in Verilog using Xilinx ISE 12.4 and target device Xilinx XC6VSX475T
  - 216 MHz (432 million packets per second with dual-ported BRAMs)
  - Clock period is 4.63 ns, 138 Gbps for the minimum packet size of 40 bytes
- **Memory efficiency (in Bits/prefix)**
- **Throughput efficiency (in Gbps/Bits)**

Architecture	# prefix	Mem. Eff. Bits/prefix	Throughput Gbps	Throughput eff. Gbps/B
CLLF	***324***	***51.8***	***138***	***2.66***
CTF	324	88.1	135	1.53
BST	324	124	125	1.01
DBPC	324	146	120	0.82
2-3 tree	324	167	120	0.72
Flashtrie	310	171	64	0.37
TreeBitmap	310	405	6	0.02

Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Sesults
- Conclusions & Future Work

Conclusions and Future Work

- **Clustered Linked List Forest**
  - Significant memory saving
  - Performance improvements with optimizations
- **Linear pipelined SRAM-based architecture on FPGAs**
  - Supports up to 712K IPv6 prefixes
  - Sustained throughput of 138 Gbps
- **Future work**
  - Extend the algorithm to virtual routers to improve their memory efficiency

THANK YOU !