CP-Trie: Cumulative PopCount based Trie for IPv6 Routing Table Lookup in Software and ASIC

MD Iftakharul Islam, Javed I Khan

Department of Computer Science
Kent State University
Kent, OH, USA.

Outline

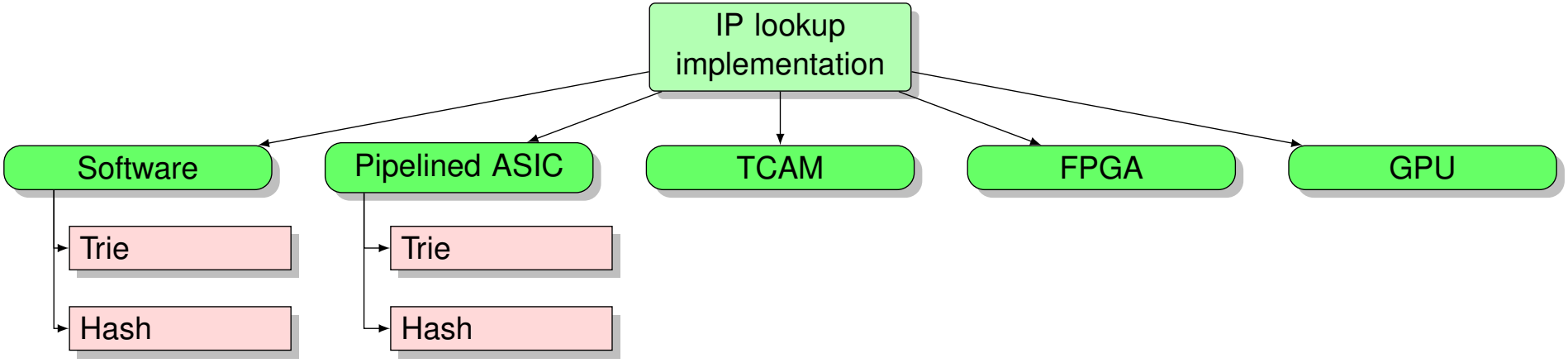
Routing table lookup

- Longest prefix match (LPM) to find the outgoing port.

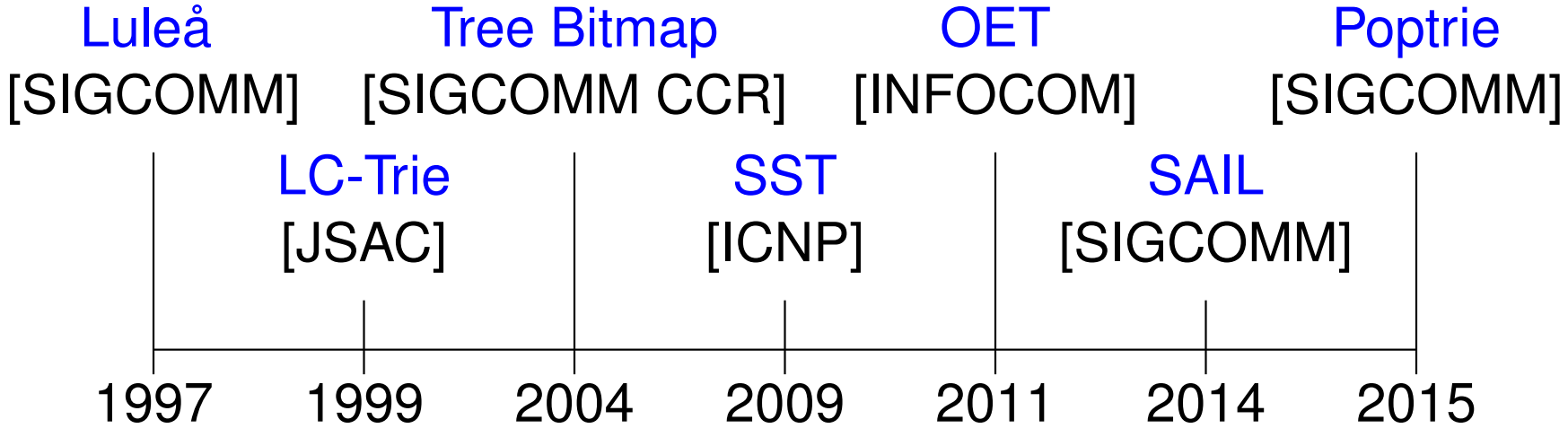
Route	Prefix	Next-hop
r1	$200a:410:8000::/40$	2
r2	$200a:410:8080::/44$	3
r3	$200a:410:8000:702::/64$	1
r4	$200a:410:8000:702::0df/128$	2

- $200a:410:8088:500::300 \Longrightarrow \{r1, r2\} \Longrightarrow 3$
- Challenges:
  - Very large routing table.
    - IPv6: $100k+$ entries.
    - Growing very rapidly
  - Extremely high throughput.
    - 4.8 Tb/s Brodcom Jericho2 chip can forward $2,000$ Mpps.
  - Longer prefix.
    - An IPv6 prefix can be 128 bits while IPv4 prefix can be 32 bits long.
    - Requires $4\times$ processing and significantly more memory compared to IPv4 lookup.

Existing solutions



- Trie based IP lookup in Software and ASIC is our main focus.

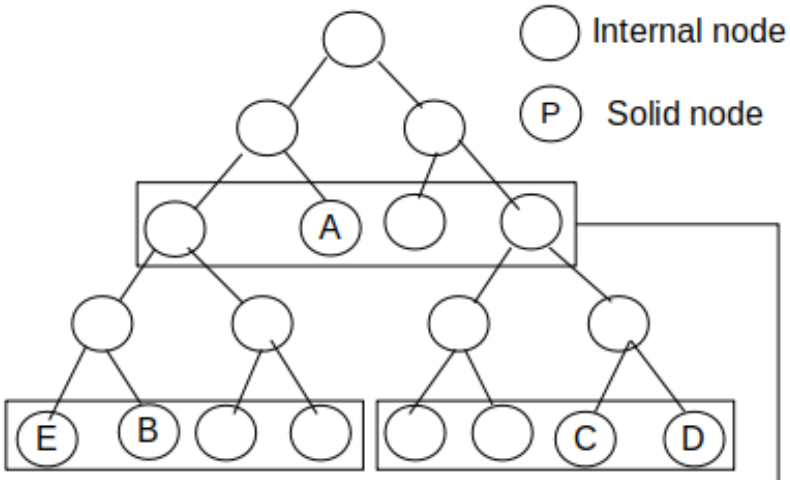


- This paper presents an extension of Poptrie named CP-Trie.

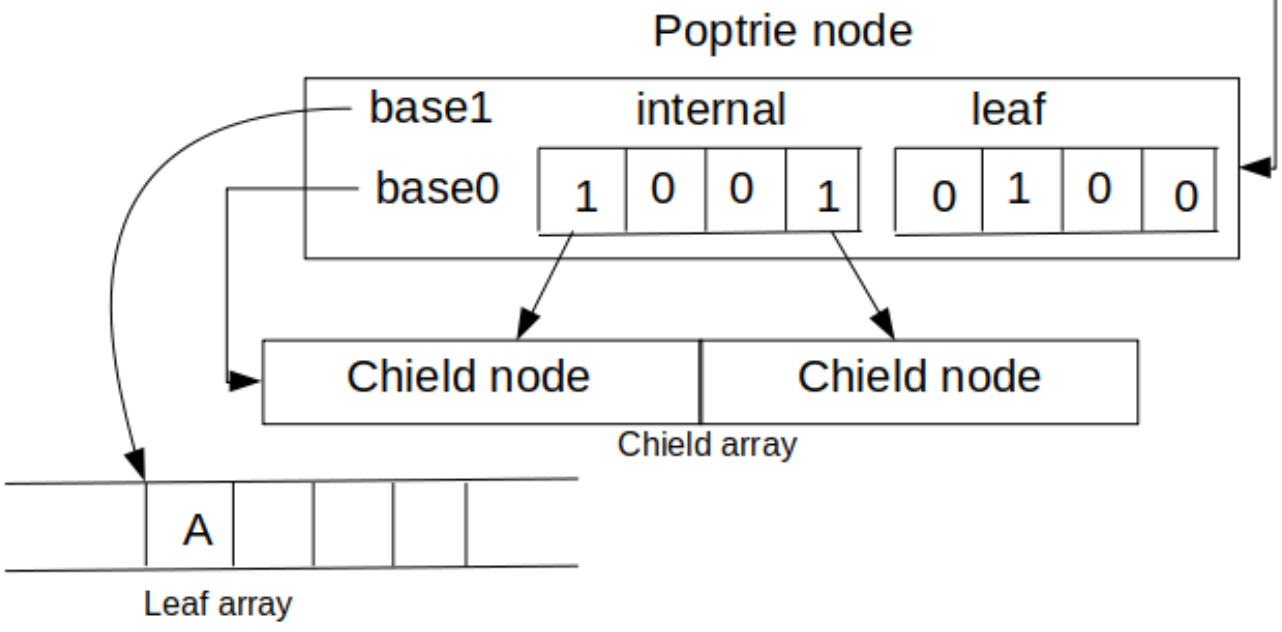
Poptrie based IP lookup

Prefix (in binary)	Next-hop
01*	A
0001*	B
1110*	C
1111*	D
0000*	E

Forwarding table



2-way trie

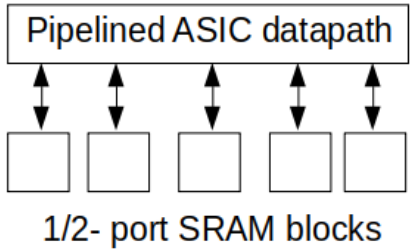


Poptrie based IP lookup

- Poptrie encodes a node with 64-bit bitmaps $\Longrightarrow$ It uses 6 bit stride ($2^6 = 64$)
  - POPCNT instruction which can process 64 bits on a 64 bit machine.
- As a result, Poptrie splits an IPv6 routing table table into 20 levels: level 16, 22, 28, 34, 40, 46, 52, 58, 64, 70, 76, 82, 88, 94, 100, 106, 112, 118, 124 and 130.
- CP-Trie stores cumulative PopCount along with bitmap with allows it to use longer stride ($8 - 16$ bit).
- CP-Trie splits an IPv6 routing table into 15 levels: level 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120 and 128.
- Fewer level have several significant benefits.

Design goals of a Trie based IP lookup

- Fewer number of steps
  - Results in faster lookup
  - Reduces power consumption in ASIC
- Fewer memory accesses (critical in pipelined ASIC).
  - In pipelined ASIC, # of memory accesses = # of SRAM blocks needed (each pipeline stage has to access SRAM in parallel.).

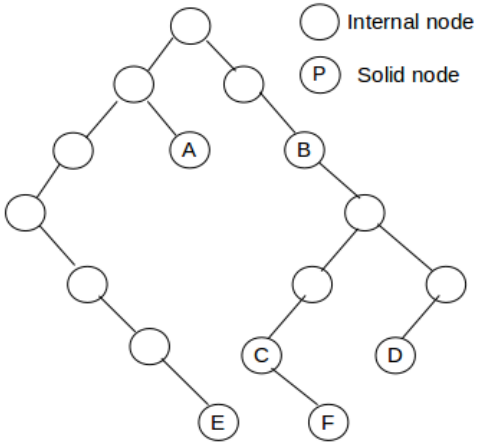


  - It also has been found that fewer large SRAM blocks are more area efficient than many smaller SRAM blocks.
  - This is why, reducing the number of memory access reduces the number of SRAM blocks needed in ASIC which results in smaller area.

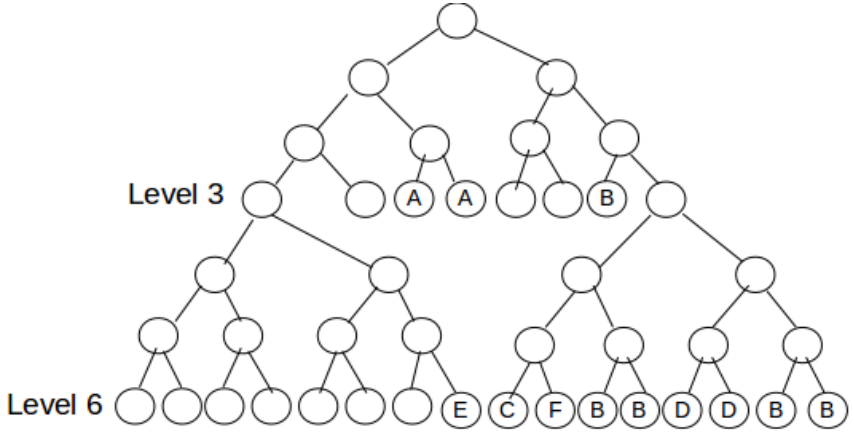
CP-Trie requires fewer steps and fewer memory accesses than Poptrie.

CP-Trie based IP lookup

Prefix (in binary)	Next-hop
01*	A
11*	B
11100*	C
11110*	D
000111*	E
111001*	F



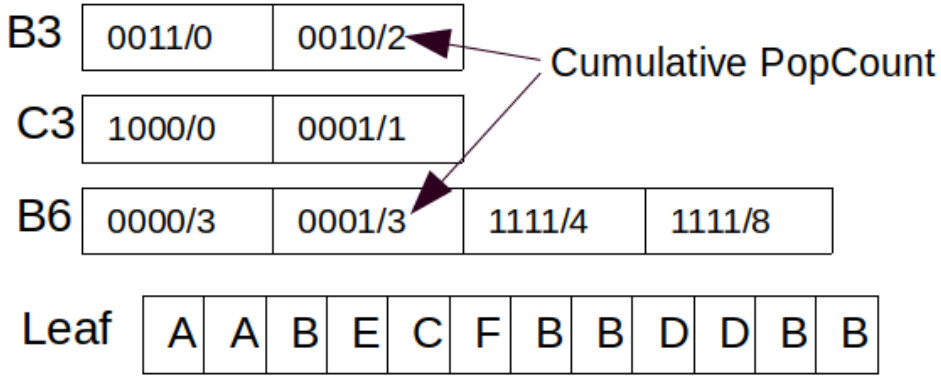
(a) routing table

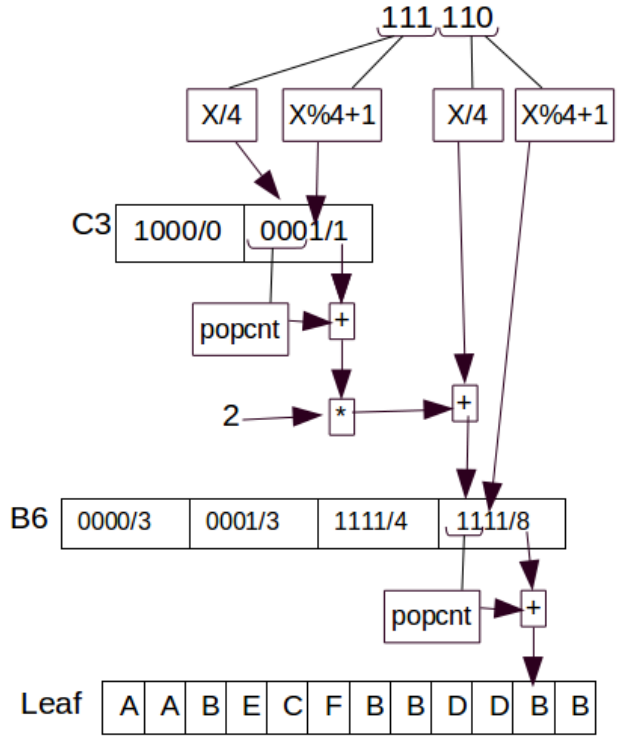


(b) leaf pushing tree

B3 = 00110010
C3 = 10000001
B6 = 0000000111111111

By grouping every 4 bits, we get:





Implementations

- Implemented both Poptrie and CP-Trie in C (Linux).
- Implemented both Poptrie and CP-Trie in Verilog RTL.
  - We use an internally developed high-level synthesis (HLS) tool to generate the Verilog RTL code. The details of the HLS tool is out of the scope of this paper.
- We generated physical chip layout using OpenROAD EDA. We also evaluate power, area and timing characteristics of ASIC using OpenROAD.
- We used 45 nm Nandgate cell library and 1 GHz clock for physical synthesis.
- We implemented the arrays using register file (Nandgate cell library does not have SRAM cell, to the best of our knowledge.)

Evaluation in ASIC

	Poptrie	CP-Trie
Clock speed	1 GHz	1 GHz
Internal Power	76.5 mW	64.6 mW
Switching Power	24.4 mW	22.2 mW
Leakage Power	1.15 mW	0.926 mW
Total Power	102.05 mW	87.726 mW
Area	0.0658 mm^2	0.0523 mm^2

- Poptrie and CP-Trie need 79 and 59 SRAM blocks respectively.
  $\Longrightarrow$ We expect CP-Trie to achieve even lower area and power consumption than Poptrie when incorporating SRAM.

Experimental results

Table: FIB dataset

Name	# of prefixes	Longest prefix length
fib0	105, 363	48
fib1	102, 126	48
fib2	79, 431	64
fib3	103, 067	128
fib4	105, 957	128
fib5	102, 739	64
fib6	104, 235	48
fib7	100, 899	64
fib8	102, 731	128

- We obtained the data set from RouteView project. The snapshot was taken on January 17, 2021 at 3 PM EST.

Memory consumption

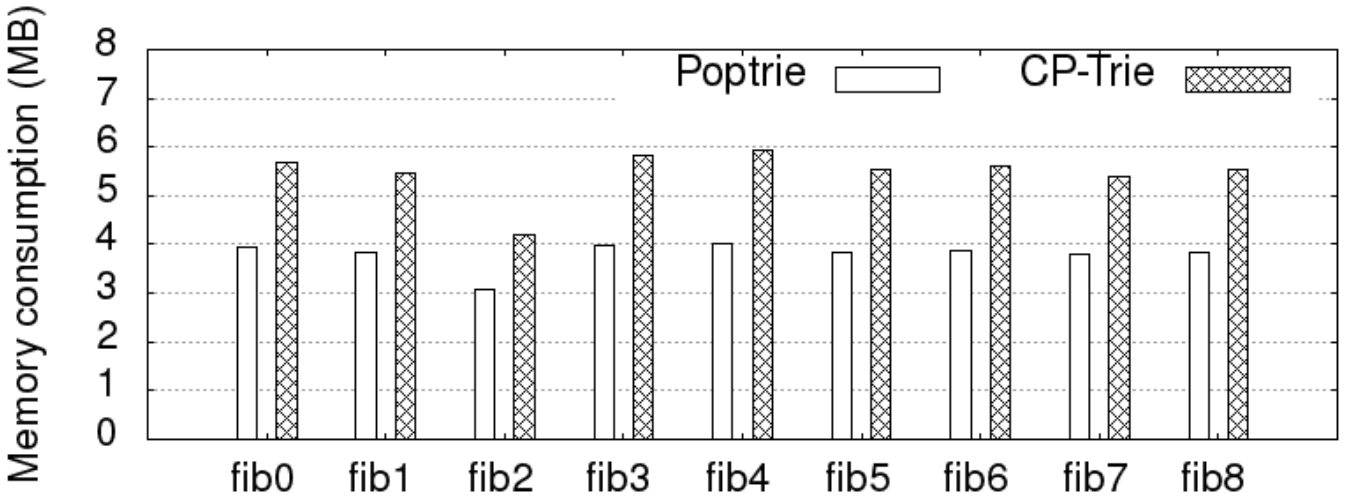
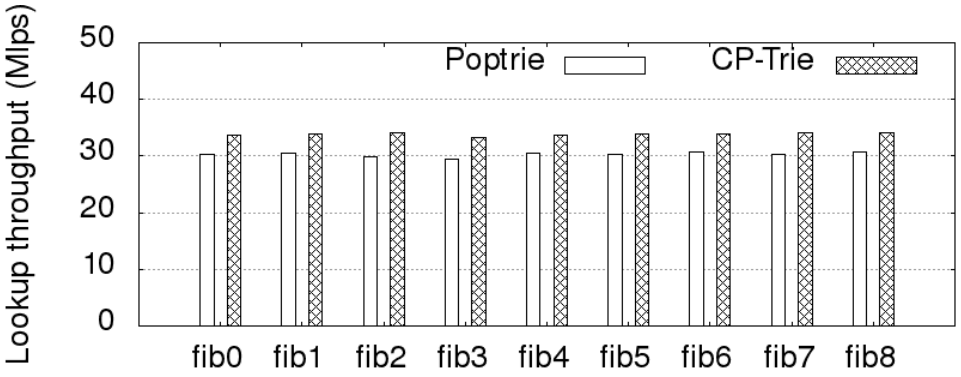
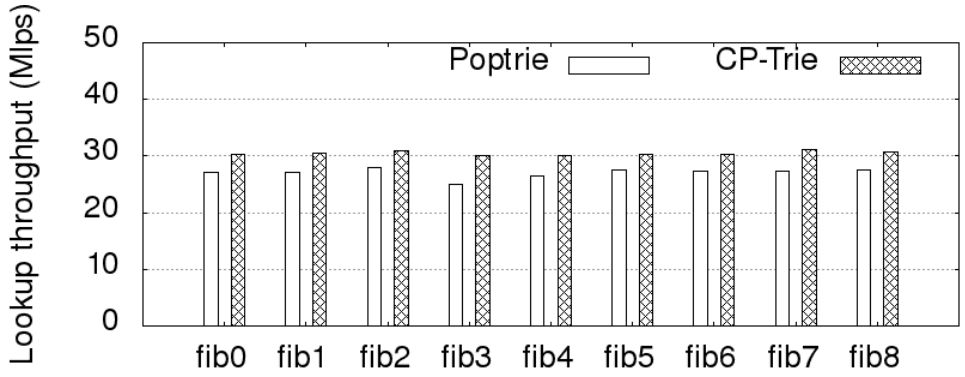


Figure: Memory consumption for different FIBs

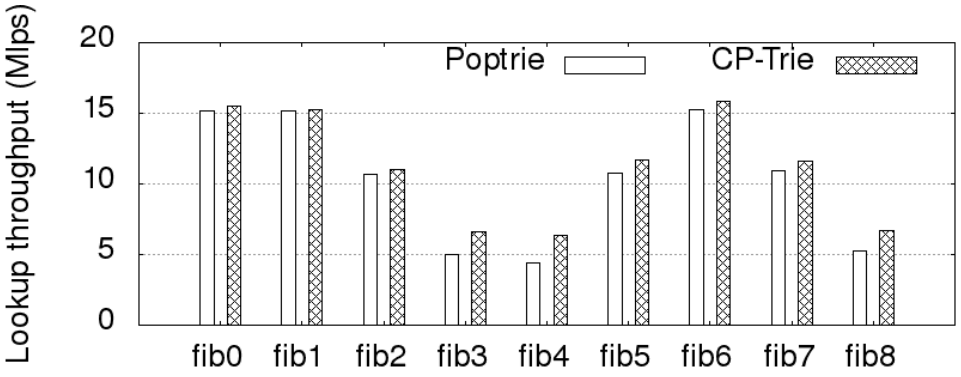
Lookup result



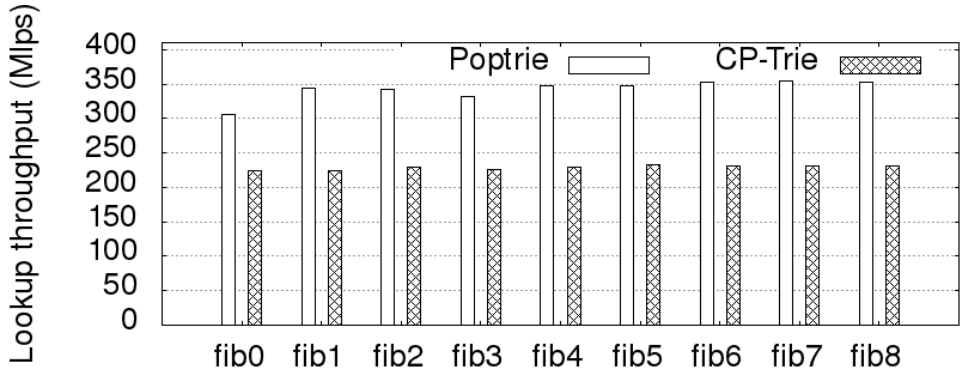
(a) Prefix traffic



(b) Repeated traffic (analogous to real Internet trace)



(c) Sequential traffic



(d) Random traffic (not a realistic traffic)

Update performance

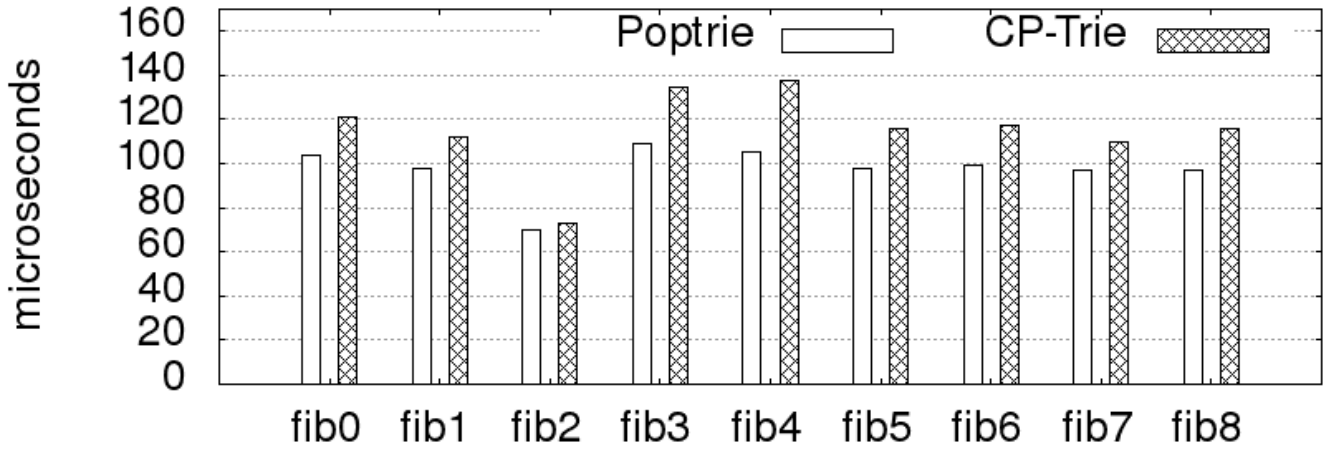


Figure: Average insertion time per prefix (smaller is better)

Thank You