Dynamic Pipelining: Making IP-Lookup Truly Scalable

Jahangir Hasan T. N. Vijaykumar

School of Electrical & Computer Engineering
Purdue University

SIGCOMM 2005

Internet Growth and Router Design

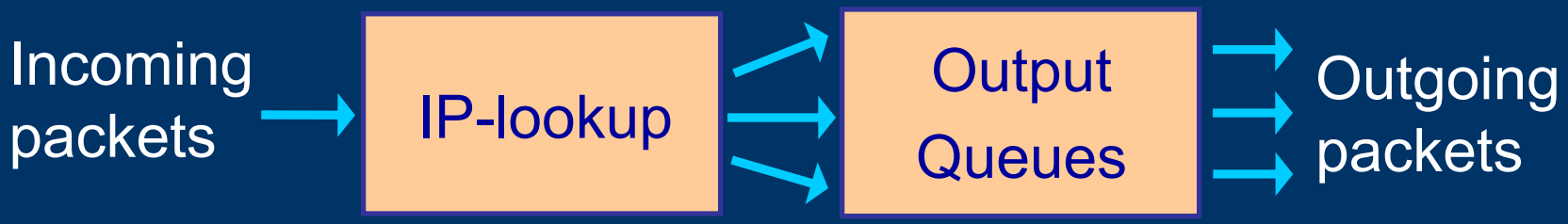
Number of hosts, total traffic growing exponentially
More hosts → larger routing tables
Higher line-rates → faster IP-lookup

Need for worst-case guarantees

- Robust system design / testing
- Network stability / security

Exponential demand yet worst-case guarantee needed

Background on IP-lookup



Routing table : (prefix, next-hop) pairs

IP-lookup : find longest matching prefix for dest addr

Challenge of Scalable IP-lookup

IP-lookup should scale well in:

1. Space – grow slowly with #prefixes
2. Throughput – match line rates
3. Power – grow slowly with #prefixes, line rates
4. Updates – O(1), independent of #prefixes
5. Cost – reasonable chip area

Many IP-lookup proposals to date

None address all factors with worst-case guarantees

This work first to attempt worst-case for all factors

Previous Work

As line-rates grow

- Packet inter-arrival time < memory access times
- Throughput matters more than latency

→ Must overlap multiple lookups using pipelining

	Space	Throughput	Updates	Power	Area
TCAMs	✓	✓	✓		
HLP [Varghese et al – ISCA'03]	✓	✓			
DLP [Basu, Narlikar - Infocom'05]				✓	✓
Our Scheme	✓	✓	✓	✓	✓

Contributions

Scalable Dynamic Pipelining

First to address all 5 factors under worst-case

→ Size 4x better than previous

→ Throughput matches future line rates

Pipeline at hardware and data-structure level

→ Optimum updates

Not just O(1) but exactly 1 write per update

→ Low power, cost for future line rates

Outline

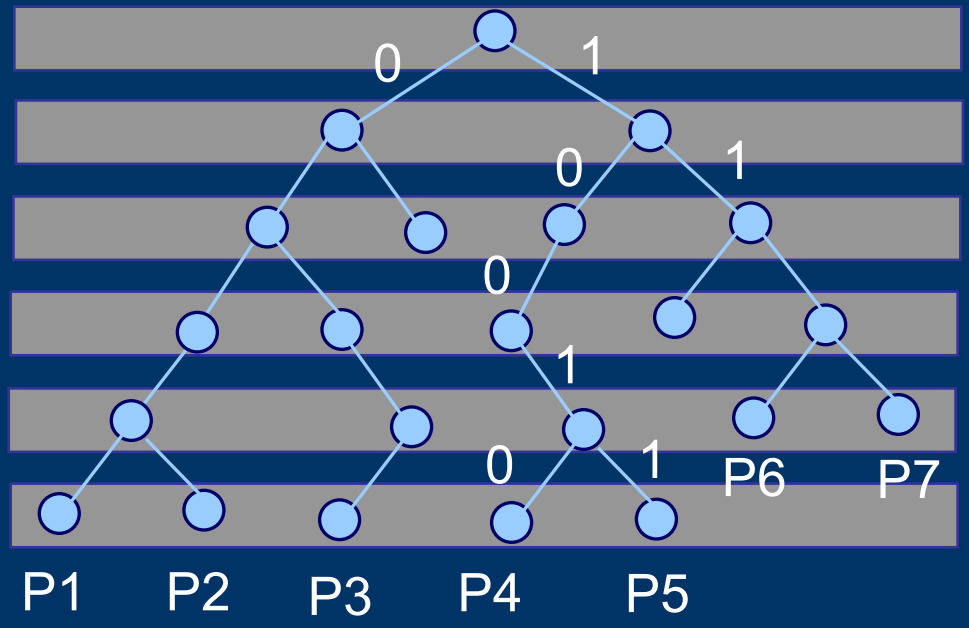
- Introduction
- ➡ Previous pipelined IP-lookup schemes
- Our Scheme: Scalable Dynamic Pipelining
- Experimental Results
- Conclusions

Background: Trie-based IP-lookup

Tree data-structure, prefixes in leaves

Process destination address level-by-level, find longest match

P4 = 10010*



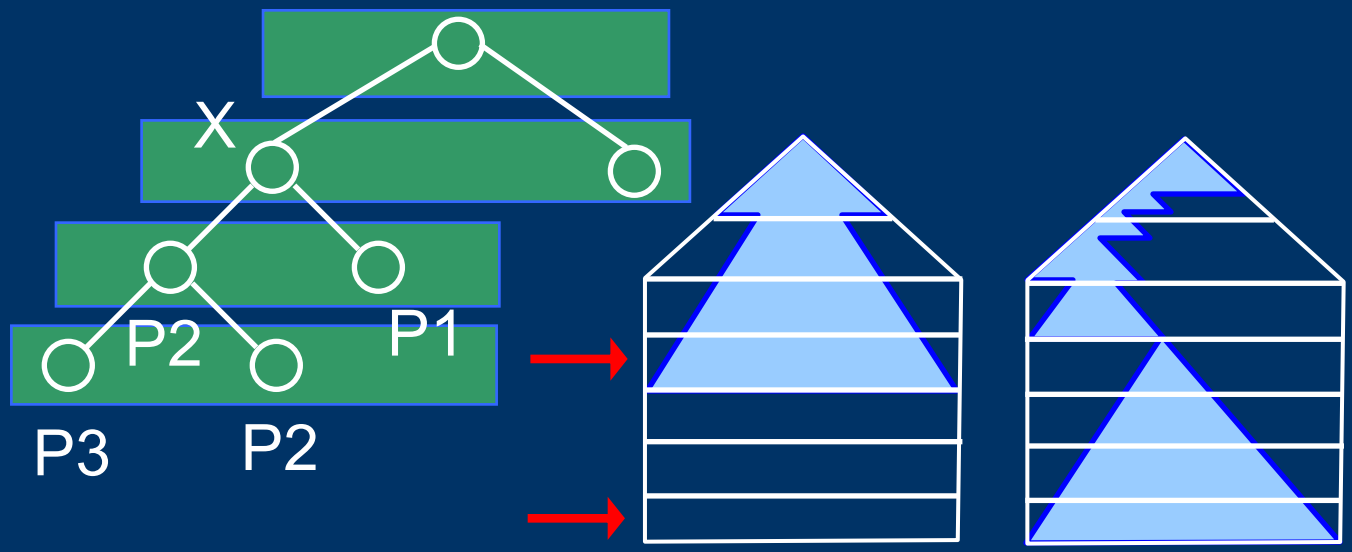
Each level in different stage → overlap multiple packets

Closest Previous Work: [Infocom'03] Data Structure Level Pipelining (DLP)

Map trie level to stage but this is a static mapping

Updates change prefix distribution but mapping persists

0*	P1
00*	P2
000*	P3
..	..



In worst-case any stage can have all prefixes

Large worst-case memory for each stage

Closest Previous Work: [Infocom'03] Data Structure Level Pipelining (DLP)

No bound on worst-case update →

- Could be O(1) using Tree Bitmap
- But constant huge, 1852 memory accesses per update
- [SIGCOMM Comm Review '04]

Outline

- Introduction
- Previous pipelined IP-lookup schemes
- ➡ Our Scheme: Scalable Dynamic Pipelining
- Experimental Results
- Conclusions

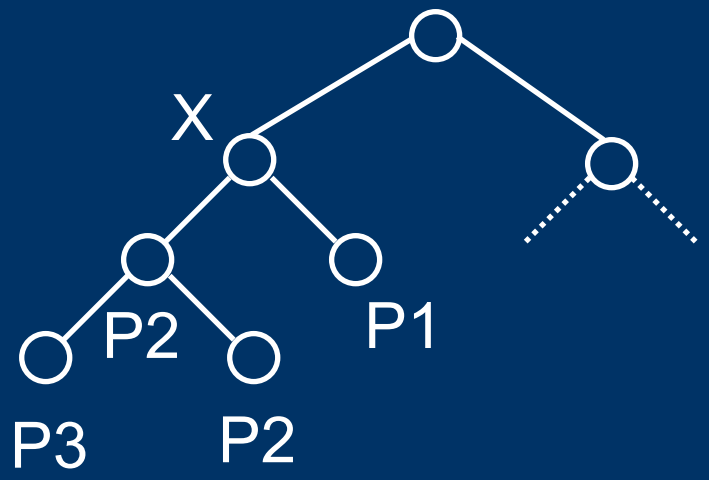
Key Idea: Use Dynamic Mapping

Map node height to stage (instead of level to stage)

Height changes with updates, captures distribution

Hence the name dynamic mapping

0*	P1
00*	P2
000*	P3
..	..



#Nodes at a given height is limited (but not at given level)→

Limited #nodes per stage → small per-stage memory

But Updates Inefficient?

Updates may change height of arbitrarily many nodes

Must migrate all affected nodes to new stages

Does this mean updates inefficient?

Surprisingly No …

Leverage very mapping that causes problem

Achieve optimum updates

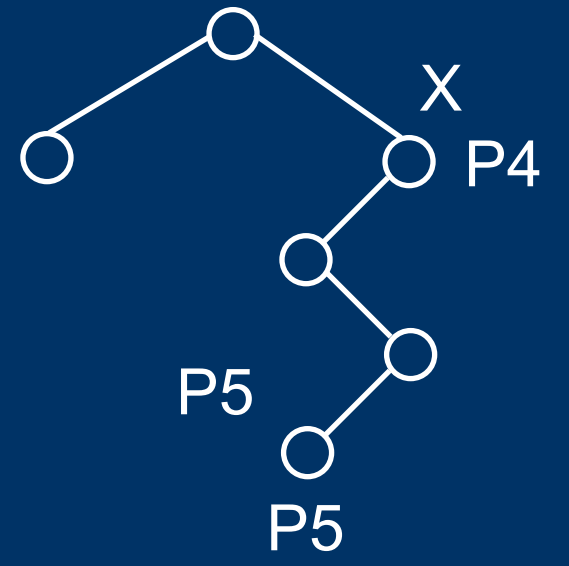
A Problematic Peculiarity of Tries

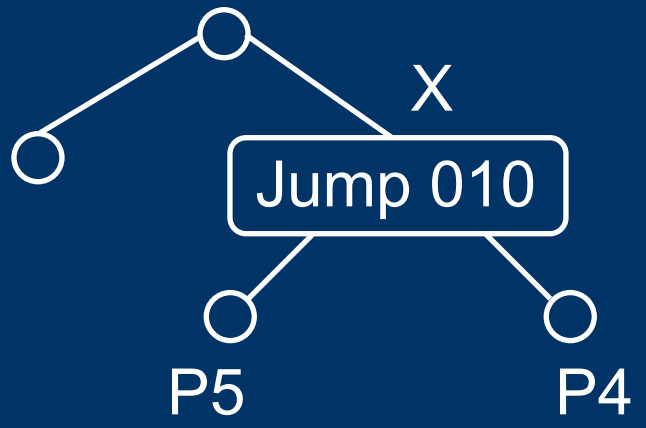
Some cases height does not capture distribution

- String of one child nodes

Artificially distort relation of height and distribution

..	..
1*	P4
1010*	P5
..	..





Jump nodes compress away such strings

Restore relation between height and distribution

1-bit Tries with Jump Nodes

Key properties

(1) Number of leaves = number of prefixes

No replication

Avoids inflation of prefix expansion, leaf-pushing

(2) Updates do not propagate to subtrees

No replication

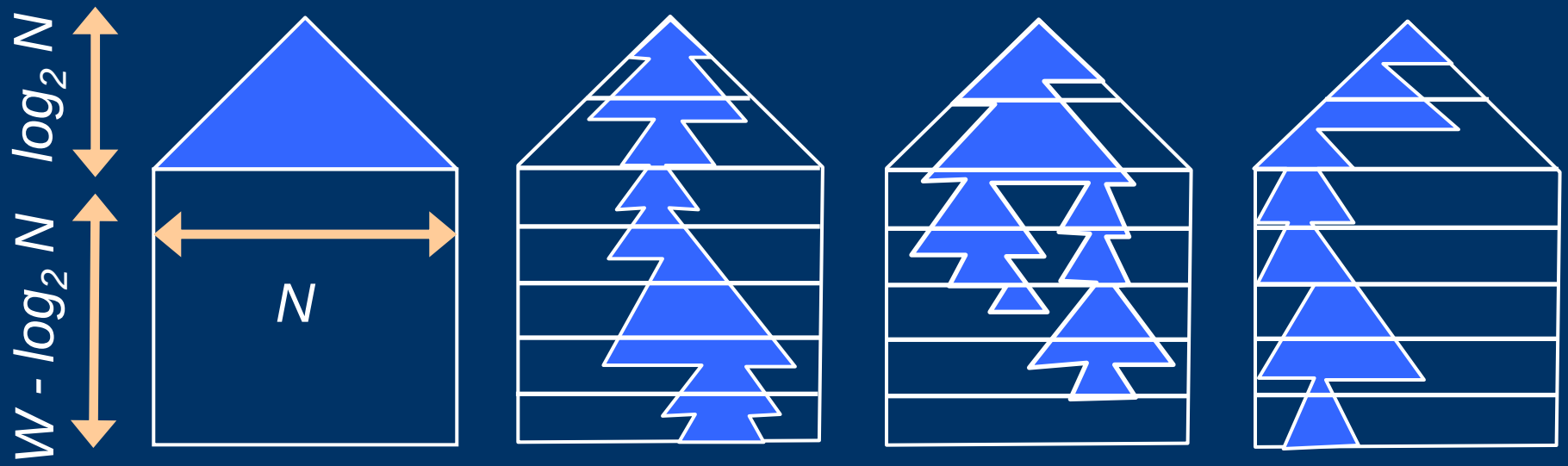
(3) Each internal node has 2 children

Jump nodes collapse away single-child nodes

Total versus Per-Stage Memory

Jump-nodes bound total size by 2N

Would DLP+Jump nodes → small per-stage memory?



No, DLP is still static mapping → large worst-case per-stage

Total bounded but not per-stage

SDP's Per-Stage Memory Bound

Proposition:

Map all nodes of height h to $(W\text{-}h)^{th}$ pipeline stage

Result:

Size of k^{th} stage = min(N / (W-k) , 2^k)

Key Observation #1

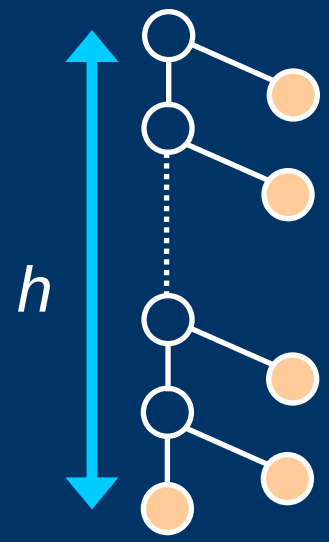
A node of height *h* has at least *h* prefixes in its subtree

At least one path of length *h* to some leaf

h -1 nodes along path

Each node leads to at least 1 leaf

Path has *h -1+1* leaves = *h* prefixes



Key Observation #2

No more than N / h nodes of height h for any prefix distribution

Assume more than N / h nodes of height h

Each accounts for at least h prefixes (obs #1)

Total prefixes would exceed N

By contradiction, obs #2 is true

Main Result of the Proposition

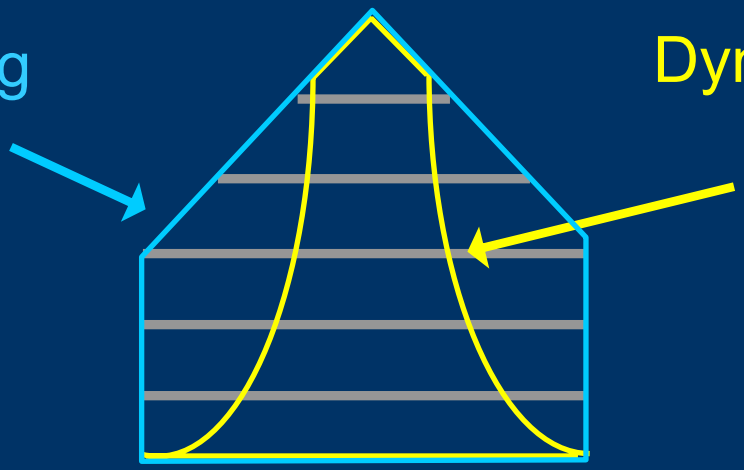
Map all nodes of height *h* to (W-h)th pipeline stage

*k*th stage has only *N* / (*W-k*) nodes from obs #2

1-bit trie has binary fanout → at most 2^k nodes in *k*th stage

Size of *k*th stage = min(*N* / (*W-k*) , 2^k) nodes

Static pipelining (DLP)



Dynamic pipelining (SDP)

Results in ~20 MB for 1 million prefix

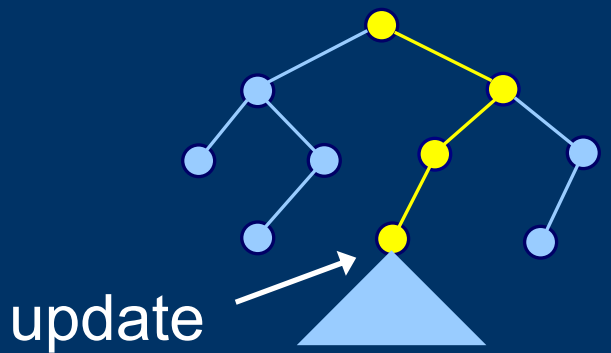
4x better than DLP

Optimum Incremental Updates

1 update → change height and stage of many nodes

Must migrate all affected nodes → inefficient update?

Key: Only ancestors' heights can be affected



Each ancestor in different stage

= 1 node-write in each stage

= 1 write bubble for any update

Updating SDP not just O(1) but exactly 1

Efficient Memory Management

No variable striding / compression → all nodes same size

No fragmentation/compaction upon updates

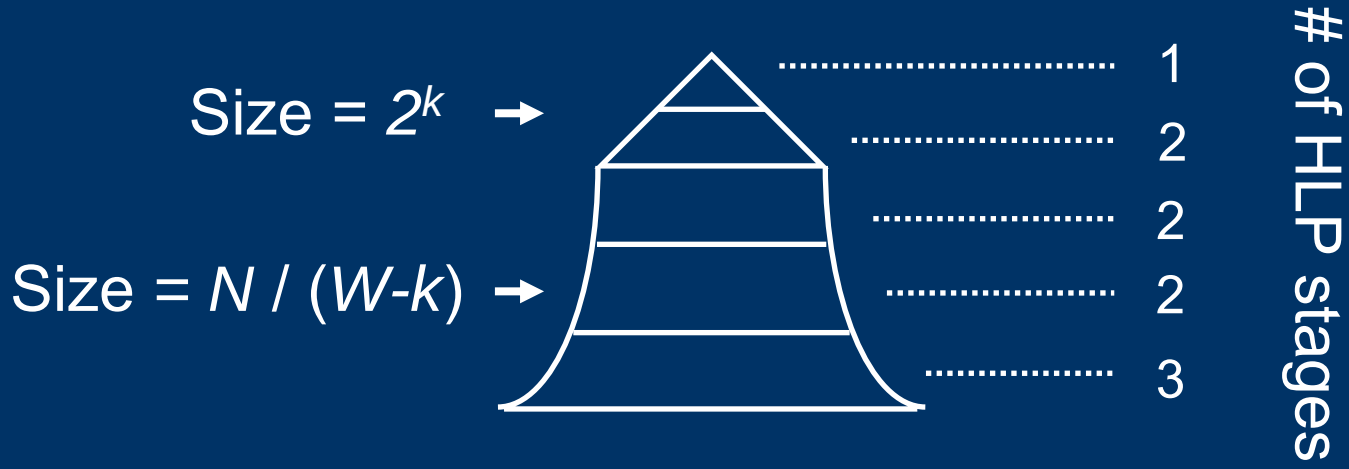
Scaling SDP for Throughput

Each SDP stage can be further pipelined in hardware

HLP [ISCA'03] pipelined only in hardware without DLP

Too deep at high line-rates

Combine HLP + SDP for feasibly deep hardware



Throughput matches future line rates

Outline

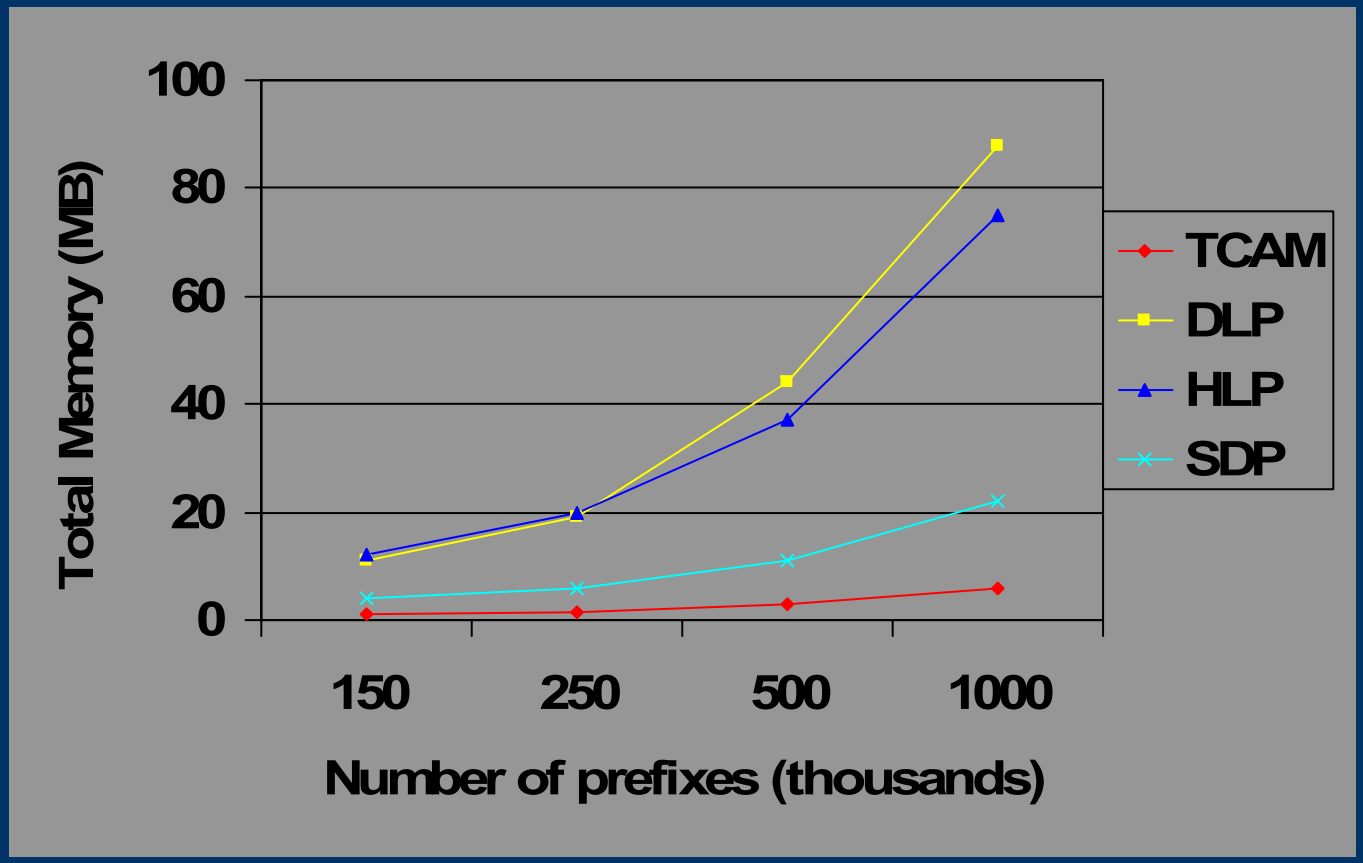
- Introduction
- Previous pipelined IP-lookup schemes
- Our Scheme: Scalable Dynamic Pipelining
- ➡ Experimental Results
- Conclusions

Experimental Methodology

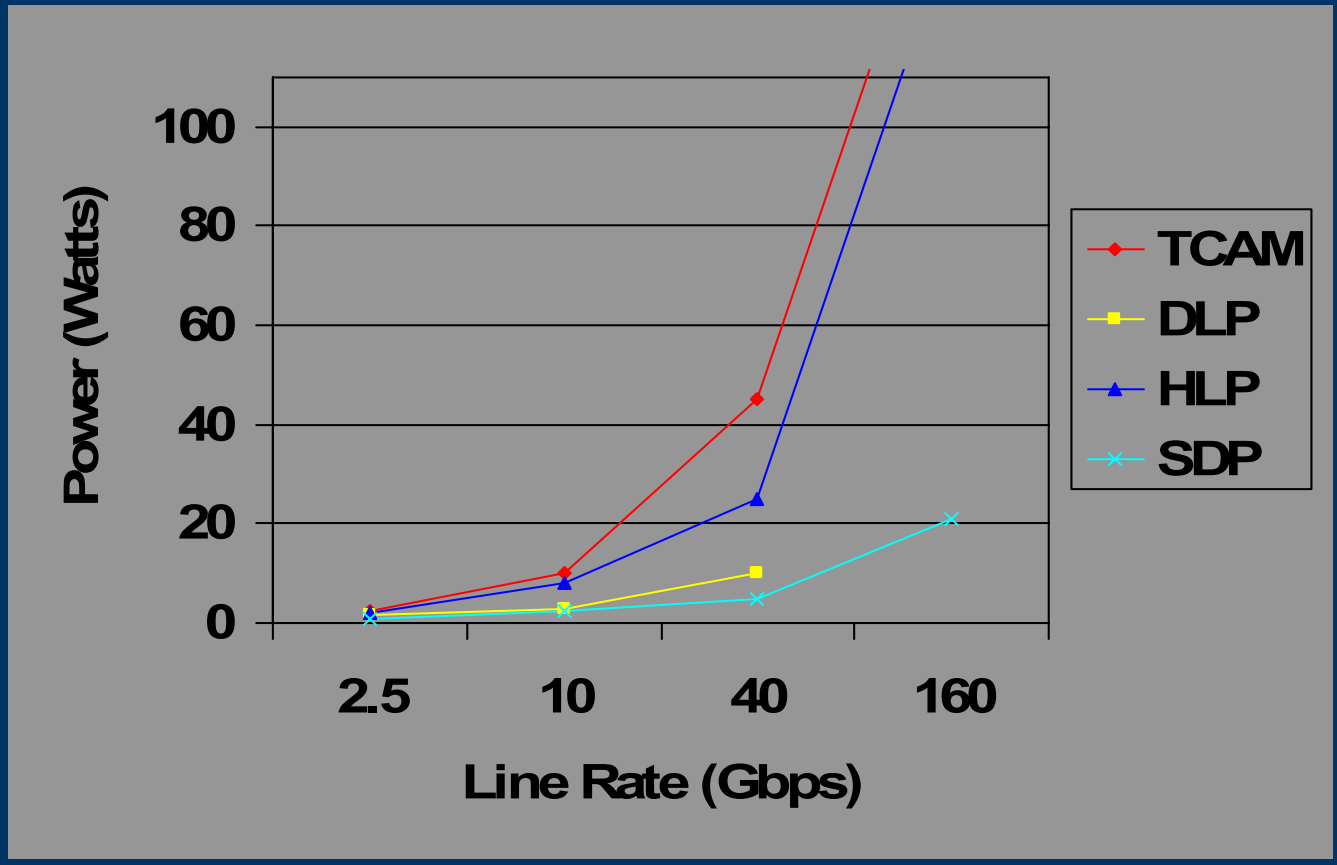
Worst-case prefix distribution, packet arrival rate

CACTI 2.0 for simulating memories
Modify CACTI to model TCAM, HLP, DLP and SDP

Dynamic Pipelining: Tighter Memory Bound

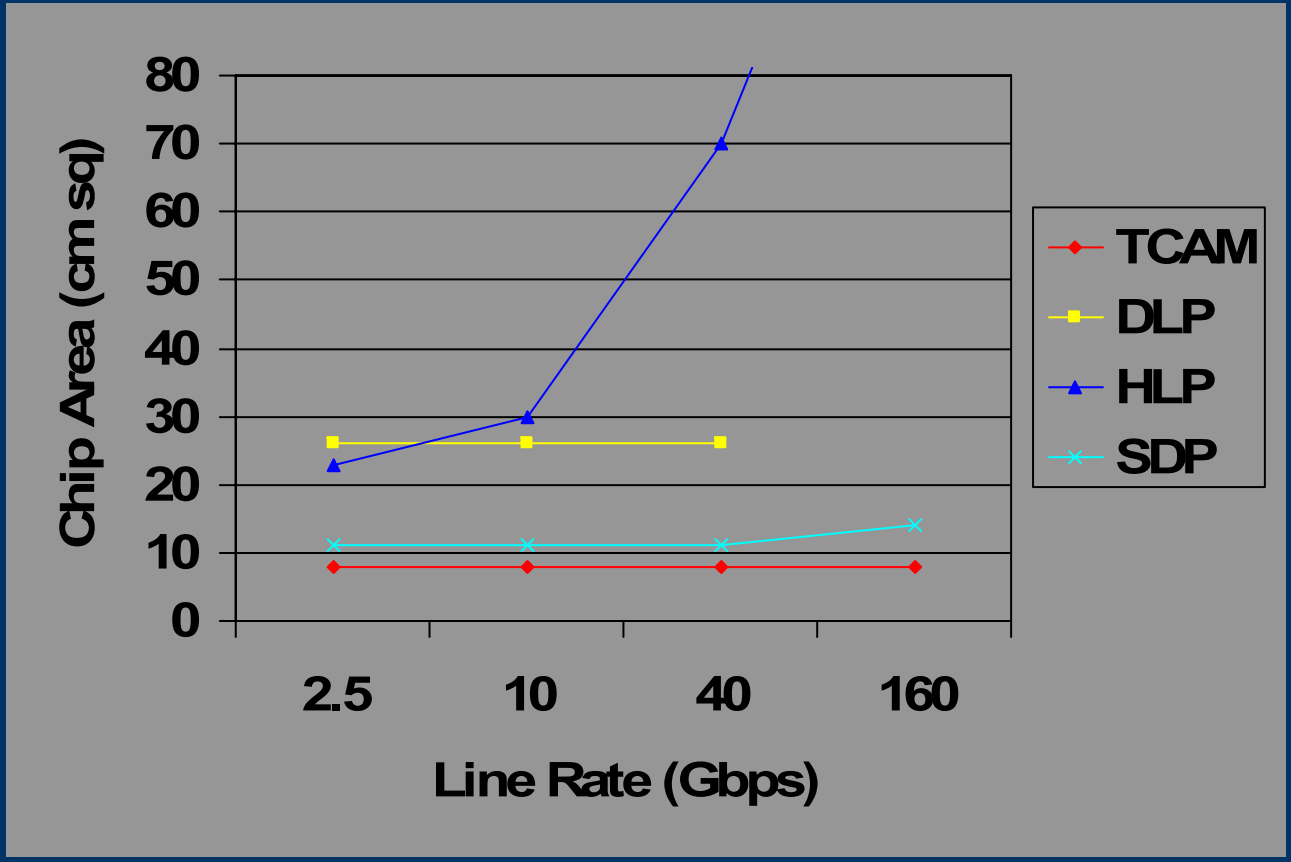


Dynamic Pipelining: Low Power



SDP's small memory + shallow hardware pipeline: Low power

Dynamic Pipelining: Small Area



* TCAM: pipelining overhead ignored, unfair advantage

SDP's small memory + shallow hardware pipeline:Small area

Outline

- Introduction
- Previous pipelined IP-lookup schemes
- Our Scheme: Scalable Dynamic Pipelining
- Experimental Results
- ➡ Conclusions

Conclusions

Previous schemes use static level-to-stage mapping
We proposed dynamic height-to-stage mapping

Dynamic mapping enables SDP's scalability

- Worst-case memory size 4x better
- Scales well upto 160Gbps
- Optimum update, 1 write-bubble per update
- Efficient memory management
- Low power
- Low implementation cost

Questions ?

The following slides are the actual questions and answers which were asked after the presentation at SIGCOMM ‘05

Did you use real routing tables in the experiments? Which ones?

No

We used the worst-case prefix distribution for all experiments

Distribution shown in paper, Section 3.2.1

Gives intiuitive “proof” for the distribution being worst case

Same worst case is used by previous work

DLP [Basu, Narlikar – INFOCOM ‘03]

"Tree Based Router Architecture" in ISCA'05 solves same problem?

"A Tree Based Router Search Engine Architecture with Single Port Memories" Baboescu et. al., *International Symposium on Computer Architecture* (ISCA) 2005

Same question was the main complaint in a review

ISCA'05 was in June, well after SIGCOMM submission

Having said that, the ISCA paper

- Does not show how to size stages for N prefixes
- Makes stage sizes equal given a particular distribution
- Shows that building this balanced pipeline is O(N)
- Does not address how to maintain balance upon updates
- Does not address throughput scalability
- Has no worst analysis for size, throughput, update cost

Large number of banks (32 to 128) high implementation cost?

1 million prefixes = 20MB = 160Mbits

We are talking about on-chip implementation

There is no pin-count issue

We show these actual area estimates for 100nm

Of course, by the time we reach 1 million prefixes

Technology will scale to allow 160Mbit on-chip

Scaling our 100nm area to 50nm gives < 4 cm sq

Did you assume a 1-bit trie for all schemes and all experiments?

HLP and DLP are multi-bit trie schemes

We address this issue in Section 6.1

- First explore design space over all possible strides
- Pick the optimum stride for HLP and for DLP

All experiments performed using these optimum strides