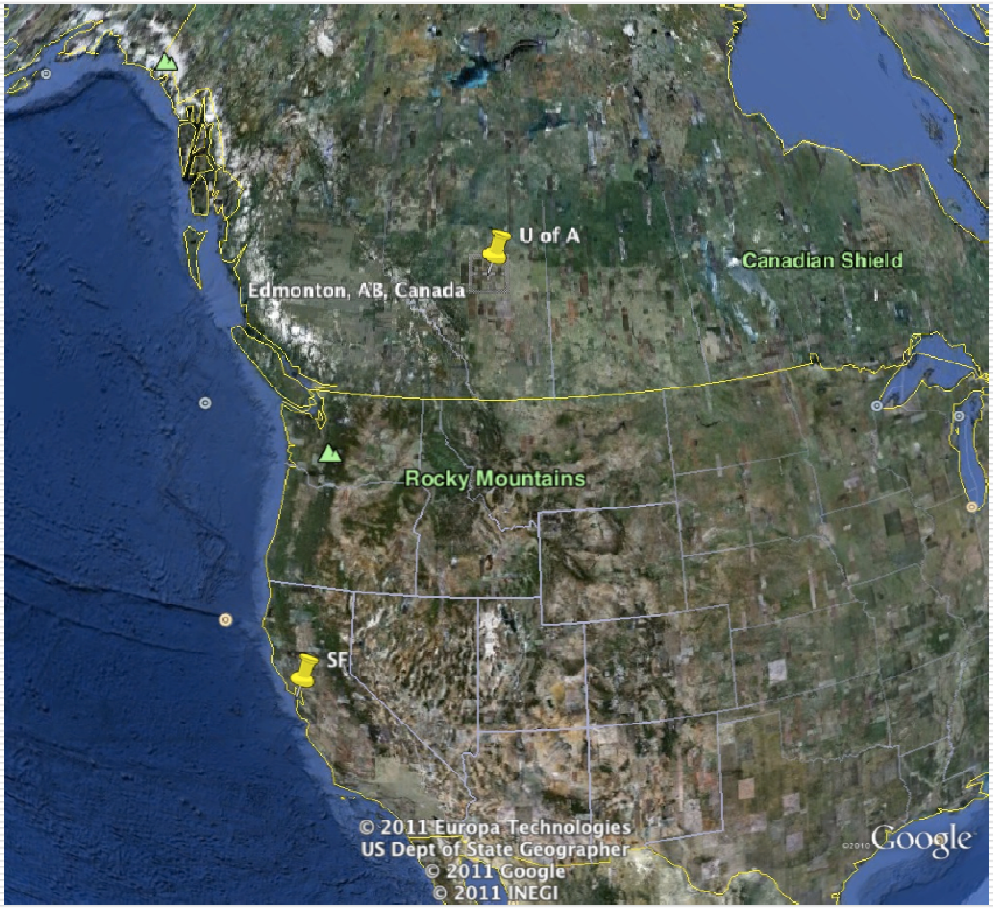

U of A
Canadian Shield
Edmonton, AB, Canada
Rocky Mountains
SF
© 2011 Europa Technologies
US Dept of State Geographer
© 2011 Google
© 2011 INEGI
Google

Two-level Cache Architecture to Reduce Memory Accesses for IP Lookups

Sunil Ravinder, Mario Nascimento, Mike MacGregor

Department of Computing Science

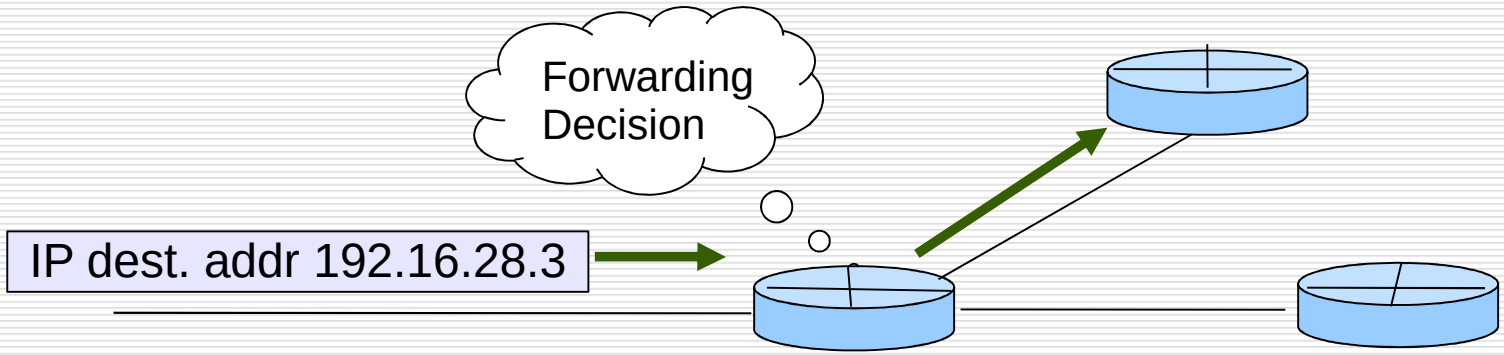
University of Alberta

Outline

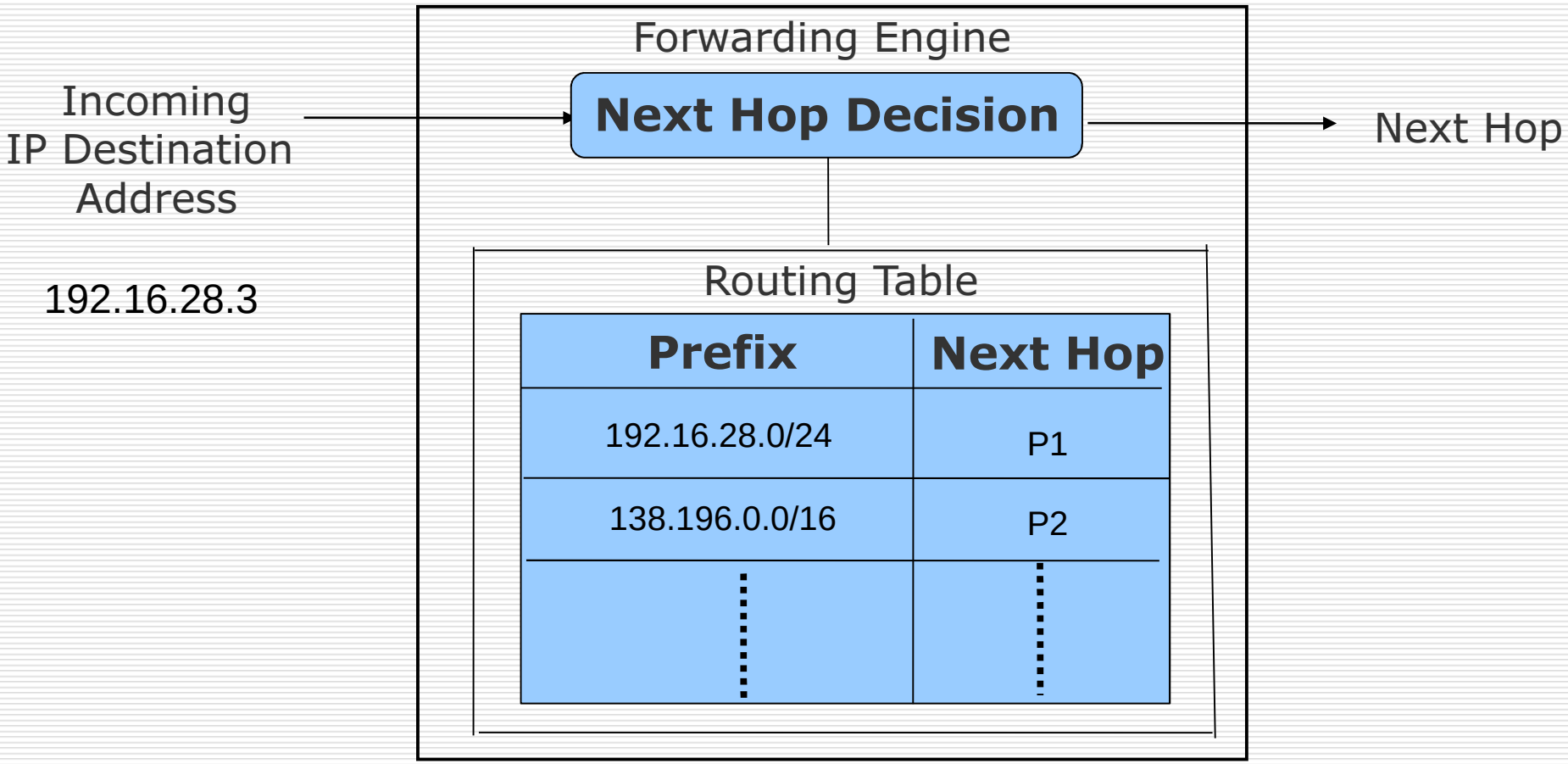
- IP Lookups: Longest-prefix matching
- Prefix caching
- Substride caching, 2-level architecture
- Modelling cache hit rates
- Optimal two-level designs
- Experimental results
- Conclusions

IP Lookups

- IP lookups provide forwarding decisions based on the incoming packet's destination address.
- Involves searching the longest prefix that matches the packet's destination address.
- Successful prefix search determines the output port (next hop).



IP Lookups


Forwarding Engine
Incoming IP Destination Address
192.16.28.3
Next Hop Decision
Next Hop
Routing Table
Prefix
Next Hop
192.16.28.0/24
P1
138.196.0.0/16
P2

A classic solution: Prefix caching

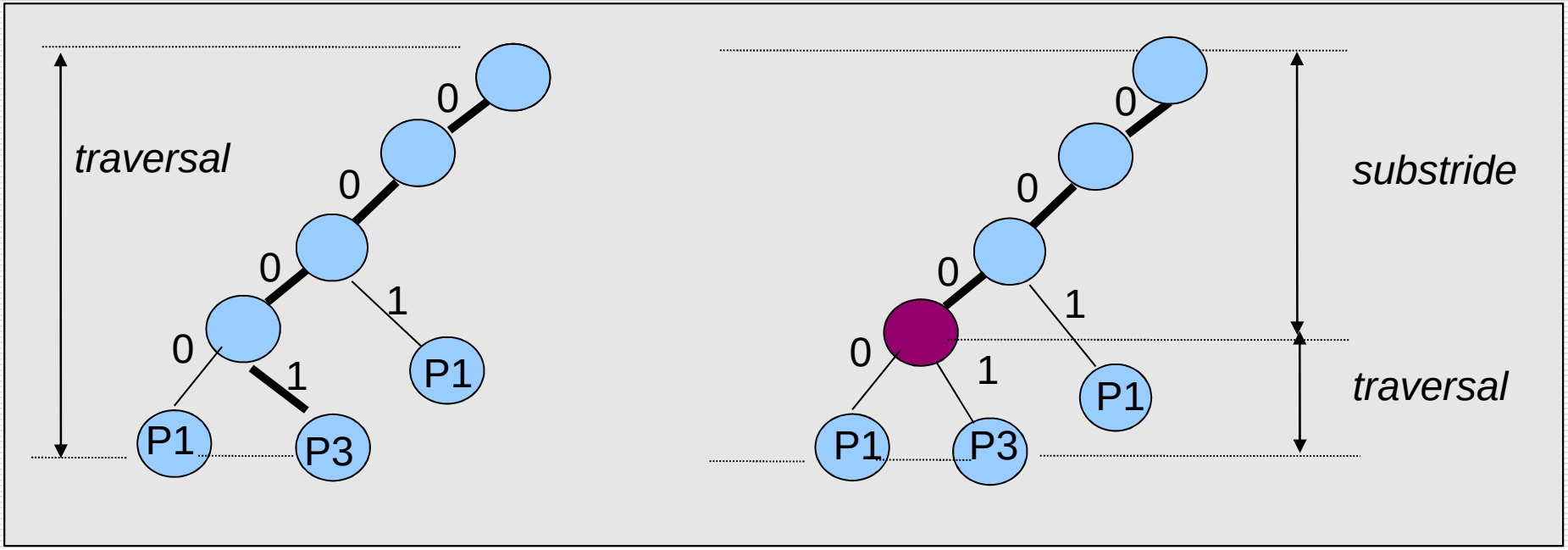
- Cache prefixes (and next hops) from previous lookups.
- A hit in the prefix cache gives the next hop.
- Benefits from temporal locality.
- Better than just caching full addresses because it also captures spatial locality.

Assisting prefix cache misses

- Missing the PC usually forces a table traversal.
- Let’s try to decrease the cost of a PC miss!
- Dynamic substride cache: A new cache organization that stores “substrides”.
- Lookups that hit in the DSC proceed directly from an internal node within the routing table - skipping several memory accesses.

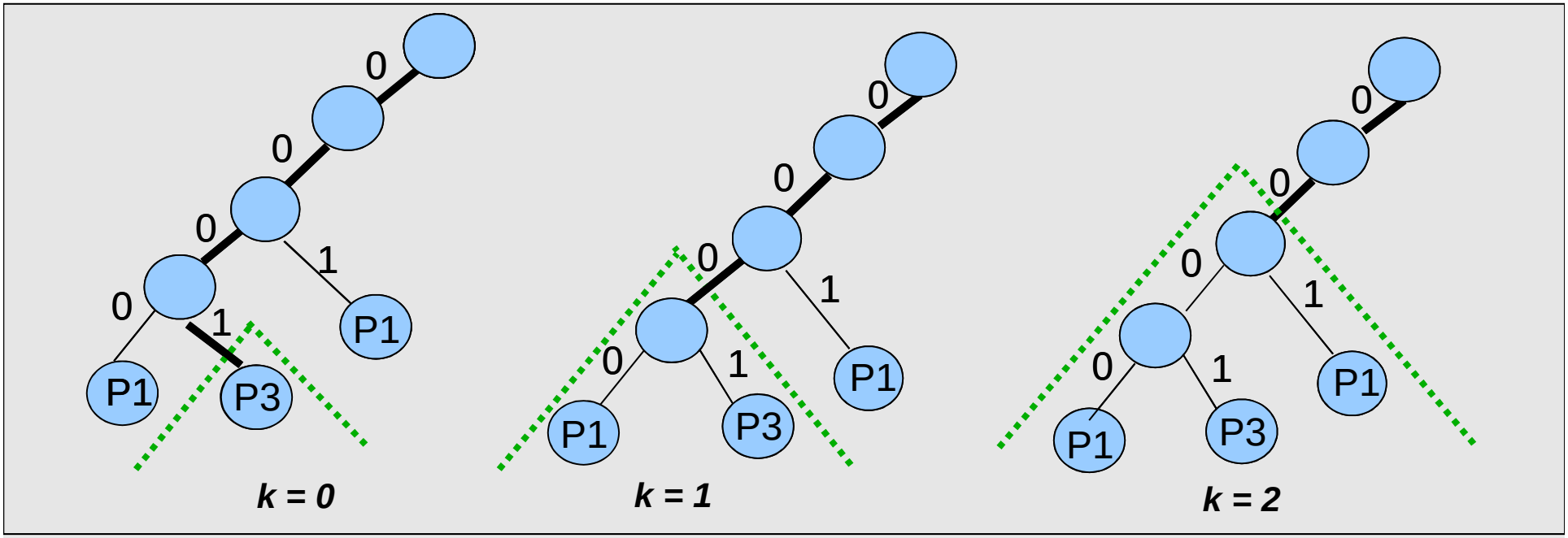
Dynamic substride cache (DSC)

- We obtain substrides by shortening a (recently looked up) prefix by *k* bits.
- e.g. for prefix 0001* and *k = 1* we obtain the substride 000*.
- DSC stores <substride, table node addr> pairs.

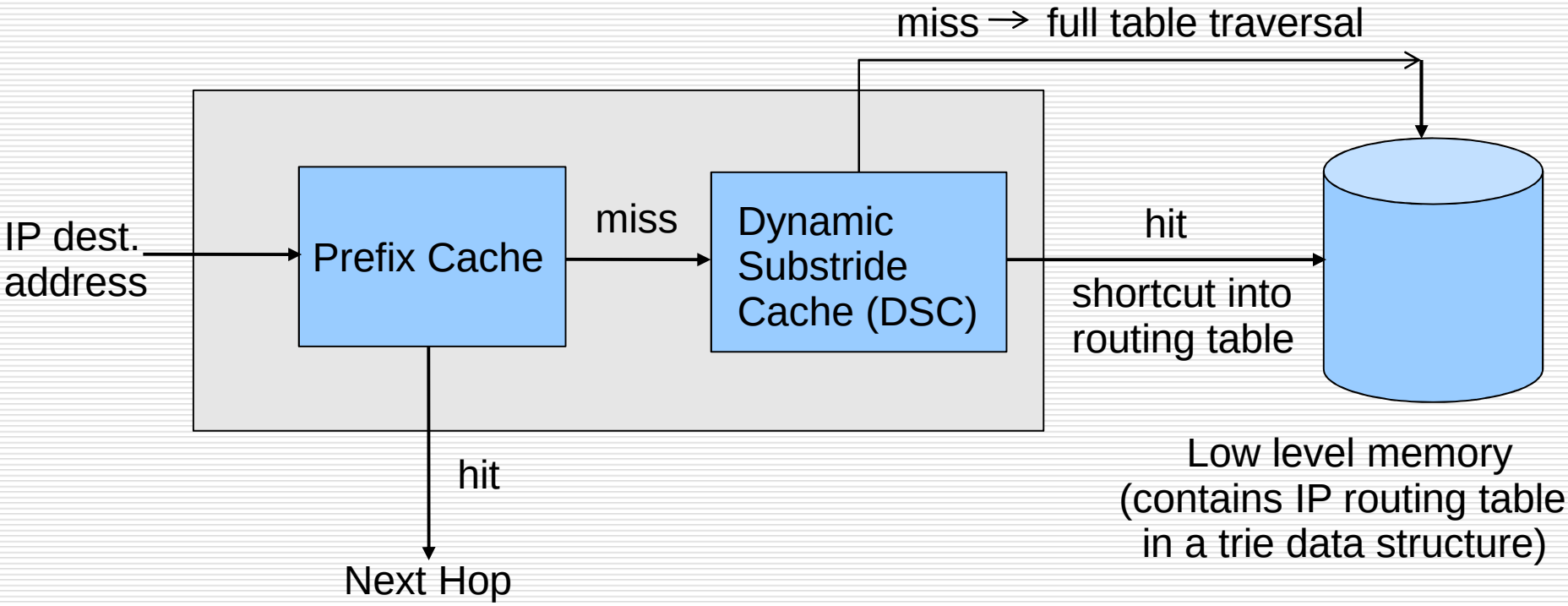


Dynamic substride cache (DSC)

- A substride captures a wider IP address space than its parent prefix.
- Larger values of *k* increase the space spanned (↑DSC hit rate), but also increase the number of memory accesses after a DSC hit (↑hit cost)



Proposed architecture



Proposed architecture

- Prefix cache sees destination address stream first. Focused on exploiting temporal and spatial locality.
- DSC assists lookups that fall in “popular” IP address ranges. Assists lookups that miss the prefix cache.
- Lookups that miss the PC, but hit in the DSC will skip many memory references.

A short example

[A]	111100 101000 (PC) 000101 ---------- 1111 1010 (DSC) 0001	**New reference:** 000100111	111100 101000 (PC) 000101 ---------- 1111 1010 (DSC) **0001**	**Miss in PC** **Hit in DSC**
[B]	**000100** 111100 (PC) 101000 ---------- **0001** 1111 (DSC) 1010	**New reference:** 01111001010	000100 111100 (PC) 101000 ---------- 0001 1111 (DSC) 1010	**Miss in PC** **Miss in DSC**
[C]	**011110** 000100 (PC) 111100 ---------- **0111** 0001 (DSC) 1111	**New reference:** 111100110011	011110 000100 (PC) **111100** ---------- 0111 0001 (DSC) 1111	**Hit in PC**

Optimal design

- Partition a given number of cache lines between the PC and DSC.
- Our objective is to minimize the average number of memory accesses.
- Requires a model for cache hit rates.
- Start from the footprint function:

$$u(k) = Wk^a$$

- *u(k)* is the number of unique references in a stream at the time of the *k*th reference
- *W* reflects working set size, *a* reflects locality

Hit rate modelling

- take derivative and evaluate at $u(k)=C$
- derivative of $u(k)$ is the instantaneous rate of unique references
- evaluate at point where unique refs. observed have just filled the cache – this gives the miss rate

$$M(C) = aW^{1/a}C^{(1-1/a)}$$

- this does not obey the boundary condition at $C=0$

Hit rate modelling

- We propose:

$$H(x) = 1 - \left(\frac{x}{A} + 1 \right)^{-\theta}$$

- θ captures the locality in the trace
- *A* captures the initial hit rate behavior
- *H(0)=0, H(∞)=1*
- use the same general form of model for both the PC and the DSC

Optimization Tableau

$$\min\left(F_0 m_0 + F_1 m_1 + F_2 m_2\right)$$

- F_i are the number of references resolved by the PC, DSC and routing table, respectively
- m_i is the number of memory references required when resolving a reference at level *i*
- $m_0=0$, $m_1=5$, $m_2=32$

Subject to:

total capacity

$$c_0 + c_1 = C$$

DSC hits

$$F_1 = \left[L - L \times H_0\left(c_0\right)\right] \times H_1\left(c_1\right) \le L$$

full table traversals

$$F_2 = L - F_0 - F_1 \le L$$

Hit rate models in the tableau

- PC and DSC hit rates are modeled as

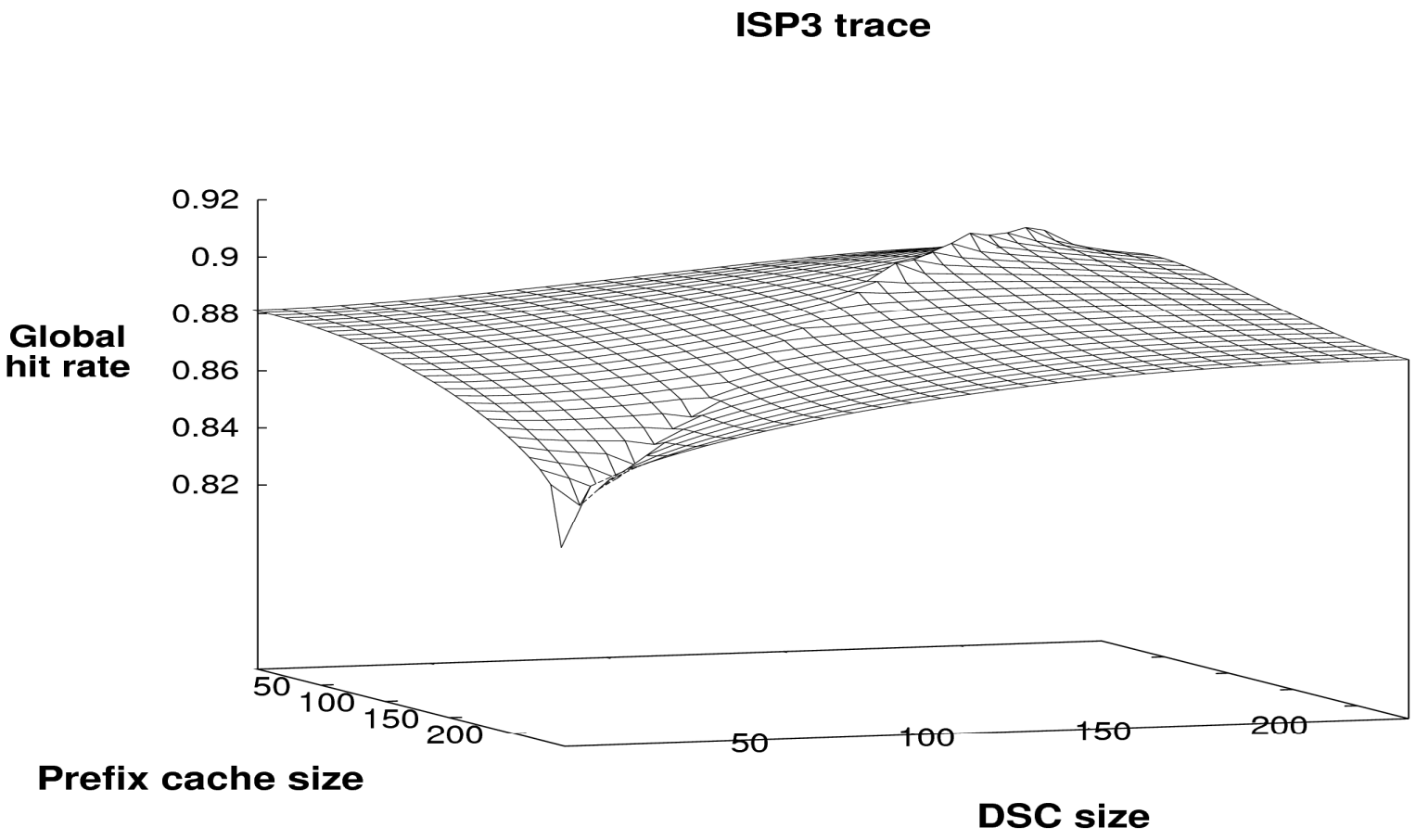
$$H(x) = 1 - \left(\frac{x}{A} + 1\right)^{-\theta}$$

- $A_0(c_0)$, $A_1(c_1)$, $\theta_0(c_0)$ and $\theta_1(c_1)$ are all nonlinear functions (rather than just constants)
- break them into linear segments and add selector variables to choose the correct segment
- interpolate linearly within a segment

Experiments

- seven traces: four high locality, three low locality
- all from public sources, paired with a routing table from the same router
- 6K to 292K prefixes in the tables
- verified optimization tableau by comparing to exhaustive search

Example optimization surface



Average number of memory accesses per lookup

Scheme	upcb.2	ISP3	FUNET	ISP2	ISP1	upcb.1	bell
PC+DSC	4.26	3.26	2.64	1.38	0.89	0.44	0.37
MPC	6.03	4.23	2.78	1.52	1.36	0.61	0.48
Shyu	6.42	4.85	4.25	1.71	1.49	1.13	0.97
Akhbari-zadeh	7.68	5.62	4.32	1.79	3.68	1.92	1.56
Kasnavi	8.34	7.63	5.69	2.84	5.18	4.39	3.67
Multizone	10.91	8.47	6.04	2.95	5.77	6.13	5.02
Address cache	16.67	9.73	6.76	3.26	6.65	9.16	8.94

Conclusions

- Adding the DSC following the PC reduces average number of memory accesses per lookup
- Reductions up to 40% compared to other current proposals.
- Works well even with low-locality traffic.
- Incremental updates possible.
- Cache hit rate model obeys boundary conditions, usable for caches in general.
- Optimization tableau verified as accurate by comparing to exhaustive search.