

TOPLUS

Topology-centric Look-Up Service

Luis Garcés-Erice

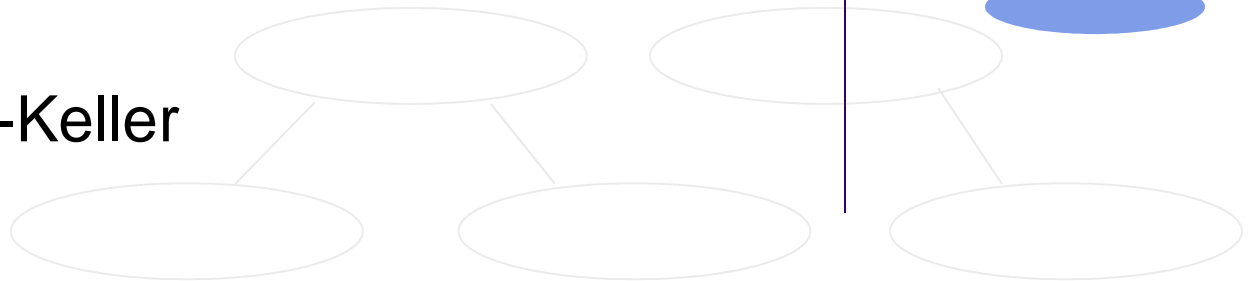
Keith W. Ross

Ernst W. Biersack

Pascal A. Felber

Guillaume Urvoy-Keller

Institut Eurécom – Sophia Antipolis, France





Outline

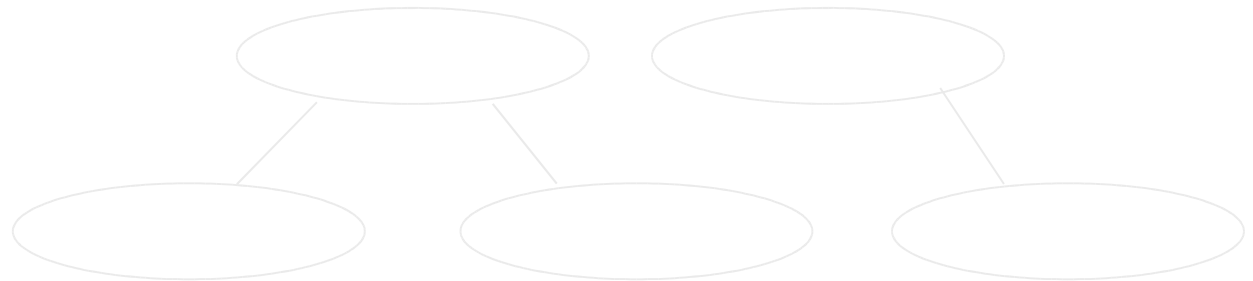
Very brief introduction to DHTs

Motivation and Goals

TOPLUS

Benchmarks

Conclusion





Introduction to DHTs

P2P Lookup Services

Assign resources to peers

Locate resources upon request

Structured P2P systems:

A **key** is a resource identifier.

A Hash Table maps a key to a bucket through a hash function **$h()$**

$h(\text{key}) \rightarrow 0, 1, 2 \dots N-1$, all **$N$** possible outputs of **$h()$**

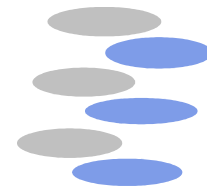
In Distributed Hash Tables (DHTs) each bucket is a peer

Key assigned to peer with the “**closest**” id

Chord, CAN, Pastry

Scalability: Each peer knows a subset of all peers

$h(\text{key})$ must be **routed** to corresponding “bucket”



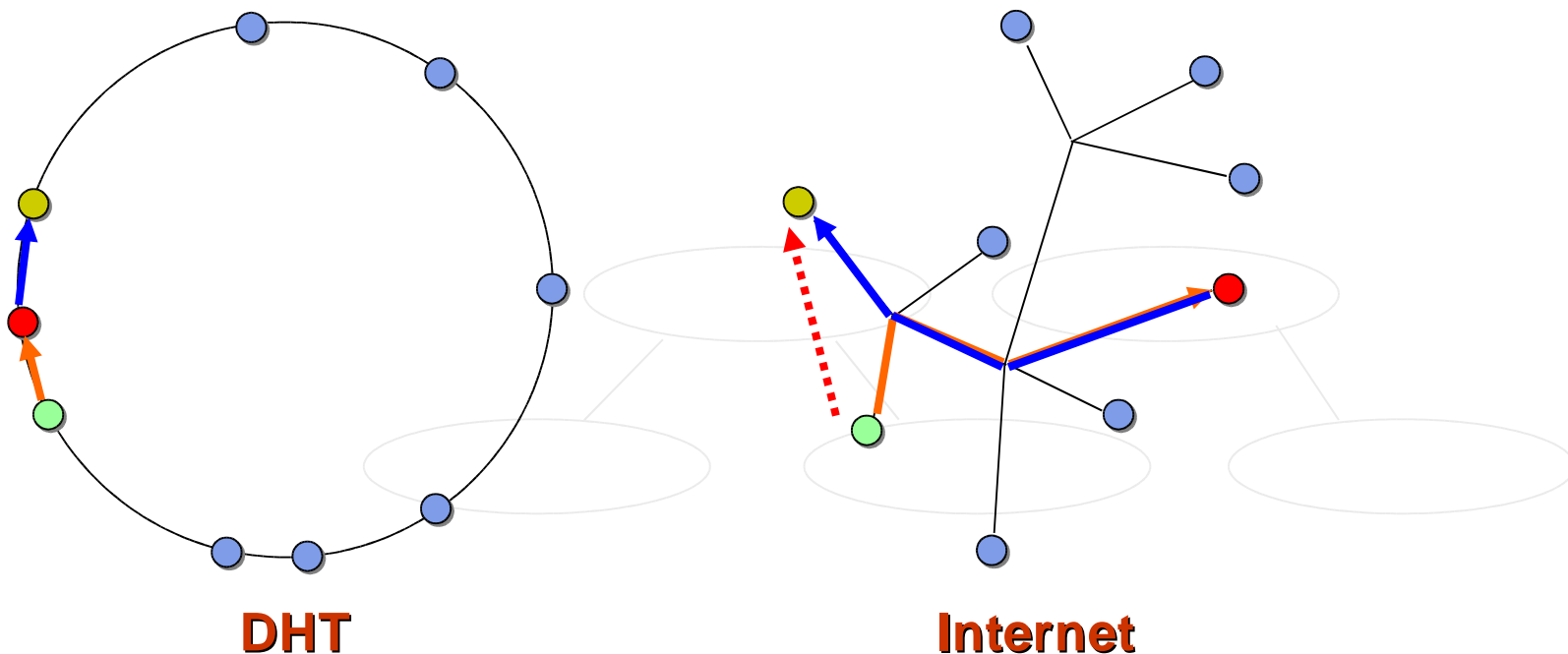
Motivation

P2P DHT overlay networks are built on top of the Internet

Queries are routed through the overlay

Neighbours in the overlay network are not close in the physical Internet network

Pastry uses closest neighbour to route (**closest** \neq **close**)





Goal

Overlay Goal

Provide small stretch: route queries in the overlay as close as possible to IP path

How can the lookup path “follow” the IP path?

Want to get close to the destination in the first step

Stretch close to **1**

Our solution

TOPLUS (TOPology-centric Look-Up Service)

A “*benchmark*” to topology-aware DHTs



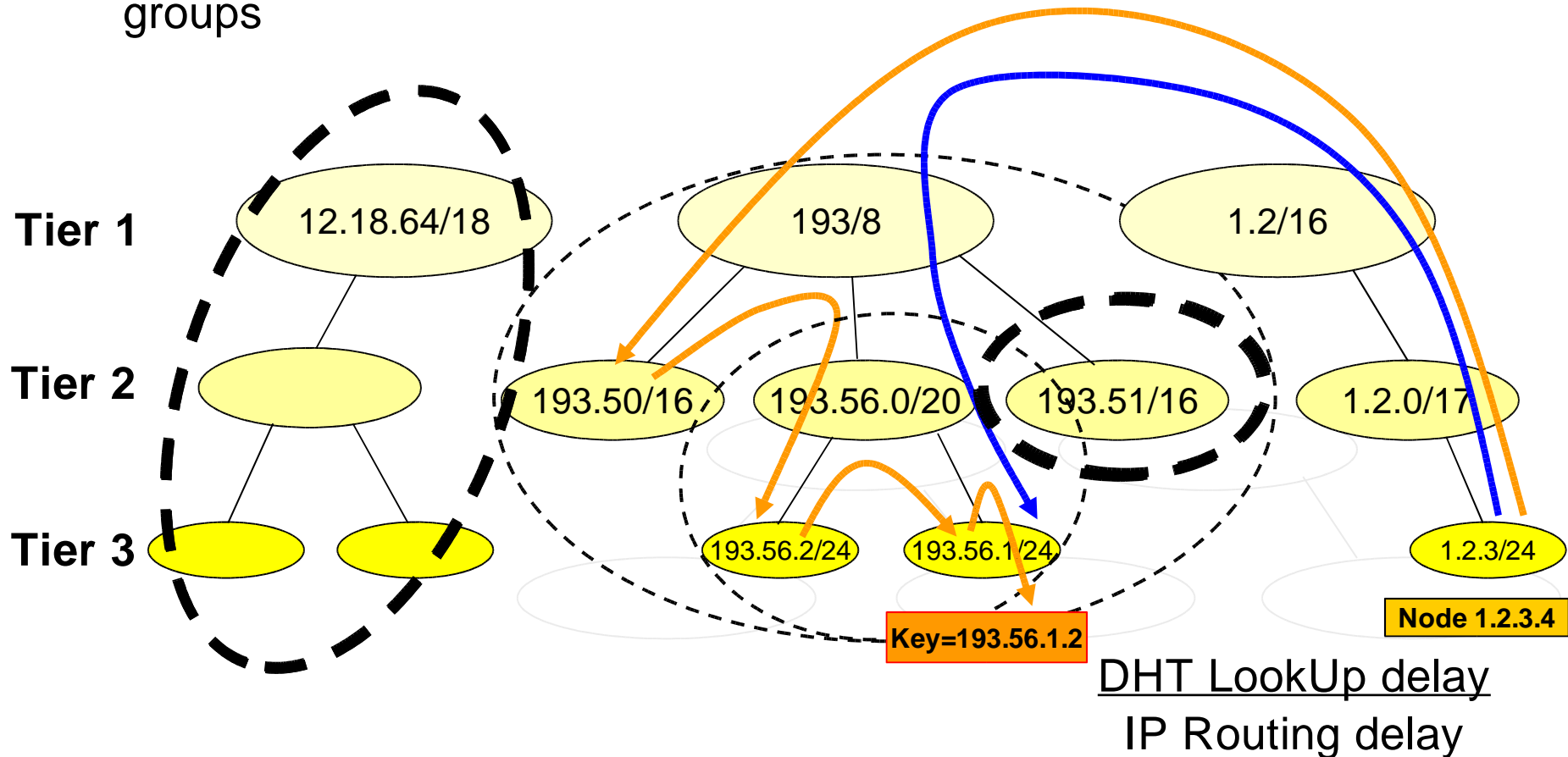
TOPLUS

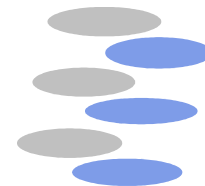


Node ID = IP@ of node

IP Prefix ranges define (nested) groups

Every node needs to know a “**delegate**” in (some of) the other groups

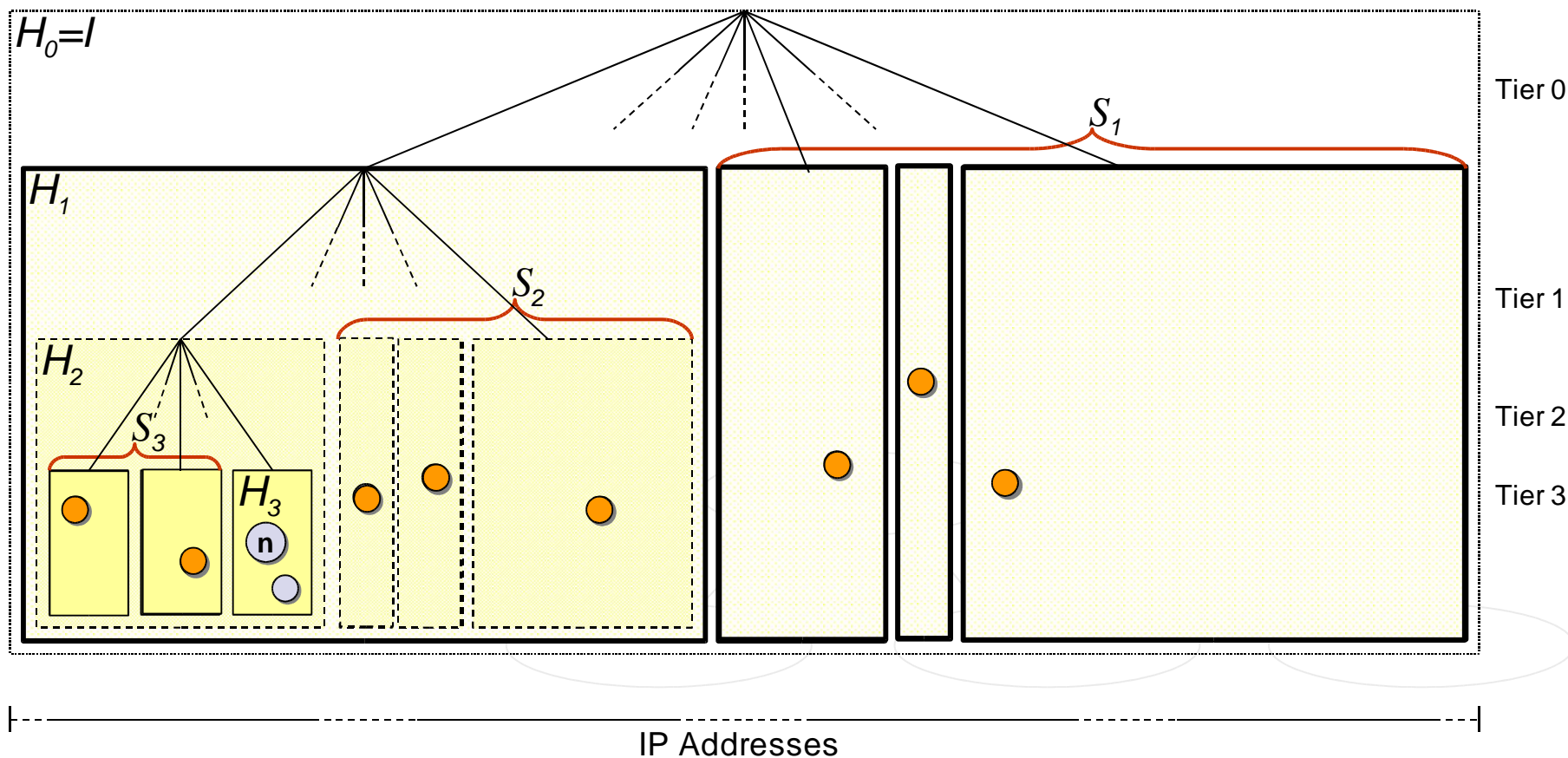




Node State

More detail about closer groups

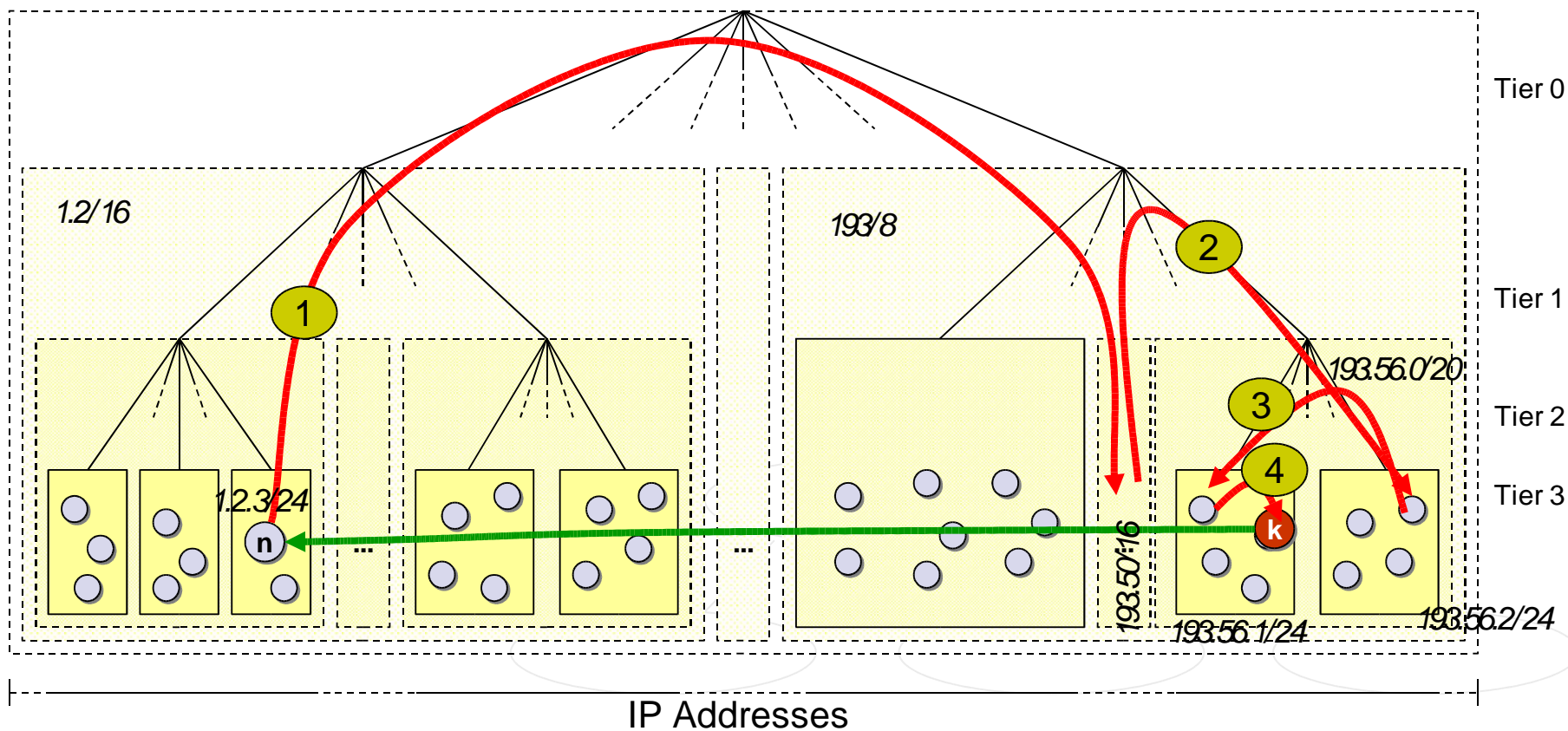
Coarser knowledge about groups far away



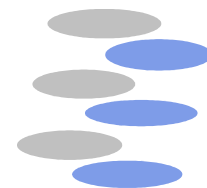


Key Look-Up (I)

Node $n=1.2.3.4$ looks-up key $k=193.56.1.2$



Number of hops $\leq H+1$, $H = \text{height of tree}$



Key Look-Up (II)

Each key k is a bit string of length $m > 32$:

First 32-bits used for routing to node responsible for that key

XOR metric:

Let node $j = j_{31}j_{30}\dots j_0$ and $k = k_{31}k_{30}\dots k_0$:
$$d(j, k) = \sum_{i=0}^{31} |j_i - k_i| \cdot 2^i$$

Note that closest ID is unique:

$$d(j, k) = d(j', k) \Leftrightarrow j = j'$$

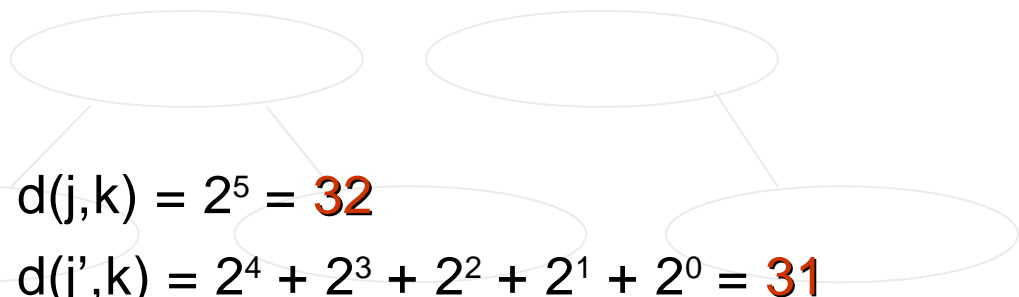
Refinement of longest-prefix match

Example (8 bits)

$k = 10010110$

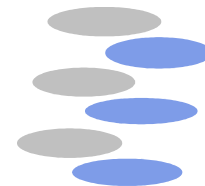
$j = 10110110$

$j' = 10001001$



$d(j, k) = 2^5 = 32$

$d(j', k) = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 31$



Key Look-Up (III)

XOR metric solves “black hole” problem

Example (32 bit):

Key k = 11100011110101001011100101011001

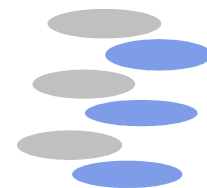
Node j = 11100011110101001011100101011100

Node j' = 11100011110101001011100101011101

Simple longest-prefix matching cannot decide

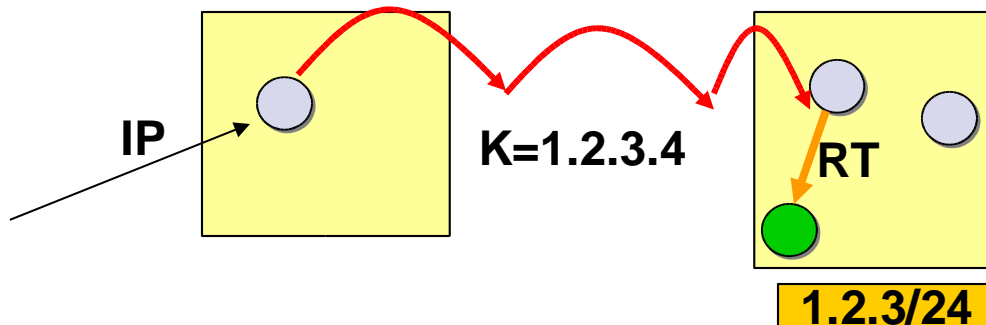
However $d(j,k)=101 > d(j',k)=100$

node j' responsible for key k



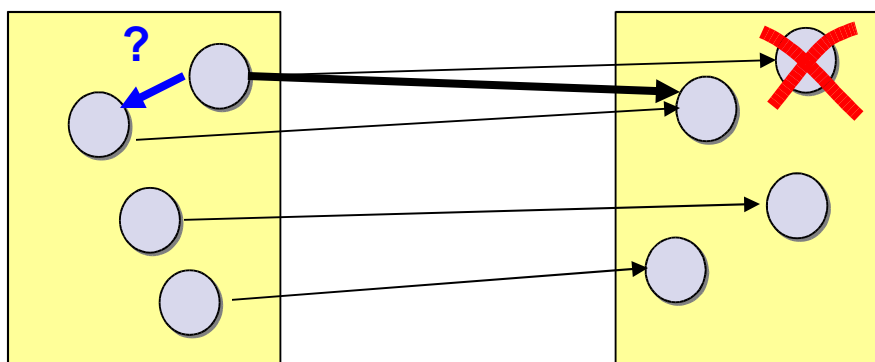
Overlay Maintenance

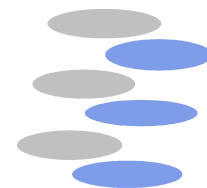
A new node **n** joining the TOPLUS network:



Delegate diversity property

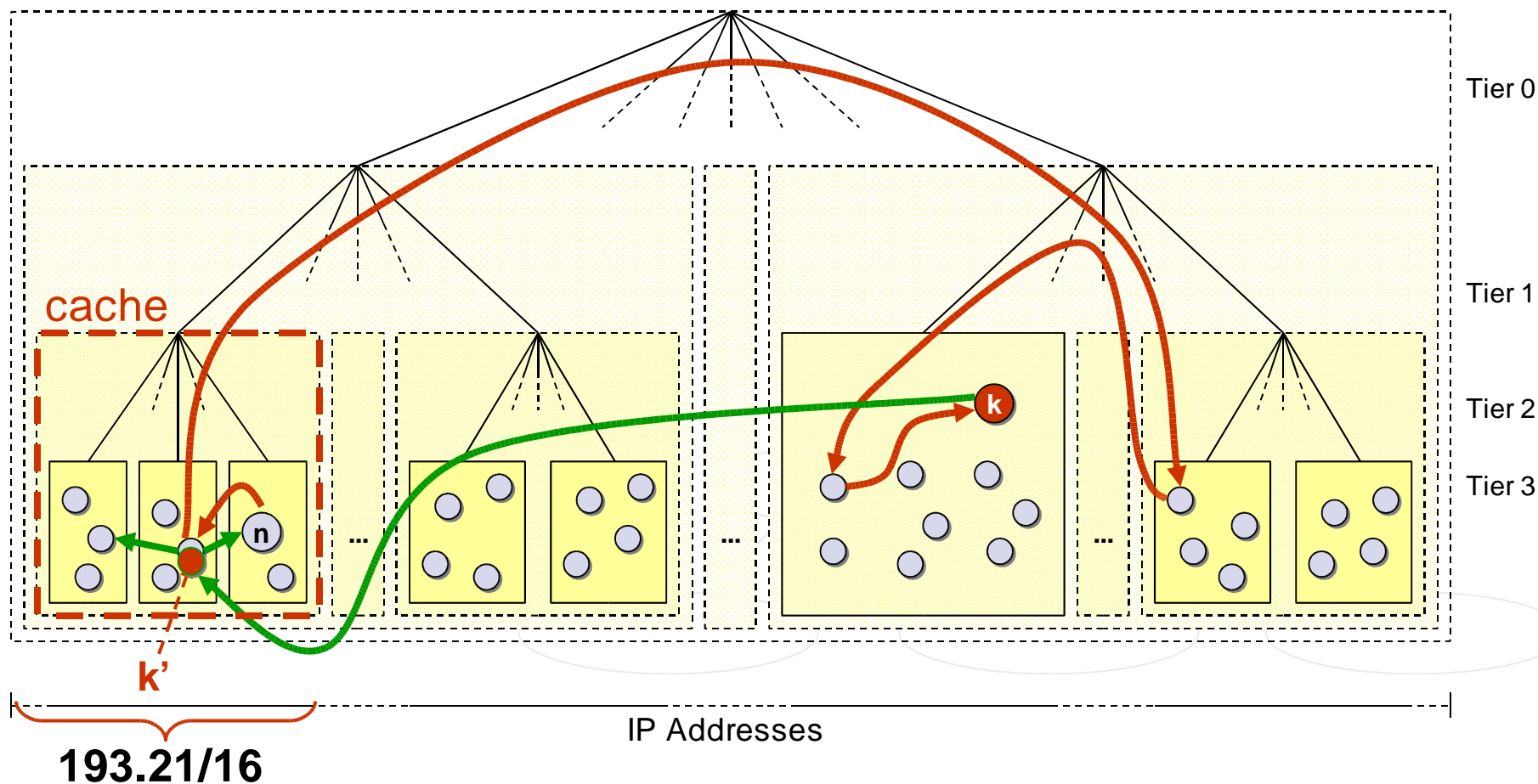
Robust links between groups, **no DHT reconstruction** when node leaves

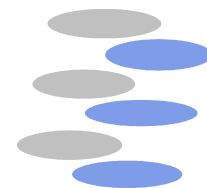




On-Demand Caching

$k = 212.17.89.100$ $\xrightarrow{193.21/16}$ $k' = 193.21.89.100$





Where to get the IP prefixes from?

Use to create the TOPLUS hierarchy:

We obtain **250,000** IP prefixes from

BGP tables (Oregon U., Michigan U. (Merit Network))

Routing registries (RIPE, Castify Networks)

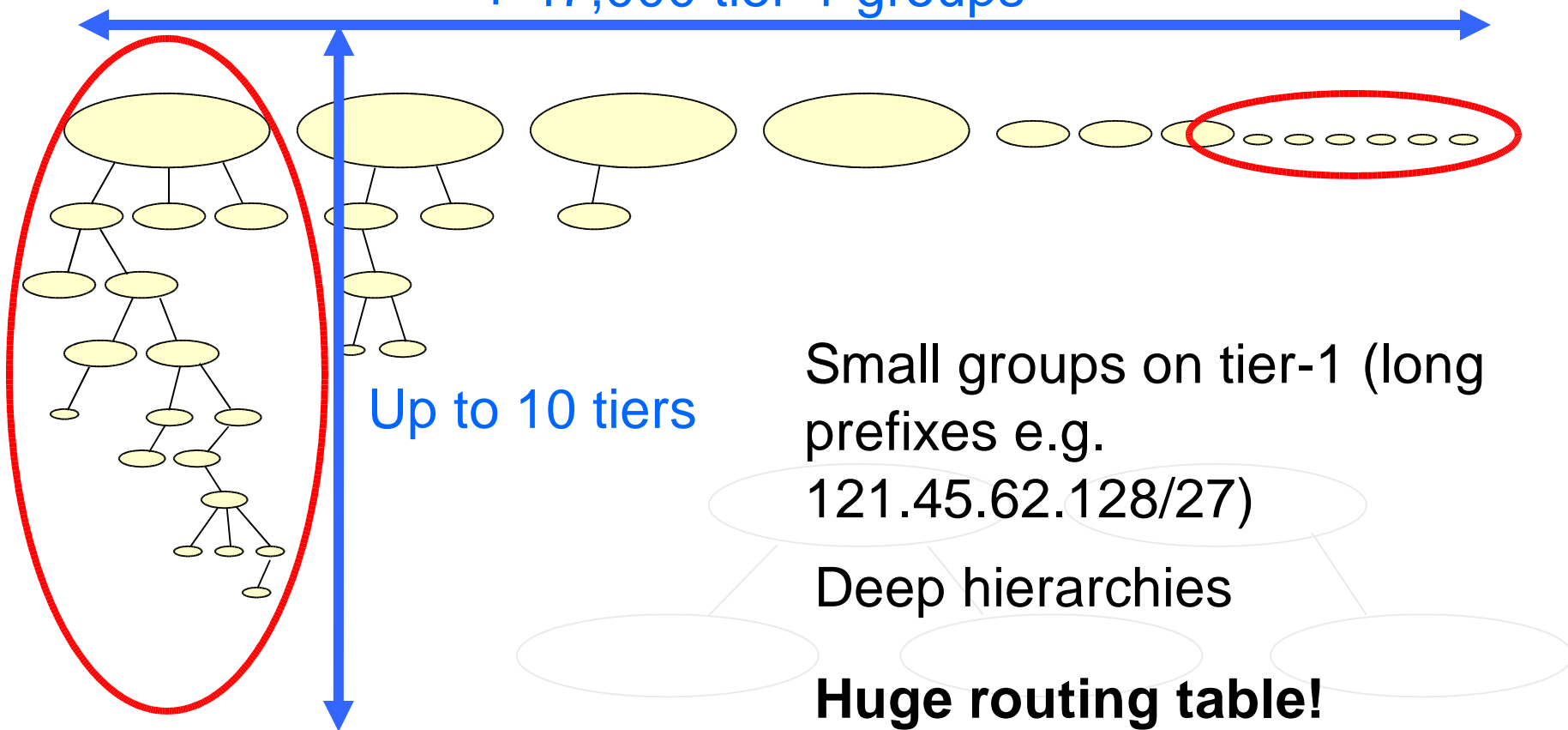




Partial Order Tree Construction (I)

Very wide; very unbalanced

+ 47,000 tier-1 groups

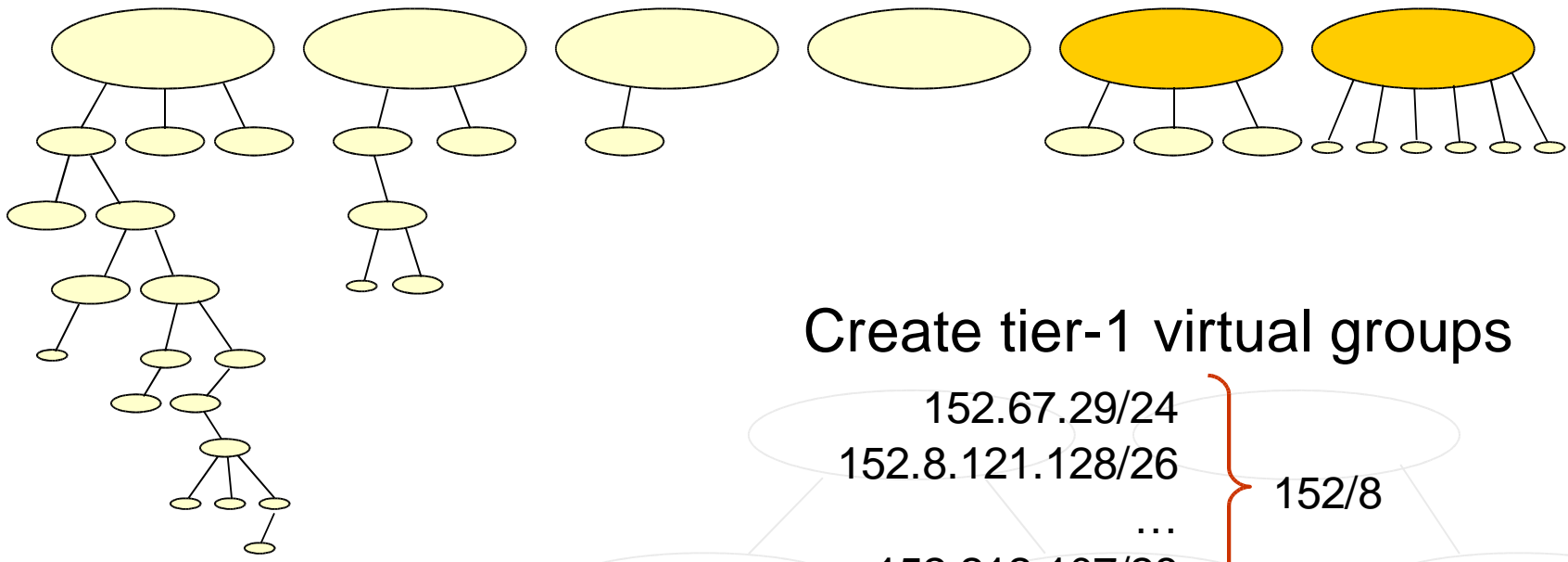




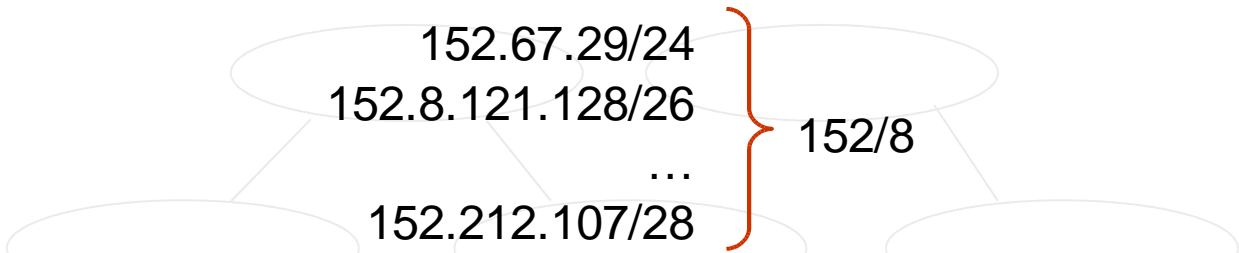
Partial Order Tree Construction (II)

Modified trees:

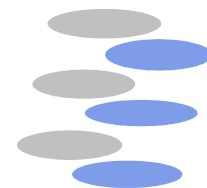
- Reduce routing table size
- Respect topology as possible



Create tier-1 virtual groups



Add a level to the hierarchy



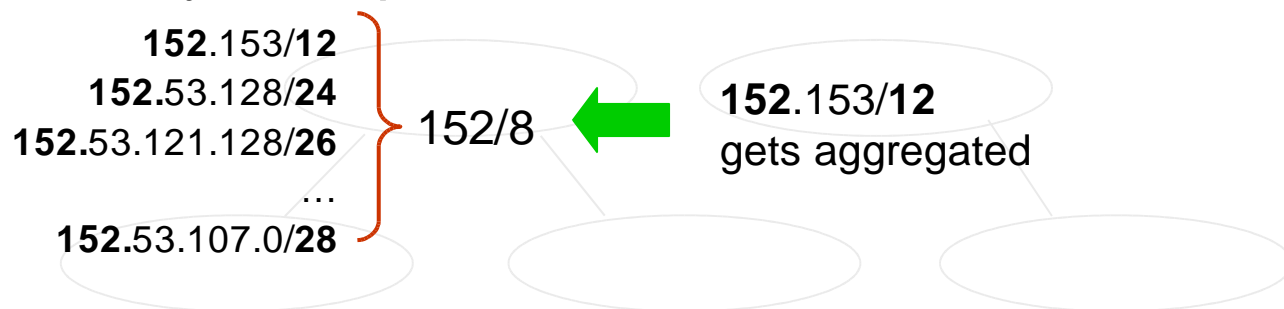
Tested Trees

Original: direct hierarchy obtained from IP prefixes

8-bit: Only prefixes from 8 to 16 bit on tier-1



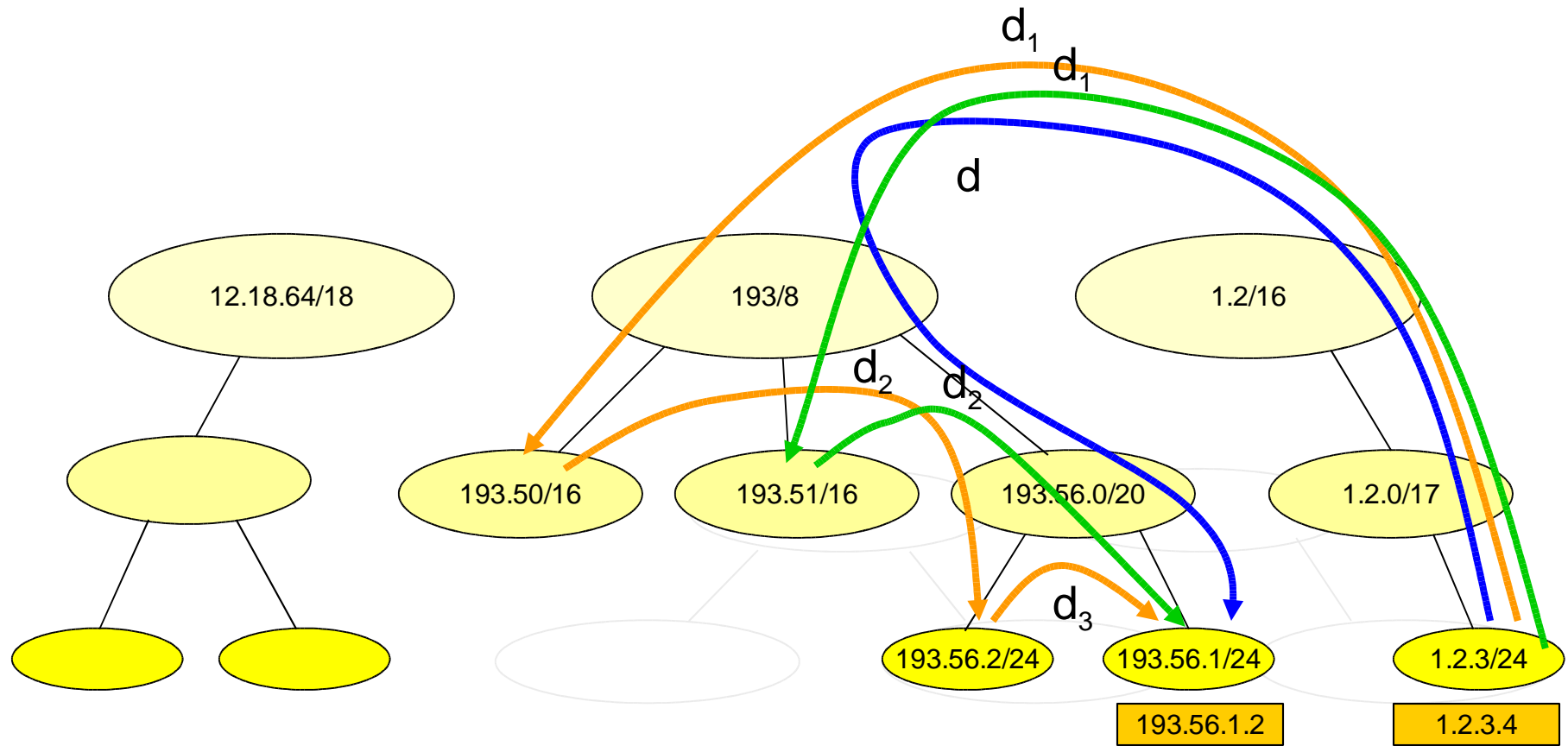
Original+1: Allow only 8-bit prefix on tier-1

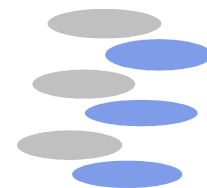


3-Tier: three fixed tiers, 8-, 16- and 24-bit



Benchmarking: Stretch



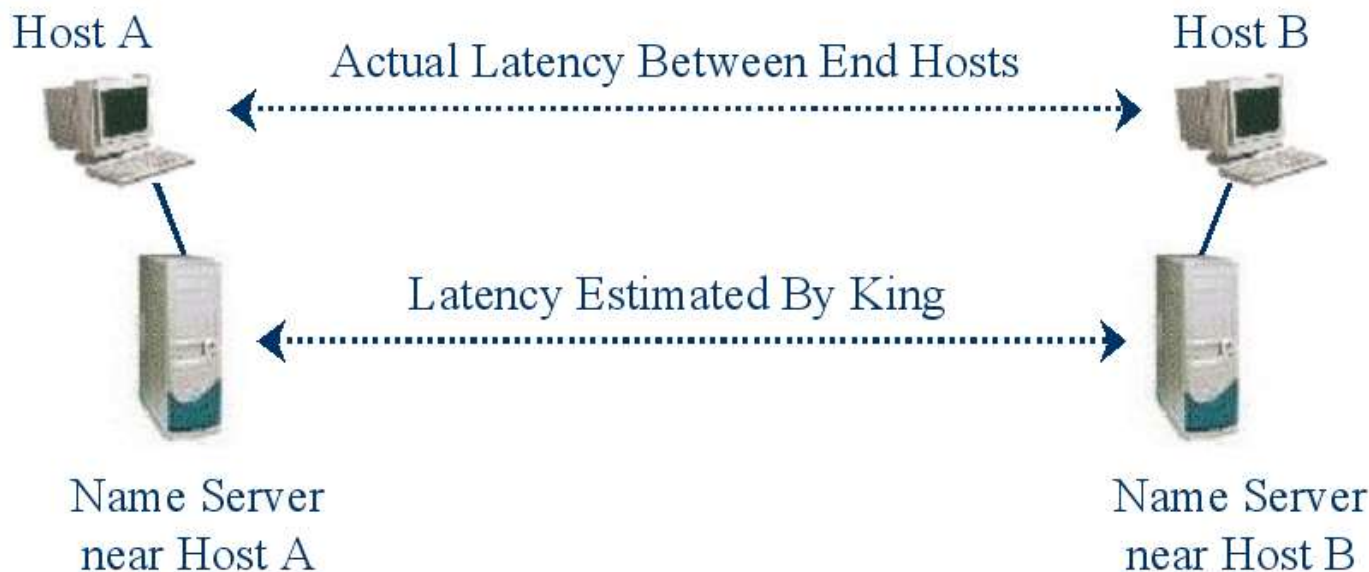


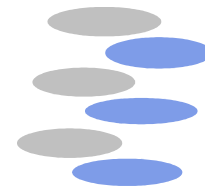
King

From University of Washington, Seattle

Given 2 IP addresses, measures distances between corresponding DNS servers

It can at least give an ***estimate*** of delay between two nodes





Benchmarking: Measurements

Stretch:

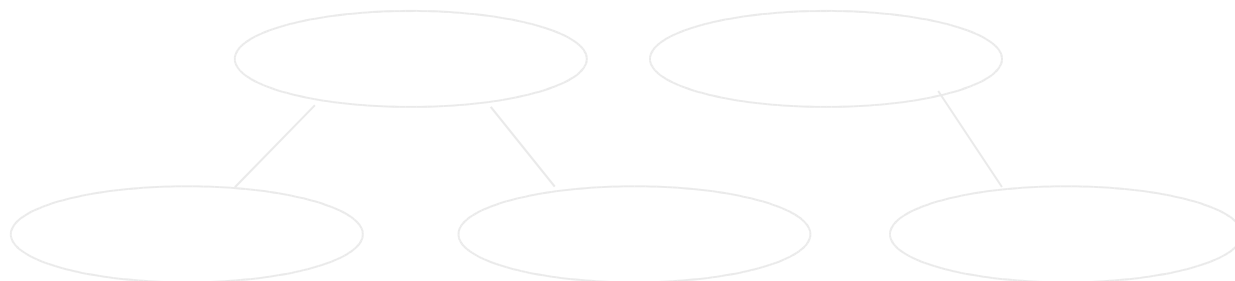
Measured from a node in our site to **1,000** random valid destinations

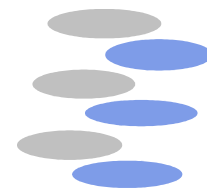
Routing table size:

Averaged over **5,000** random nodes

Observe DHT trade-off:

Stretch vs. Routing Table size



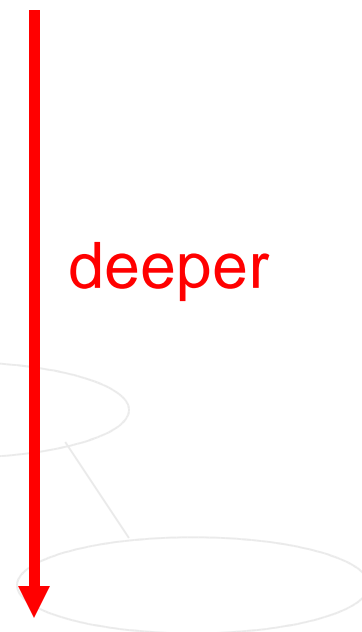


Benchmarking: Stretch (II)

Average Stretch **per tier** :

The deeper in the hierarchy, the larger the stretch

Stretch		
Tier	Original	8-bit
1	1.00	1.00
2	1.29	1.53
3	1.31	1.56
4	1.57	1.58





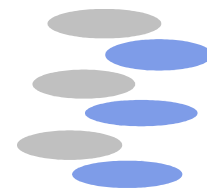
Benchmarking: Routing Table

Routing Table size: Average of 5,000 random peers

Routing table size				
Tier	Original	8-bit	Original+1	3-Tier
1	47,467	8,593	143	143
2	47,565	8,713	436	248
3	47,654	8,821	831	261
4	47,796	8,950	1,279	-
5	47,890	9,016	696	-

Very large tier-1

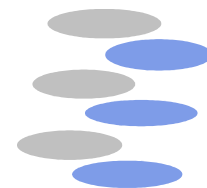
8-bit Prefix tier-1



Benchmarking: Stretch vs R.Table Size

Inversely proportional

	Stretch	RT Size
Original	1.17	47,547
8-bit	1.28	8,699
Original+1	1.90	656
3-Tier	2.32	211



Conclusion

Topology-centric DHT design

Features

- Small stretch

- Fast XOR-based routing

- Natural On-demand P2P caching hierarchy

- Static deployment very easy

Open Issues

- Non-uniform population of ID space

- Correlated node failures

