YAPPERS :A Peer-to-Peer Lookup Service over Arbitrary Topology

IEEE INFOCOM 2003

謝志峰

2003/5/29

Outline

- Introduction
- YAPPERS
- The Immediate Neighborhood
- The Extended Neighborhood
- Maintaining Topology
- Enhancements
- Evaluation
- Conclusions

Introduction(1/4)

- There are two classes of solutions currently proposed for decentralized peer-to-peer content location:

  - **Gnutella** : It is relies on **flooding** queries to all peers.Peers organize into an overlay. To find content, a peer sends a query to its neighbors on the overlay.The neighbors forward the query on to all of their neighbors until the query has traveled a certain radius.

  - **Distributed Hash Table(DHT)**:Peers organize into a well-defined structure that is used for routing queries.

Introduction (2/4)

- YAPPERS(Yet Another Peer-to-Peer System)
- Four main design goals in designing our lookup service are:
  - 1) impose **no constraints** on topology.
  - 2) **optimize for partial lookups** where there are many values for a key and clients are satisfied by a small subset of the values.
  - 3) contact only nodes that can contribute to the search result rather than flooding blindly.
  - 4) minimize the effect of topology changes so that the maintenance overhead is independent of the overlay size.

Introduction (3/4)

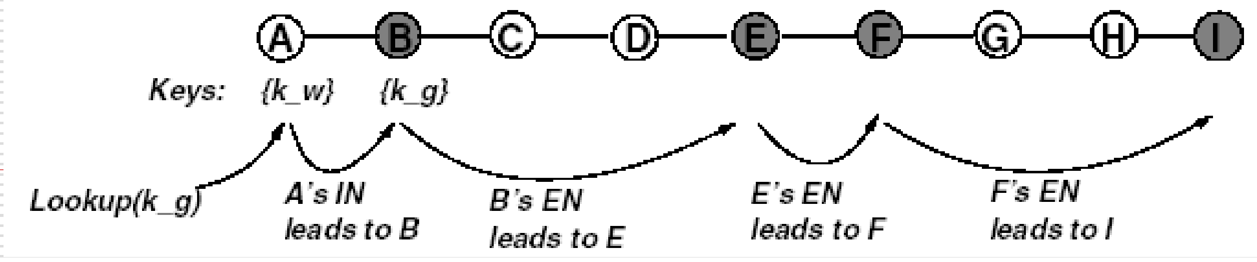
- We assume the following model for each of the peer nodes:
  - When a node is created, there is a third-party network layer that determines a set of live nodes as its new neighbors in the overlay.
  - Each node "owns" a (possibly empty) set of ***key, value*** pairs. When the node joins the network, it registers these pairs with the lookup service. The node may choose to register additional pairs with the service, or delete some registered pairs at any point in time.
  - A node may initiate either **partial-lookup** or total-lookup queries at any point in time.
  - When a node leaves the system, the *key*, *value* pairs that it "owns" do not need to be preserved in the system.
  - A node may establish connections to other nodes directly, even if they are not neighbors in the overlay network.

Introduction (4/4)

- With these assumptions, let S be the set of *key*, *value* pairs registered with the lookup service. We define:
  - $TotalLookup(N, k)$ as the set of values returned by the service for a total-lookup query on key k originating at node N.
  - $PartialLookup(N, k, n)$ as the set of n values returned by the service for a partial-lookup query on key k originating at node N. If the total number of values for k is less than n, the partial lookup is defined to be equivalent to the total lookup.
  - $V\ alues(k) = \{v|k, v\ \ S\}$
- Then the lookup service is correct if and only if:
  - 1) $TotalLookup(N, k) = V\ alues(k)$ for all nodes N and keys k.
  - 2) $PartialLookup(N, k, n)\ \ V\ alues(k)$ and $|PartialLookup(N, k, n)| \geq \min(n, |V\ alues(k)|)$ for all nodes N, keys k and integers n.

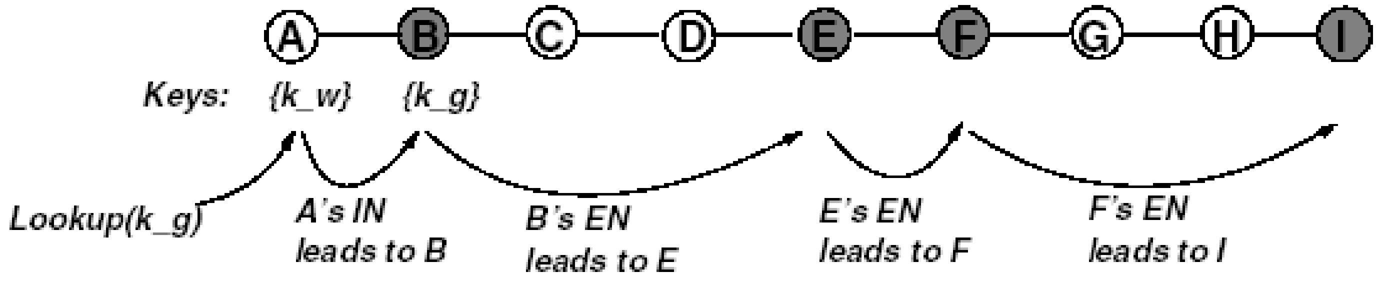
YAPPERS (1/3)

- The keyspace of all the keys that need to be stored is partitioned into a small number of buckets.
- For ease of exposition, consider an example where the keyspace is divided into two buckets.
- Let us say that keys are either **white** or **gray**. Every node in the network is also assigned a color, white or gray, based on some criterion.
- A query for a gray key needs to be forwarded only to gray nodes in the network, and a query for a white key only to the white nodes.



YAPPERS (2/3)

- To guarantee that gray queries end up being forwarded to all the gray nodes, we require that each node knows all nodes within 3 hops of it, and forwards a gray query to all the gray nodes it knows about in this 3-hop radius.
- More generally, if we permit a node to store a *key, value* pair at an appropriately-colored node within *h* hops of it , it is sufficient for a node to know all its neighbors within (2*h* + 1) hops in order to guarantee that we can exhaustively hop through all the gray nodes without touching any white node.
- We call the nodes within ***h* hops the *immediate neighborhood*** of a node, and the nodes within **2*h* + 1 hops the *extended neighborhood*** of the node.



YAPPERS (3/3)

- YAPPERS divides a large overlay network into many small and overlapping neighborhoods (the immediate neighborhoods).

- The data within each neighborhood is partitioned among the neighbors like a distributed hash table.

- When a lookup occurs and the neighborhood cannot satisfy the request, YAPPERS intelligently forwards the request to nearby neighborhoods, or the entire network if necessary.

- These forwardings require each node to know a larger set of nodes (the extended neighborhood) that covers its neighbors’ neighbors.

The Immediate Neighborhood (1/3)

- The immediate neighborhood of a node *A*, denoted by *IN*(*A*), is the set of nodes where *A* may store its *key*, *value* pairs.
- Moreover, the solution must strive to maintain the following two useful characteristics:
  - *Consistency*: If a node *X* is in two different neighborhoods *IN*(*A*) and *IN*(*B*), both *A* and *B* assign the same color to node *X*.
  - *Stability*: If a node *X* is in *IN*(*A*), then *X* is assigned the same color regardless how *IN*(*A*) changes dynamically when nodes enter or leave the system.
- Both characteristics are desirable because they improve the overall system efficiency.
- Consistency avoids costly synchronizations among nearby nodes to determine which nodes have which colors.
- Stability reduces data relocation when the underlying overlay network changes.

The Immediate Neighborhood (2/3)

- Given a node *A*, which nodes should be included in *IN*(*A*)?
- $IN(A) = \{v \mid G(v,A)\ \ h\}$ where *G* returns the minimum distance between two nodes in *G*. In other words, *IN*(*A*) contains all nodes within *h* hops of *A* in the overlay network, including node *A* itself.
- Specifically, our YAPPERS implementation uses $h = 2$.
- We chose a small immediate neighborhood in order to provide long-term stability to the system.
- If the immediate neighborhood is large, then frequent node arrivals and departures within the network will incur large overheads in maintaining an accurate view of the immediate neighborhood and reduce the efficiency of searches when the view is incorrect.

The Immediate Neighborhood (3/3)

- Given *IN*(*A*), how do we partition the key space into multiple colors and assign each color to nodes in *IN*(*A*)?
- a node *X* with IP address *IPX* is assigned key *k* if *HASH*(*k*) ≡ (*HASH*(*IPX*) mod *b*) where *b* is the number of distinct hash buckets we use.

- In other words, the keyspace is divided into *b* hash buckets, or *b* different colors *C*0, *C*1, . . . , *Cb*-1.
- We say that a key *k* is of color *C* (or hashes to color *C*) if the hash function assigns *k* to bucket *C*. If a node IP address hashes to bucket *C*, we say the node is of color *C*.

The Extended Neighborhood (1/6)

- Flooding the overlay network like Gnutella is a solution.
- However,such flooding disturbs many nodes that do not actually have any answers for the search.
- To avoid these disturbance, a node A keeps track of a bigger neighborhood than its immediate neighborhood so that it can make bigger “jump” and avoid flooding its direct neighbors.

The Extended Neighborhood (2/6)

- we first define the *frontier* of node *A*, denoted by *F*(*A*), as all nodes that are not in *IN*(*A*) but are directly connected to a node in *IN*(*A*). Formally, if *N*(*v*) is the set of nodes directly connected to node *v*, then

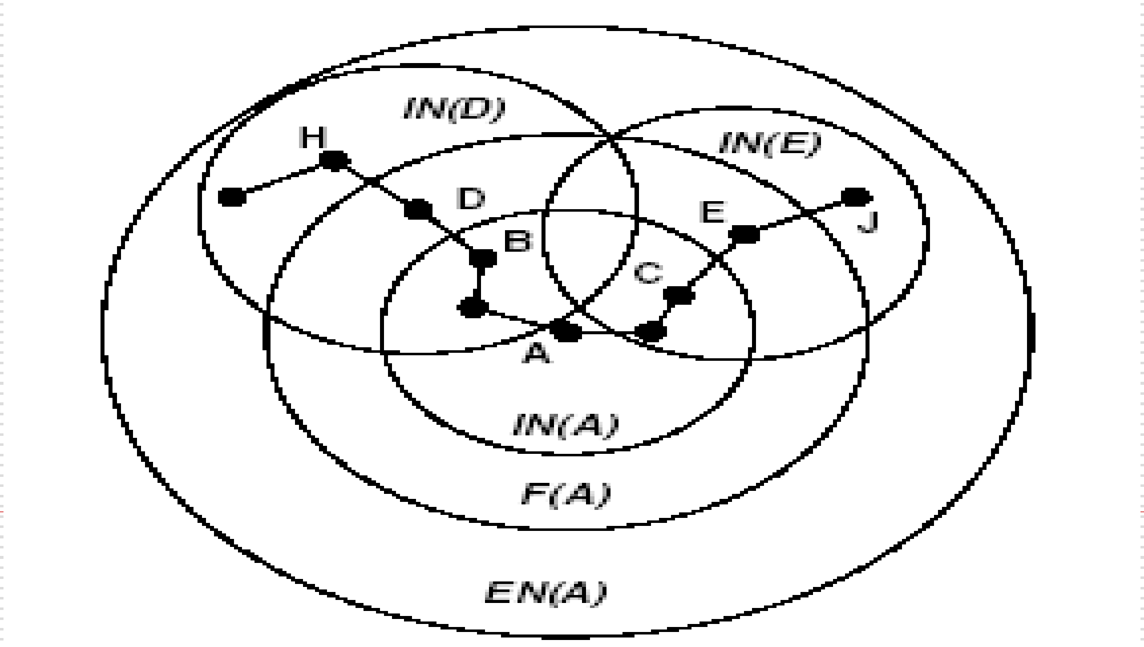
$$F(A) = \bigcup_{\upsilon \in IN(A)} N(\upsilon) - IN(A)$$

- With the frontier, the extended neighborhood *EN*(*A*) is then simply the union of the immediate neighborhoods of all nodes in the frontier of *A*. Formally,

$$EN(A) = \bigcup_{\upsilon \in F(A)} IN(\upsilon)$$

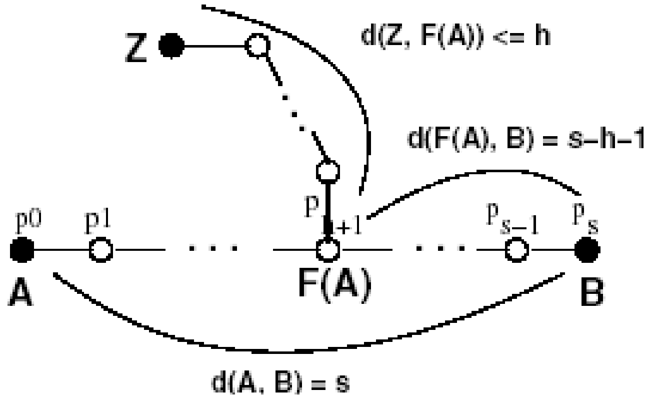
The Extended Neighborhood (3/6)

- In this figure, $h = 2$.
- So nodes ***B*** and ***C*** are in *IN*(*A*).
- Nodes ***D*** and ***E*** are in the *F*(*A*) because they are connected to *B* and *C* respectively.
- Therefore, *EN*(*A*) includes ***H*** (part of *IN*(*D*)) and ***J*** (part of *IN*(*E*)).



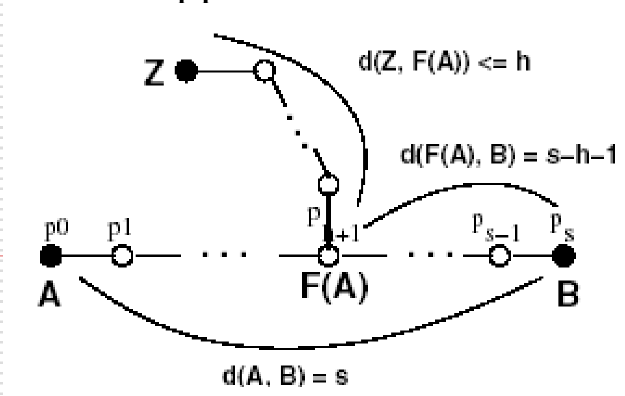
The Extended Neighborhood (4/6)

- *Theorem 1:* (Completeness) For any two nodes A and B of color C, there exists a sequence of nodes $A = Z0, Z1, Z2, \ldots, Zw\text{-}1, Zw = B$ such that for all $i < w$, Zi has color C and Zi forwards the request to $Zi+1$ when executing the *forward* routine.
- Assuming two black nodes A and B are the closest nodes that cannot reach each other and is at least $h + 1$ hops away, we can construct another pair Z and B where Z is even closer to B and cannot reach B. hops while not knowing about each other.



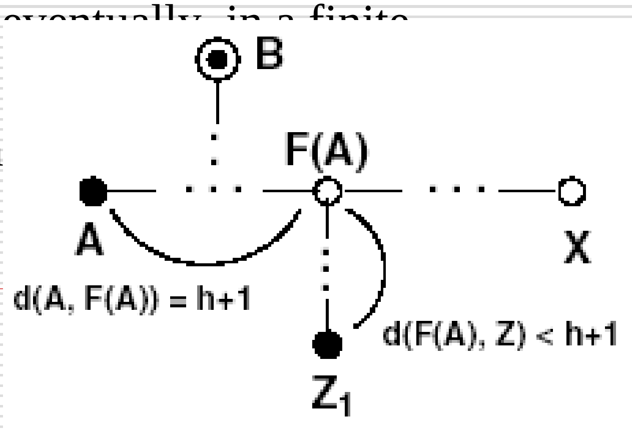
The Extended Neighborhood (5/6)

- Now consider the shortest overlay network path $p0, p1, \ldots, ps$ from A to B where $p0 = A$ and $ps = B$. There are two cases: $s \geq h + 1$ and $s < h + 1$. We show that both cases lead to contradictions, hence prove our claim.
- *Case 1: $s \geq h + 1$.*
- Now consider $\delta(Z,B)$, the distance between Z and B.
- By the triangle inequality, $\delta(Z,B) \leq \delta(Z, F(A)) + \delta(F(A),B)$.
- By construction, we know $\delta(Z, F(A)) \leq h$ and $\delta(F(A),B) = s\text{-}h\text{-}1$.
- Therefore, $\delta(Z,B) \leq h+s-h-1 = s-1$, which means Z is closer to B than A and is a contradiction to our choice that A and B are the closest pair of black nodes that do not satisfy our claim. Hence, the case $s \geq h + 1$ cannot happen.



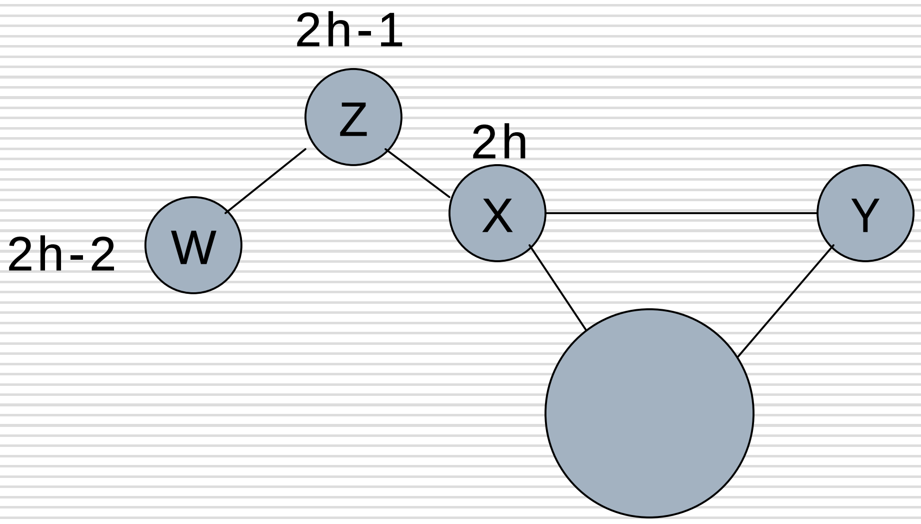
The Extended Neighborhood (6/6)

- *Case 2: s* $< h+1$.
- Let $t = \delta(A,X)$, the distance between A and X.
- If $t \leq h + 1$, then $X \in IN(A) \cup F(A)$ and node A would have learned that node B is also black when determining where X stores black keys.
- For the same reason as $t > h + 1$, $(Z1,X) > h + 1$.
- However, $\delta(Z1,X) \leq \delta(Z1, F(A)) + \delta(F(A),X) \leq h + (t - h - 1) = t - 1 < t = \delta(A,X)$.
- Therefore if we repeat the step above (with A replaced by $Z1$), we can find $Z2$ such that $Z1$ forwards the request to $Z2$ and $\delta(Z2,X) \leq t - 2$.
- Since each step brings us at least one node closer to X, eventually, in a finite number of steps $i \leq t - h - 1$, we get $\delta(Zi,X) \leq h+1$.
- When this happens, $X \in IN(Zi) \cup F(Zi)$ which means n B is a black node and forward B the lookup request.



Maintaining Topology (1/2)

- Edge Deletion: YAPPERS uses a 2*h*+1 hops extended neighborhood, an edge deletion requires both *X* and *Y* to broadcast the deletion event to its surviving neighbors with a time-to-live field of 2*h* hops.
- Edge Insertion:The solution is to do the same thing as edge deletion but append both *X*'s and *Y* 's new extended neighborhood information in the broadcast.



Maintaining Topology (2/2)

- **Node Departure and Arrival**:
  - As node *X* appears on the network, it first asks its new neighbors for their current views of the topology.
  - Node *X* then merges these views to create its own extended neighborhood.
  - Afterwards, node *X* initiates an edge insertion broadcast to each of its new neighbors appended with the appropriately trimmed subset of the new topology.

Enhancements (1/3)

- We noticed two problems: the *fringe node* problem and the *large fan-out* problem.
- In the *fringe node problem*, a low connectivity node, through the use of backup assignment, allocates a large number of secondary colors to its high-connectivity neighbor which has no desire for the extra colors.
- One obvious solution to the *fringe node* problem is to prune away low connectivity nodes.

Enhancements (2/3)

- the *large fan-out* problem:
- When a node *A* does the forwarding, each of *A*'s frontier nodes may point to one or more different forwarding nodes.
- Consequently, the forwarding fan-out degree at node *A* is proportional to the number of the frontier nodes.

Enhancements (3/3)

- the *large fan-out* problem:
- When node *A* forwards a lookup request for color *C*,
  - 1) If a frontier node *F* assigns *C* to node *B* via the backup mechanism, then forward the request to *B*.
  - 2) If a frontier node *F* assigns *C* to a set of nodes *S*, do not forward to any nodes in *S* if *SIN*(*A*) = . Otherwise, only forward to one of the nodes in *S*.
  - 3) In step (2), when choosing one node from *S*, try to pick common nodes between multiple frontier nodes.
- Step (1) is necessary because the only way of reaching a backup node *B* could be through the frontier node *F*.
- If no backups are necessary, then steps (2) and (3) try to avoid forwarding to far-away nodes if a closer one exists.

Evaluation (1/7)

- To estimate statistics on YAPPERS running over a real overlay network, we simulated YAPPERS on a snapshot of the Gnutella network [15] containing 24, 702 connected nodes and several synthetically-generated regular graphs.

- With our implementation, we examine the search efficiency with respect to the expected fraction of nodes contacted per query, the search overhead in terms of the fan-out degree for forwarding search messages, and the optimal *b* (number of hash buckets) to use.

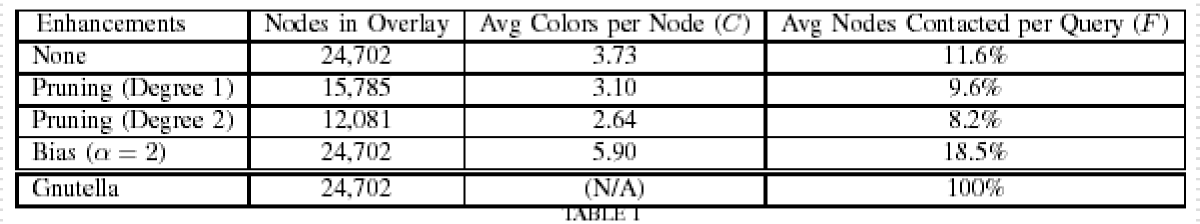
Evaluation (2/7)

- The efficiency of executing a total-lookup request is captured in the expected fraction of nodes contacted, denoted by , $\overline{F}$ during the search. This fraction $\overline{F}$ is equal to $\frac{\overline{C}}{b}$, where $\overline{C}$

  is the average number of colors assigned to each node and *b* is the number of hash buckets (colors) used.
- To see this, notice that $N \cdot \overline{C} = \left(N \cdot \overline{F}\right) \cdot b$ because $N \cdot \overline{C}$ is the total number of colors in the system, $\left(N \cdot \overline{F}\right)$ counts the number of nodes
- having a particular color, and hence $\left(N \cdot \overline{F}\right) \cdot b$ also counts the number of colors in the system.

Evaluation (3/7)

- Table I illustrates how different enhancements of YAPPERS affect $\overline{C}$ and $\overline{F}$ when running over the Gnutella snapshot with *b* = 32.
- The last line in the table provides the baseline comparison of running straight Gnutella.
- From the table, notice that to execute a total lookup, Gnutella has to contact every node.
- In contrast, YAPPERS only contacts 8% to 18% of the nodes depending on the enhancement.



Enhancements	Nodes in Overlay	Avg Colors per Node ($\overline{C}$)	Avg Nodes Contacted per Query ($\overline{F}$)
None	24,702	3.73	11.6%
Pruning (Degree 1)	15,785	3.10	9.6%
Pruning (Degree 2)	12,081	2.64	8.2%
Bias ($\alpha = 2$)	24,702	5.90	18.5%
Gnutella	24,702	(N/A)	100%

TABLE I

Evaluation (4/7)

- For completeness, we show the cumulative distribution of colors per node for YAPPERS in Figure 7.
- The x-axis shows the number of colors assigned to a node and the y-axis shows the percentage of nodes with equal or fewer colors.
- Pruning indeed reduces the number of colors on the high degree nodes as we reach the 100 percentile at 11 colors per node with pruning as opposed to 27 without.

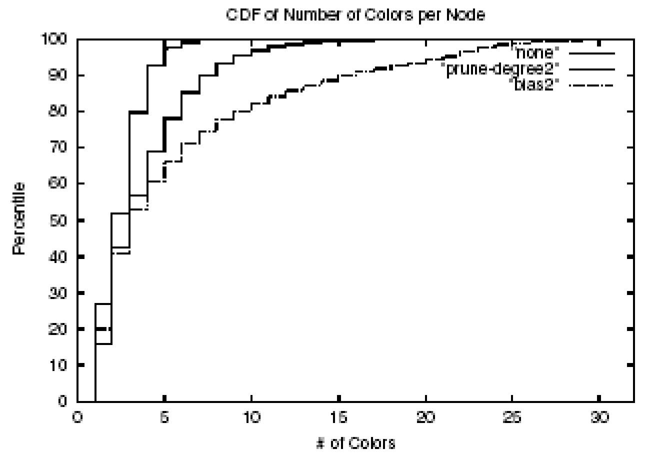
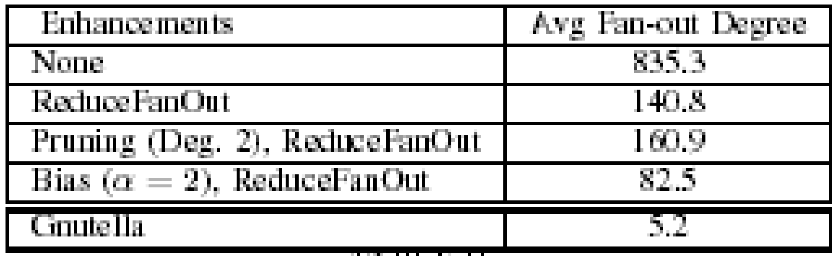


Fig. 7. Cumulative distribution plot of number of buckets per node for YAPPERS with $b = 32$ and no enhancements running on Gnutella snapshot.

Evaluation (5/7)

- Table II shows the average fan-out degree for YAPPERS with various enhancements.
- As we apply the fan-out reduction techniques and avoid overloading high connectivity nodes with extra colors, the fan-out is reduced to 82.
- Large fan-out degree has both positive and negative impact on performance. On one hand, lookups are propagated much faster through the network in YAPPERS. On the other hand, we need extra state information to keep track of these nodes in the extended neighborhood and connection states (if any).



Enhancements	Avg Fan-out Degree
None	835.3
ReduceFanOut	140.8
Pruning (Deg. 2), ReduceFanOut	160.9
Bias ($\alpha = 2$), ReduceFanOut	82.5
Gnutella	5.2

TABLE II

Evaluation (6/7)

- The natural question is: how many hash buckets (colors) is ideal for YAPPERS?
- First, Figure 8 shows the search efficiency where we have the fraction of nodes contacted during a lookup on the yaxis and the number of hash buckets, *b*, used on the x-axis.

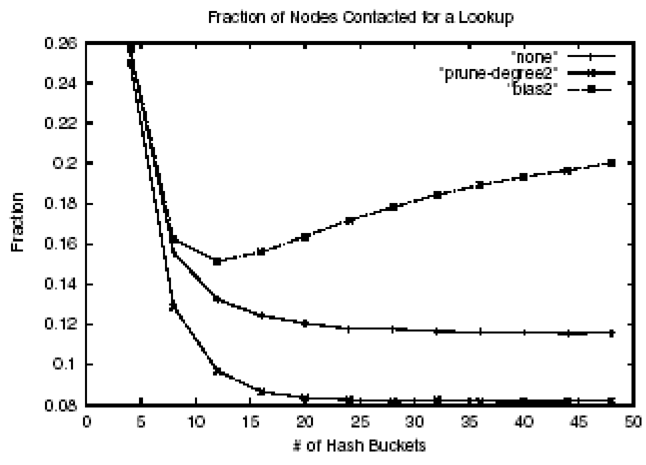


Fig. 8. Fraction of nodes contacted during a lookup

Evaluation (7/7)

- Figure 9 shows the overhead in terms of the average forwarding fan-out degree per node per hash bucket (color).
- The y-axis shows the average degree and the x-axis shows the number of hash buckets, *b*, used.
- We see that *b* = 16 is the sweet spot for this Gnutella snapshot.

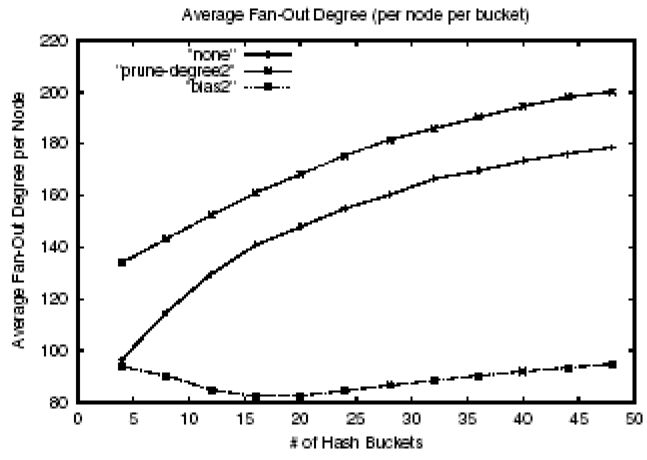


Fig. 9. Average fan-out degree per node on a lookup

Conclusions

- Our scheme is a hybrid that uses distributed hash tables (DHTs) in small areas and uses intelligent forwarding over large areas.
- The main advantages are that our scheme:
  - disturbs only a small fraction of the nodes in the overlay for each search.
  - does not require restructuring the underlying overlay network.
  - each node requires only knowledge of a small neighborhood and is independent of the rest of the overlay, and is thus less affected by node arrivals and departures.