Improving Peer to Peer Search With Multi-Tier Capability-Aware Overlay Topologies

Mudhakar Srivatsa, Bugra Gedik, Ling Liu
College of Computing
Georgia Institute of Technology
{mudhakar, bgedik, lingliu}@cc.gatech.edu

Abstract. The P2P model has many potential advantages (e.g., large scale, fault-tolerance, low cost of administration and maintenance) due to the design flexibility of overlay networks and the decentralized management of cooperative sharing of information and resources. However, the mismatch between the randomly constructed overlay network topology (combined with its broadcast-style message forwarding infrastructure) and the underlying packet routing introduces difficult performance problems, exemplified by the *Short-Cut Effect*. This paper presents two peer-to-peer (P2P) system-level facilities to address the problems. First, we propose a capability-aware mechanism to structure the overlay topology in the form of layers that takes peer heterogeneity into account. Second, we develop a *Probabilistic Broadening* search technique, empowered with capability-sensitive query forwarding scheme which integrates gracefully with result caching techniques to improve the search performance of a P2P system. We believe that efforts on bridging the gap (mismatch) between overlay networks and underlying Internet will bring P2P services beyond pure "best effort" and closer to serious applications with quality of service requirements.

1 Introduction

With applications such as Gnutella and Freenet, the peer-to-peer (P2P) model is quickly emerging as a significant computing paradigm of the future Internet. Unlike traditional distributed computing, P2P networks aggregate large number of computers and possibly mobile or hand-held devices, which join and leave the network frequently. This new breed of systems creates application-level virtual networks with their own overlay topology and routing protocols. The overlay topology provides mechanisms to create and maintain the connectivity of an individual peer (node) to the network by establishing neighbor relationship with a subset of other nodes (neighbors) in the overlay network. The P2P routing protocols allow individual computers and devices to share information and resources directly, without dedicated servers. Although P2P networking technologies may provide some desirable system properties for supporting pervasive and cooperative application sharing across the Internet, such as anonymity, fault tolerance, low maintenance and low administration cost, as well as transparent and dynamic operability, there are some known problems with most of the current P2P systems.

- The first known problem is the *topology mismatch*. The stunning growth and the bandwidth intensive nature of Gnutella-like P2P applications coupled with naive approaches such as Gnutella's random power-law topology presents an inherent mismatch between a P2P application and the physical Internet infrastructure [11].
- The second known problem is the so called *short-cut* effect [2], which occurs primarily due to the diversity in the network resources available to each peer [12].
- The third known problem is the inefficient utilization of network bandwidth in pure broadcast-based P2P search [16].

In this paper we address these problems by developing a number of system-level facilities. First, we propose *multi-tier capability-aware* overlay topologies, aiming at reducing short-cut effect, the topology mismatch and improving the utilization of network resources available to peers. The key idea is two folds. We distinguish low bandwidth peers from high bandwidth peers, assuming higher bandwidth peers will connect to more peers and lower bandwidth peers will have sparse connections in the P2P overlay network. In addition, we advocate a multi-tier network-connection aware topology such that the number of lookup queries served by a peer is commensurate with its capability, and peers with higher bandwidth are assigned to serve more lookup queries, and at the same time are served more efficiently. Such a topology advocates that sparsely connected peers will not be on the path of highly connected peers, thereby avoiding or significantly reducing the short-cut effect. Second, we propose a *Probabilistic Broadening* search technique, which exploits the huge variances observed in terms of the number of resources (files in Gnutella) shared by each peer [1], for improving search quality and efficiency in the P2P system. Combined with result caching, this technique can obtain query answers with significant reduction on P2P traffic and latency. The main idea is to use capability-sensitive metrics to rank neighbors and select a subset of neighbors to forward the query at each step. Our initial experimental results show that the probabilistic broadening is very effective for locating moderately large number of matches with reduced bandwidth consumption and processing cost.

2 Problem Statement and Our Approach

Most loosely coupled P2P search techniques use a *breadth first search (bfs)* (or *scoped broadcast* or *pure bfs*) algorithm. In the pure bfs scheme a query Q is specified by a quadruplet: <*originator, keywords, ID, TTL*>, where $Q.originator$ is the query originator, $Q.keywords$ is the list of user supplied keywords, $Q.ID$ is the unique query identifier and $Q.TTL$ is the Time-to-Live of the query. The query originator assigns the query Q a unique ID ($Q.ID$) and sets the scope of the query $Q.TTL$ to $initTTL$. When a peer p receives a query Q from any neighbor peer q, peer p checks if it is a *duplicate* query (using $Q.ID$). If so, peer p drops the query Q; else peer p sends results from its local file index to peer q. If the query Q's TTL has not yet expired ($Q.TTL > 0$) then peer p forwards the query Q with its TTL decremented by one to all its neighbors (except peer q). With the above search technique, a Gnutella-like P2P document sharing system runs into the following problems:

Problem I: Short-Cut Effect Due to Peer Heterogeneity: It has been observed in [12] that peers are highly diverse in terms of their network resources and their participation times. Unfortunately, most decentralized P2P systems, construct an overlay network randomly resulting in unstable and less powerful peers hindering the system's performance as exemplified by the *Short-Cut Effect* [2]. Consider Figure 1 (note that thicker lines represent links with higher bandwidth). Let peer 1 broadcast a query Q with TTL two. The following sequence of events may lead to peer 4 not receiving the query request although it is two hops from peer 1.

1. At time t, peer 3 receives the query via peer 2. Peer 3 does not forward this query to peer 4 since its TTL is now equal to zero.
2. At a later time instant t' ($t' > t$), peer 3 receives the same query Q directly from peer 1 with $Q.TTL = 1$. This query is dropped at peer 3 as a duplicate query.

Hence, the presence of short-cuts in the overlay network may harm the performance of the system in two ways. First, the number of peers visited by a scoped-broadcast query is lesser, and consequently the number of results obtained by the query decreases. Second, the presence of a weakly connected node on a path between two highly connected nodes (i.e., short-cut) can suffocate the throughput between the powerful nodes.

One way to tackle this problem is to store the query Q's TTL along with the its unique ID on every peer visited by the query and broadcast a duplicate query $dup(Q)$ iff $dup(Q).TTL > Q.TTL$. Although, this ensures that all peers within the scope of the query originator indeed receive the query, it leads to several redundant messages. If X is the set of peers visited by query Q, $dup(Q)$ would visit all peers in set X and each of them would in turn broadcast such redundant queries (since $dup(Q).TTL > Q.TTL$). This triggers a chain of such duplicate query messages (with different TTLs), making such a solution infeasible in view of its bandwidth consumption.

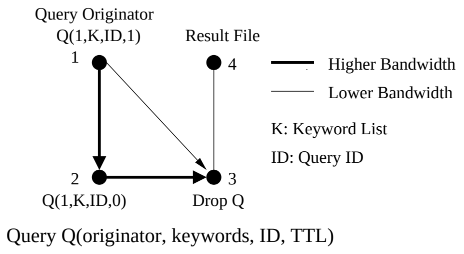

Figure 1: Short-Cut Effect

In this paper, we propose to handle the short-cut effect and other problems caused due to peer heterogeneity by structuring the overlay topology into a multi-tier capability aware P2P network, with each tier consisting of a collection of peers with certain capability level.

Problem II: Inefficient Utilization of Network Bandwidth: A pure broadcast based search is known to consume lots of bandwidth. The iterative-deepening [16] and the random walk [8] technique exploit the fact that a typical user is *satisfied* with a certain *threshold* of query results. The former performs search by iteratively increasing the scope of the search, starting with $initTTL = 1$ and incrementing $initTTL$ by one on every iteration with the *hope* that the query is satisfied within a smaller scope (at some $initTTL < maxTTL$). The later technique performs a random walk on the overlay network until the query is satisfied. However, these schemes do not exploit the huge variances observed in peer capabilities with respect to resources such as bandwidth and the number of documents shared by each peer [12]. In order to obtain results with reduced traffic and latency, we propose a *Probabilistic Broadening* search technique for the document search phase, aiming at utilizing the varying degree of resource sharing exhibited in the present P2P networks. We also demonstrate that this capability-sensitive query forwarding scheme provides an elegant solution for the *stale results* problem in result caching [14].

In comparison with the iterative deepening scheme, our experiments show that the broadening algorithm is suitable for locating moderately large number of matches, while the iterative deepening performs better when the search is intended to locate a small number of matches. Hence, we propose to use the probabilistic broadening technique for the *document search phase* and the iterative deepening technique for the *replica search phase*. The document search phase corresponds to the first phase of Gnutella search, which refers to the process of querying the P2P network for user-supplied keywords. The replica search phase corresponds to the second phase wherein the file is download from multiple file providers using a technique called *swarming* [4]. In order to implement swarming, the results collected in document search phase are required to include a *hash* of the matched documents. When the user chooses a certain result for download, the second phase of search requiring *a small number of matches* is performed using the hash value of the selected document to find peers having *exact replicas* of that document.

3 Capability-Aware Overlay Topologies

3.1 Overview

Before we describe the design of multi-tier capability-aware overlay topologies, we introduce the concept of Heterogeneity Levels, which is used as a guideline to construct connection aware topologies, and a set of Performance Metrics, which serve as the basic model to evaluate and compare different classes of overlay topologies.

We classify nodes into *Heterogeneity Levels* based on their *capabilities* with level zero used to denote the *least powerful* set of nodes. Peers at the same heterogeneity level are assumed to be *almost* homogeneous in terms of their capability. The key idea is to ensure that two nodes whose HLs vary significantly are not directly connected in the overlay network. So, we *allow a connection between two nodes i and j only if* $|HL(i) - HL(j)| \leq 1$.

It is observed in Gnutella [12] that nodes show several orders of magnitude difference in the amount of network bandwidth available to them (slow dial-up connection versus high speed cable connection). Hence, we categorize peers into different heterogeneity levels based on their available network bandwidth. However, noting that the network bandwidth of interest to us is the amount of bandwidth expended in query forwarding on the P2P network, we assume that the capability of a peer i with a 1Mbps connection is say 8 times that of a peer j with a 25Kbps dial-up connection (as against 40 times).

Now we show that the notion of heterogeneity level can be used to counter the short-cut effect. Recall the fundamental reason for the occurrence of short-cut effect: a weak peer lying on the shortest path (in terms of the number of hops in the overlay network) between two powerful peers. Given the fact that we construct the overlay network by only connecting nodes of comparable HLs, we can avoid short-cuts by constructing topologies such that the connectivity of the nodes increases with their HL; thereby making it less likely for a lower level node to appear on the shortest path between two higher level nodes. Further, this is not only important but also very feasible because nodes at higher HL are *more powerful* and hence are capable of handling more neighbors (connections).

We now introduce our performance metrics:

- *Load Distribution:* Distribution of the ratio of the load experienced by a node to its capability.
- *Coverage:* Number of nodes that receive a scoped-broadcast message. Coverage is an indicator of the extent to which short-cuts are mitigated by the topology.
- *Amortized Bandwidth:* Ratio of total bandwidth consumed by a query to the number of results obtained.
- *Amortized Latency:* Ratio of total response time for a query to the number of results obtained.
- *Fault-Tolerance:* Fault-tolerance is measured as the fraction of the number of results obtained when a random set of nodes fail.

We first present an analytic model for designing capability-based multi-tier topologies that minimize the variance of load distribution. We then formally define three classes of overlay topologies: *Hierarchical Topology*, *Layered Sparse Topology* and *Layered Dense Topology*. Each of these topologies enforces certain degree of control over the overlay network structure (with hierarchical topology being the most rigid and restrictive and the dense topology being the least) and is constructed based on the analytical model.

3.2 Analytical Model

Given a collection of heterogeneous nodes with the characteristics discussed above, we would like to construct an overlay topology that minimizes the variance of load to capability ratio. We characterize the overlay topology using the *degree* of each node in the system, where degree refers to the number of open connections maintained by a node. In principle, nodes at the same heterogeneity level have the same (or very similar) capability; it is quite reasonable to assume that they maintain the same (or very similar) number of connections (degree).

Let N be the total number of nodes in the system, $numHL$ be the number of HLs, C_i be the capability of a node at level i ($0 \leq i \leq numHL - 1$), f_i be the fraction of nodes at level i and d_i be the degree of any node at level i. Degree d_i consists of three components, namely, d_i^{up}, d_i^{down} and d_i^{in} such that $d_i^{up} + d_i^{down} + d_i^{in} = d_i$; d_i^{up} denotes the number of edges from a node at level i to nodes at level $i+1$, d_i^{down} denotes the number of edges from a node at level i to nodes at level $i-1$ and d_i^{in} denotes the number of edges from a node at level i to other nodes at the same HL level i.

We derive the load on every node p as a function of its heterogeneity level, the initial value of TTL and the degree parameters d_i^X ($0 \leq i \leq numHL - 1$ and $X \in \{up, down, in\}$). Given the load on every node p in the system and its capability one can compute the variance of load distribution, namely, $Var(load(p)/C_{HL(p)})$ over all nodes p. We resort to optimization techniques like simulated annealing [6] to obtain near optimal solutions for the degree parameters d_i^X that minimize the variance of load distribution. Refer to Appendix A and [15] for further details on our analytical model.

This analytical model serves as the basis for our multi-tier topology design. In the following sections we discuss three classes of multi-tier topologies that are derived using our analytical model with additional restrictions on the degree parameters d_i^X.

3.3 Three Classes of Multi-Tier Topologies

In this section we describe three classes of capability-aware multi-tier topologies: *Hierarchical*, *Sparse*, and *Dense* topologies, each of which exhibit certain restrictions on the topology parameters, namely, d_i^{up}, d_i^{down} and d_i^{in} ($1 \leq i \leq numHL - 1$). These restrictions are important with respect to the design of an overlay topology in (i) Reducing the search space for our optimization problem. (ii) Constructing overlay topologies that not only distribute load evenly, but also perform reasonably well with respect to other performance metrics presented in Section 3.1.

To simplify the discussion without loss of generality, we assume in all examples used in the rest of the paper that we have three classes ($numHL = 3$) of nodes (with HLs ranging from 0 to 2). We also assume that the capabilities of peers at HL = 0, 1 and 2 are in the ratio 1:4:8 (in tune with our argument in Section 3.1). For instance, one could classify peers as indicated in Table 1.

HL	*Peer Class*
0	56Kbps Modem/ISDN and 112Kbps Dual ISDN
1	256 to 768Kbps DSL
2	1Mbps Cable, 1.5Mbps T1/Intranet/LAN

Table 1: Peer Classes

In the following portions of this section, we define and analyze the hierarchical, sparse and dense topologies.

Definitions

Strongly Connected Layer: In all these three topologies the nodes in the highest HL are strongly connected. Namely, $d_i^{in} \geq 0$ for $i = numHL - 1$.

Hierarchical Topology: In a hierarchical topology every node below the highest HL is connected to exactly one node in the immediate higher level. Namely, $d_i^{up} = 1$ and $d_i^{in} = 0$ for $0 \leq i \leq numHL - 2$. See Figure 2.

Sparse Topology: In a sparse topology every node below the highest HL is connected to one or more nodes in the immediate higher level. Namely, $d_i^{up} \geq 1$, $d_i^{in} = 0$ for $1 \leq i \leq numHL - 2$. See Figure 3.

Dense Topology: In a dense topology every node below the highest HL is connected to one or more nodes at the same and/or the immediately higher level. Namely, $d_i^{up} \geq 1$ and $d_i^{in} \geq 0$ for $0 \leq i \leq numHL - 2$. See Figure 4.

Performance Analysis

Short-Cuts: The main motivation to consider a hierarchical topology is because it is *short-cut free*. Consider Figure 2, for any two nodes p and q which can be reached without involving a node at the highest HL, there is a unique path (because of tree structure); And for nodes a and b that have to be reached through the highest HL, there is a unique path from node a to a node c in the highest HL and from node b to a corresponding node d in the highest HL and there is no short-cut between nodes c and d. Observe that the sparse and dense topologies are not short-cut free. Referring to Figure 3, let the shortest path between nodes a and c be *a-b-c*. But, it is always possible that there exists a path a-x_1-x_2-x_3-c (where x_1, x_2 *and* x_3 belong to $HL = 2$) which has lower latency than *a-b-c*. Also, the dense topology will have more short-cuts than a sparse topology since it is *closer* to a random topology in terms of the randomness in its overlay topology.

Load Distribution: In a hierarchical topology, due to its *tree-like* structure, the nodes at the highest HL are likely to be more heavily loaded (relative to their capability) than the nodes at lower levels, thereby making the load distribution somewhat skewed. The sparse topology would display much better load distribution since the restriction on the degree parameters is much weaker than that of a hierarchical topology. Finally the dense topology, which has no additional degree constraints, would display the best load distribution. Nevertheless, all the three topologies show remarkably better load distribution when compared to a random topology that is agnostic to peer heterogeneity.

Coverage: A hierarchical topology is short-cut free, but its tree-like structure, which increases the diameter of the overlay network, tends to bring down the coverage when compared to a random topology. A sparse (or dense) topology has much higher coverage because it permits more connections thereby bringing down the diameter of the overlay network. However, if one compares the coverage of a sparse (or a dense) topology with a random topology of the same diameter one would observer that the former has larger coverage since they avoid short-cuts to a great extent.

Amortized Bandwidth: Bandwidth consumption of a query Q can be estimated by the sum of the coverage of the query Q and the number of duplicates of the query Q. In a hierarchical topology, the tree structure ensures that there is no duplication of queries (except for the nodes at the strongly connected highest HL). Hence, this topology consumes minimum aggregate bandwidth when a scoped-broadcast algorithm is used. However, when iterative algorithms are used, each iteration visits fewer nodes (lesser coverage) in the hierarchical topology; therefore, a larger number of iterations are required to obtain a given threshold of results, leading to an increase in the required bandwidth. Along the same lines of argument, the sparse and dense topologies consume more aggregate bandwidth when a pure-broadcast algorithm because they have larger coverage; but, their larger coverage rewards them with more results thereby bringing down the amortized bandwidth consumption. Also, when the iterative algorithms are used their larger coverage brings down the expected number of iterations, and hence the bandwidth, required to satisfy

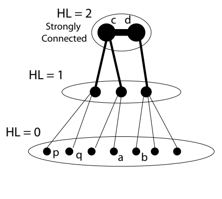

Figure 2: Hierarchical Topology

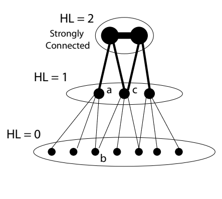

Figure 3: Sparse Topology

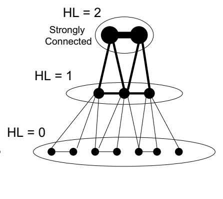

Figure 4: Dense Topology

a given query.
Amortized Latency: Latency is inversely related to coverage. Larger coverage implies that more nodes (and hence more results) can be visited in a smaller number of hops. Hence, the expected amortized latency in a hierarchical topology would be larger than a random topology, which would in turn be larger than a sparse/dense topology
Fault-Tolerance: In a hierarchical topology, if a node fails, then the entire *sub-tree* rooted at that node gets disconnected from the P2P network temporarily. Hence, maintaining a hierarchical topology is very costly. The sparse and dense topologies show increasingly higher fault tolerance, but still they assign *higher responsibility* to higher level nodes thereby making them marginally weaker than the random topology. However, the *uniform faults* assumption, wherein all nodes are equally likely to fail, is particularly not true in the case of Gnutella-like systems. Note that in a large-scale decentralized P2P system like Gnutella, most *failures* are due to nodes leaving the P2P system. It is observed in [12] that nodes with powerful network connections stay longer in the system as against nodes with weak network connections (like dial-up users). Hence, the probability of failure of a higher-level node (because of it leaving the system) is much lower than that of a lower-level node. Under this assumption of *non-uniform faults* the multi-tier topologies are likely to be more fault-tolerant since they assign more responsibility to more capable nodes that are less likely to fail.

Summary: Table 2 presents a summarization of a comparison between the overlay topologies. In conclusion, we promote the sparse topology as the best structure for building a multi-tier capability-aware overlay network since it strikes a good balance between load distribution, short-cuts and other performance metrics.

Topology Type	*Load Distribution*	*Short-Cuts*	*Coverage*
Random	very poor	very large	medium
Hierarchical	poor	nil	low
Layered Sparse	good	very small	high
Layered Dense	very good	small	high
Topology Type	*Bandwidth*	*Latency*	*Fault Tolerance*
Random	high	medium	good
Hierarchical	medium	high	very poor
Layered Sparse	low	low	good
Layered Dense	low	low-medium	good

Table 2: Summary: Overlay Topologies

3.4 Topology Construction and Maintenance

The multi-tier topology construction consists of two main components: (i) bootstrapping a new node. (ii) maintaining the overlay topology in the presence of node departures and network failures.

Topology Construction
In a typical decentralized P2P system, a collection of publicly well-known bootstrap servers maintain a cache of recently joined nodes. Now, when a new node p wants to join the P2P system, we require the bootstrap server to provide it with an appropriate set of nodes q (depending on $HL(p)$) according to table 3 and the node degree information required to be maintained by node p according to the topology degree parameters, namely $d^X_{HL(p)}$, $X \in \{up, down, in\}$.

Topology	$p \notin maxHL$	$p \in maxHL$
Hierarchical	$HL(q) = HL(p) + 1$	$q \in maxHL$
Sparse	$HL(q) = HL(p) + 1$	$q \in maxHL$
Dense	$HL(q) = HL(p) + 1 \vee HL(q) = HL(p)$	$q \in maxHL$

Table 3: Bootstrapping node p in a Multi-Tier Topology

A new node p **proactively** opens connections to nodes that are at the same or higher level than node p. However, the connections from a node p to a node q at a lower heterogeneity level ($HL(q) < HL(p)$) than node p are initiated by node q rather than node p. Since the nodes with larger bandwidth are likely to stay connected to the system for a longer period of time, the probability that a node would loose a connection with a higher-level node is quite low, thereby reducing the topology maintenance costs discussed below.

Topology Maintenance
Once a node has joined the overlay network, it may loose connections with some or all of its neighbors due to various reasons including faults in the underlying network, departure or failure of the neighbors themselves. In the event of such failures, if a peer p were to still retain connection(s) with a subset of its neighbors, then these peers (and recursively their neighbors) can provide a set of peers to whom peer p can potentially connect to; else peer p contacts one of the bootstrap servers to obtain a new entry point peer. Note that a node only needs to maintain the required number of connections with other nodes at the same or higher levels. Also, observe that the maintenance cost of a sparse or a dense topology would be much lower than a hierarchical topology.

Due to the space restriction, we have omitted the discussion on formal properties of the three classes of multi-tier topologies and the concrete algorithms for construction and maintenance of such topologies. Readers may refer to [15] for more technical details.

4 Probabilistic Broadening Search

4.1 Design Ideas

Recall the scoped bfs search technique (pure-bfs) used in most Gnutella-like P2P systems today, query originator initializes $Q.TTL$ to $maxTTL$ and consequently the query Q reaches all peers that are at most $maxTTL$ hops from the originator. Alternatively, the restricted depth first search ($rdfs$) or the iterative deepening technique attempts to be *lucky* by *satisfying* the query within a smaller scope (fewer hops). Each query is associated with another parameter $Q.threshold$, specifying the number of results required. The query originator iterates over query's $initTTL$, starting from $Q.initTTL$ = $minTTL$ to $maxTTL$ ($minTTL$ and $maxTTL$ are system parameters) until the query Q is satisfied. However, all these search techniques do not exploit the huge variances observed in terms of both the number of documents shared by each peer and the bandwidth capability of different peers. In an open P2P system like Gnutella in which a large number of non-cooperating peers are present, it has been observed that a large number of peers (70%) are free-riders [1] and that about 90% of documents is shared by about 10% of the nodes [12]. This means that most of the peers do not contribute to the peer community but merely utilize the resources provided by a small subset of peers in the community.

The key idea behind our probabilistic broadening algorithm is to promote a focused search through selective broadcast. The selection takes into account the peer heterogeneity and the huge variances in the number of documents shared by different peers. Our search algorithm iterates over the number of neighbors to whom the query is forwarded at each step. This is accomplished by adding a breadth parameter B to the query. The query originator iterates over the breadth parameter $Q.B$ from $minB$ to $maxB$ ($minB$ and $maxB$ is a system defined parameter). In each iteration the query Q takes $maxTTL$ hops but is forwarded only to $Q.B$ neighbor peers at each hop. A main distinction of our algorithm, in contrast to other existing search schemes, is that, in a given iteration when the query is required to be forwarded to say n neighbors, conscious effort is put forth to ensure that the query is sent to the *best* subset of n neighbors, rather than *any* or all of the n neighbors. We use a capability-aware ranking algorithm to select the best B neighbors from a set of neighbor peers. The ranking algorithm also takes into account the performance of peers in the recent past and dynamically updates the rankings as peers join or leave the system.

The probabilistic broadening search algorithm comprises of three major components: *Ranking neighbor peers*, *Processing a query using the ranking information*, and *Maintaining the rankings up-to-date*. We below describe the ideas of the algorithmic design of these three components. Readers may refer to [15] for the algorithm details.

4.2 Ranking Neighbor Peers

In a P2P system, peers are constantly interacting with their neighbors as a part of their query forwarding responsibility. Also, the results of a query retrace the path to the query originator. Note that the results forwarded to peer p by its neighbor peer q not only includes the results from the local file index of peer q, but also includes the results from other peers which received the query via peer q. A peer p could build the following metrics to rank the *goodness* of a neighboring peer q based on its interactions with peer q:

- Max Degree: Degree refers to the number of neighbors to which a peer is connected. Peers maintaining large degree can be expected to have high processing power and network bandwidth.
- Max Results: The number of results returned per query by a neighbor over the last T time units.
- Max Rate: The ratio of the number of results returned per query to the time taken to return them over the last T time units.

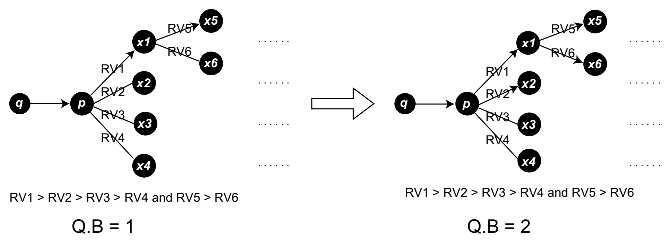

Figure 5: Broadening Algorithm: Illustration

In addition to use each metric independently as a ranking criterion, one could use a weighted combination of the above parameters to obtain a more sophisticated ranking metric: $RV(p) = w_1 * Degree(p) + w_2 * Results(p) + w_3 * Rate(p)$ where $w_1 + w_2 + w_3 = 1$. From our experiments, we observed that in a random topology, all the metrics suggested above have very similar performance. However, the connectivity information (Max Degree) is inbuilt in our multi-tier topologies; hence Max Results and Max Rate show better performance on our topologies. Hence, without loss of generality, in the rest of the paper we use $MaxResults$ as the metric for all discussions.

4.3 Query processing by Probabilistic Broadening

Query processing using the probabilistic broadening search technique consists of three steps: (i) The query originator iterates through the breadth parameter $Q.B$ from $minB$ to $maxB$ until the query Q is satisfied. In each iteration (if the query is not yet satisfied), the query originator issues the query with the query's TTL set to $maxTTL$. (ii) When a peer p receives a query Q from peer q, peer p chooses the best $Q.B$ neighbors (excluding peer q) and forwards the query only to them. (iii) When a peer q receives the results for a query Q from its neighbor peer p it updates the goodness metric for peer p. Figure 5 illustrates our algorithm at peer p for $Q.B = 1$ and $Q.B = 2$ with ranking values $RV1 > RV2 > RV3 > RV4$ and $RV5 > RV6$.

4.4 Maintaining Rankings Up-To-Date

The key problem in maintaining the rankings up-to-date is the following: Say at time instant t, a peer p ranks its neighbors $X = \{x_1, x_2, \cdots, x_n\}$ as $x_1 \succ x_2 \succ \cdots \succ x_n$. Let the average breadth parameter ($Q.B$) at which a typical query is satisfied be b. So, peer p sends most of its queries to peers $x_1, x_2, \cdots, x_b$ at time t' ($t' > t$). Consequently, the ranking measures of peers $x_{b+1}, x_{b+2}, \cdots, x_n$ are not updated by peer p since peer p did not forward queries to (and thus not get results from) these peers. Hence, changes to the file indices of peers in the overlay network that are *accessible* through peers $x_{b+1}, x_{b+2}, \cdots, x_n$ are not considered subsequently.

To capture the dynamics of the P2P network we modify *neighbor selection step* as follows: Instead of *deterministically* selecting the best $Q.B$ peers, peer p selects $Q.B$ peers *probabilistically*, that is, each of the neighbor peer x_j is selected with a probability proportional to its ranking value $RV(x_j)$. Hence, most of the queries get routed through the neighbors who have performed well before; yet, by probabilistically sending the query to some inferior neighbors, peer p can figure out if they can provide us better results *now*.

4.5 Discussion

In comparison with other iterative-algorithms and random walks, the probabilistic broadening algorithm ($pbbfs$) is similar in its attempts to make the query processing more sequential at the cost of increased latency. A unique feature of the probabilistic broadening algorithm is that it performs a focused search starting with a small number of (highly ranked) very capable peers and proceeding towards larger number of less capable peers. But, $pbbfs$ is not quite suitable when the number of query results required is smaller (like in the replica search phase). This is because the coverage of the $rdfs$ (or random-walk) scheme is lower for smaller number of iterations as against $pbbfs$; and usually a small threshold can be satisfied within a small number of iterations. To illustrate this: assume that the average connectivity (number of agents) of a peer is equal to d. Hence, the number of peers that can be visited in n iterations of $rdfs$ (or random-walk) is of the order of d^n, whereas the number of peers visited by $pbbfs$ in n iterations would be of the order of n^{maxTTL}. For small values of n, $pbbfs$ visits more peers and hence fetches much more results than the threshold required. But, for moderate values of threshold (around the average number of results returned by bfs), $pbbfs$ performs better because of the focused nature in its search technique.

4.6 Integrating $pbbfs$ with Result Caching

It is observed by many [9, 13] that the query traffic on a Gnutella network is short, infrequent and bursty. Also, a substantially large fraction of the queries are repeated in a short interval of time. The major hurdle in caching results in a highly dynamic nature like Gnutella is the problem of effectively handling stale-results [9].

The key motivation for using $pbbfs$ with result caching arises from the fact that both of these techniques share a common design philosophy. Result caching is a fine-grained ranking system in which, a peer p can estimate the number of results that could be obtained through each of its neighbors for any given query Q. Also, the ranking metric $MaxResults$ used in the broadening scheme can be computed as *aggregates* of the cached results. Hence, the techniques used to maintain peer rankings up-to-date in the $pbbfs$ scheme can be effectively adapted to solve the stale-results problem in the result caching. For example, sudden fluctuations in the rankings of a neighbor (say x_k) are highly likely to be due to the fact that some peers accessible through x_k left the network. Hence, the confidence level on the cached entries obtained from x_k could be reduced. This could be implemented, for example, by decreasing the time-to-live counter for the relevant entries at a larger rate. Hence, the amicability of result caching with $pbbfs$ and the benefit from improved performance under bursty traffic makes it very attractive for us to use the two in conjunction.

5 Experiments and Results

We have performed three sets of simulation-based experiments. The first set evaluates the effectiveness of multi-tier capability-aware topologies against a random topology. The second set compares probabilistic broadening algorithm ($pbbfs$) with iterative deepening ($rdfs$) and the pure breadth first search technique (bfs). The third set shows the effect of combining $pbbfs$ with result caching and our solution for handling the stale results problem.

5.1 Simulation Set Up

Our simulation setup comprises of three main modules: the Network Generator, the Document Generator, and the Search Simulator. Tables 4, 5 and 6 present the major tunable parameters, most of which are obtained from observations made on the real Gnutella network.

Network Generator. We use our multi-tier bootstrap algorithm and construct the overlay network incrementally by adding one node at a time. For our simulation we use 50%, 30% and 20% of the peers for the corresponding HL = 0, 1 and 2. *Document Generator*. We use Zipf-like distribution wherein the number of documents that match the i^{th} most popular query is proportional to $1/i^{\alpha}$ (with parameter $\alpha = 1$). We use *Document Bias (Db)* to specify non-uniform distribution of documents amongst peers (Db of 20%-80% means that 20% of the peers hold about 80% of the documents).

Search Simulator. The search simulator implements one of the following search techniques: scoped broadcast (bfs), probabilistic broadening ($pbbfs$), restricted deepening ($rdfs$), and probabilistic broadening with result caching.

5.2 Evaluation of Overlay Topologies

We compare the overlay topologies using five performance metrics: Coverage, Mean Bandwidth Consumption, Search Latency, Fault Tolerance, and Load Distribution using the probabilistic broadening algorithm with $initTTL$ equal to 7. Our last experiment overlay topologies demonstrates the importance of partitioning peers into the right set of heterogeneity levels.

Load Distribution

Figure 6 shows the variation of load distribution among the four overlay topologies that are normalized with respect to a $N = 1000$ node random topology. For $N = 10,000$ the sparse and topology show 80 times lower variance, while a hi-

Parameter	*Description*	*Default*
N	Number of Nodes	10,000
HL	Number of Peer Classes	3
$Maxd$	Maximum Peer Degree	10
$Mind$	Minimum Peer Degree	1
Dd	Degree Distribution (Random/Power Law)	Power Law

Table 4: Network Generator

Parameter	*Description*	*Default*
Nd	Number of Documents	100,000
Ndd	Number of Distinct Documents	10,000
Db	Document Bias	20%-80%

Table 5: Document Generator

Parameter	*Description*	*Default*
$THR1$	Phase I result threshold	32
$THR2$	Phase II result threshold	4

Table 6: Search Simulator

erarchical topology shows 10 times lower variance. The dense topology shows marginally higher variance than the sparse topology because the presence of more short-cuts decreases the accuracy of its analytical model.

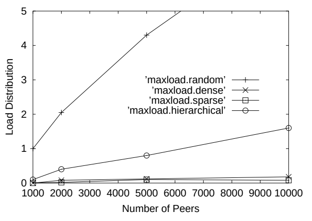

Figure 6: Variance of load/capability

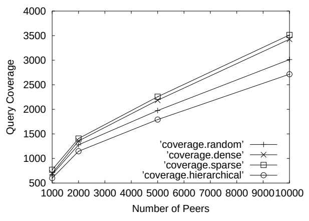

Figure 7: Query Coverage

Coverage

We have performed two tests on coverage: The first test compares the coverage among different overlay topologies. The second test shows the distribution of the coverage with respect to the heterogeneity level of peers.

Figure 7 shows the average coverage of a peer in the network, with scope $initTTL = 7$. For $N = 10,000$ nodes, the layered dense and sparse topologies show 20% more coverage, while the hierarchical topology shows 15% lower coverage because of its increased diameter, even though it is shortcut free.

Table 7 shows the variation of coverage with different HLs of peers that are normalized with respect to $HL = 0$ peers. One key conclusion drawn from this table is that *the extra work done by the higher HL peers rewards them with larger coverage (and thus more results).*

Topology	$HL = 0$	$HL = 1$	$HL = 2$
Hierarchical	1.0	5.2	15.3
Layered Sparse	1.0	3.6	7.7
Layered Dense	1.0	2.9	6.0

Table 7: Coverage Vs HL for $N = 10,000$ nodes

Amortized Bandwidth Consumption

Figure 8 presents the average number of messages required for a given result threshold for a search on various topologies. For $N = 10,000$ nodes, the sparse topology consumes almost 25% lesser bandwidth than the random topology; whereas the dense and the hierarchical topology consume about 12.5% lesser bandwidth compared to the random topology. Also, observe that the difference in the amortized bandwidth consumption increases with the number of nodes in the system.

Amortized Latency

Figure 9 shows the plot between amortized latency and the

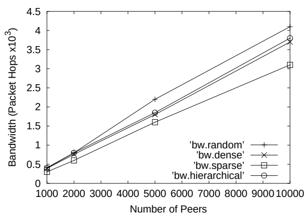

Figure 8: Amortized Bandwidth

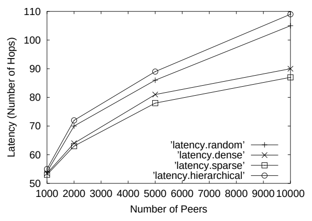

Figure 9: Amortized Latency

number of peers for different topologies. For $N = 10,000$ nodes, the figure shows that the sparse and dense topology save about 20% and 15% latency respectively when compared to a random topology; however, the hierarchical topology incurs about 5% additional latency. Also, similar to the amortized bandwidth consumption, the difference in the latency incurred between the topologies increases with the number of nodes in the system.

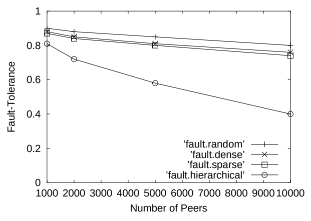

Figure 10: Fault Tolerance with uniform faults

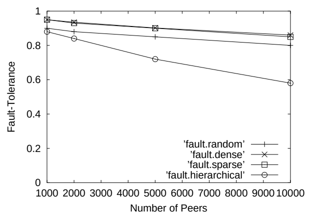

Figure 11: Fault Tolerance with non-uniform faults

Fault Tolerance

We study the fault-tolerance of the four topologies under two conditions: *Uniform Faults* and *Non-Uniform Faults*.

Figure 10 shows the quality of the results obtained when a random 10% of the peers fail under uniform faults. Quality of results is expressed as the ratio of the number of results obtained under faulty conditions to that obtained when all the peers were functional. For $N = 10,000$ nodes, the hierarchical topology shows about 50% lower fault-tolerance than the random topology; while the sparse and dense topology exhibit only 2-3% lesser fault-tolerance.

Figure 11 shows the fault-tolerance of the topologies under non-uniform faults with 10% of failed peers, where the probability of peer failure at HL = 0, 1 and 2 are in ratio 3:2:1. For $N = 10,000$ nodes, the sparse and dense topology show about 4-5% more fault-tolerant than the random topology and the hierarchical topology shows 20% improvement as against the uniform faults case.

Topology Type	*Coverage*	*Amortized Bandwidth*	*Amortized Latency*	*Fault-Tolerance (non-uniform faults)*	*Load Distribution load/capacity variance*
Hierarchical	0.97	1.06	0.96	1.06	21.0
Sparse	0.99	1.17	1.16	0.92	11.0
Dense	0.99	1.13	1.12	0.91	9.0

Table 8: Importance of choosing correct $numHL$

Importance of Choosing Correct *numHL*
To demonstrate the importance of determining the correct value of $numHL$, we compare the performance of a system with an incorrect value for $numHL$ to that of a system which used the right value. Assume that the number of genuine peer classes in the system is three. We constructed a hierarchical, a sparse, and a dense topology using these peers for $numHL = 2$ and $numHL = 3$. Table 8 shows ratios of the performance measures obtained for $numHL = 2$ to those obtained for $numHL = 3$. For instance, a sparse topology with $numHL = 2$ shows 1% lesser coverage, 17% more bandwidth, 16% more latency, 8% lesser fault-tolerant, and 11 times more variance in load distribution than a sparse topology with $numHL = 3$.

5.3 Evaluation of Search Techniques

We evaluate the performance of the probabilistic broadening search technique ($pbbfs$) by comparing it with the pure bfs and the iterative deepening ($rdfs$) approaches in terms of their bandwidth and latency requirements. Figure 12 shows the amount of bandwidth consumption (in Packet Hops) for $initTTL$ equal to 7. We observe that when the threshold is set to the average number of results returned by a scoped bfs search, probabilistic broadening ($pbbfs$) shows 23.1% lesser bandwidth requirement.

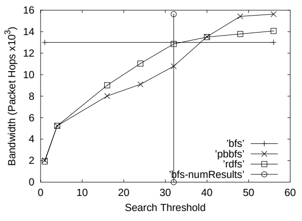

Figure 12: Search Techniques: Bandwidth (TTL = 7)

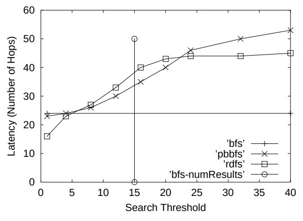

Figure 13: Search Techniques: Latency (TTL = 7)

An interesting observation from the figure is that the threshold based schemes fail for large values of the threshold. This is because the threshold-based iterative algorithms were designed with the hope that the query would be satisfied without searching all the peers in the system. When the entire scope of the network has to be searched, the redundant portions in their iterations become the main cause for poorer performance. For comparison, the horizontal line in the figures show the bandwidth consumption of the pure bfs algorithm. The vertical line indicates the average number of results returned by bfs.

Figure 13 shows the search latency, namely the amount of time taken from the time query is issued to the time a certain $threshold$ of results are obtained. Again the vertical line denotes the average number of results returned by pure bfs. The naive flooding based bfs scheme is the fastest as it exploits the maximum parallelism of all neighbor peers. On the other hand, the built-in conservative approach in the iterative algorithms makes them slower than the pure bfs. Nevertheless, the probabilistic broadening based search $pbbfs$ shows about 13% lower latency when the search threshold equals the average number of results returned by a pure bfs.

5.4 Evaluating $pbbfs$ with Result Caching

Now we compare the performance of the search techniques with result caching enabled. A simple LFU based cache of a fixed size was used to evaluate the search techniques with respect to their bandwidth consumption. Figure 14 shows the plot between bandwidth and threshold. The vertical and horizontal lines indicate the results for the pure bfs scheme. Note that when the search threshold is set to the average number of results returned by pure bfs with caching, $pbbfs$ shows about 20% lower bandwidth consumption than $rdfs$.

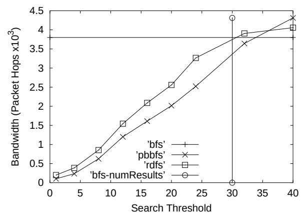

Figure 14: Search Techniques with Result Caching

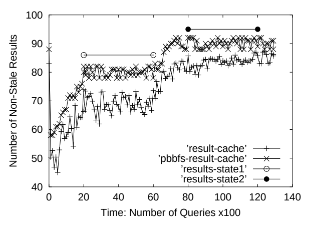

Figure 15: Result-caching with $pbbfs$

We also performed experiments to show how well $pbbfs$ aids result caching in responding to changes in the network topology in the form of peer joins or departures. We performed experiments where a larger fraction of peers leave or join the network. We present an anecdotal description of one such experiment. In Figure 15, we compare the number of non-stale results obtained using result caching with $pbbfs$ to handle the stale results problem (*pbbfs-result-cache*) to that of the naive result caching technique (*result-cache*) under the following scenario:

State 1: At time $t = 0$, 20% of the peers leave the network. Within 2000 system wide queries, $pbbfs$ adapts itself to the sudden loss of peers. In the mean while, the naive result-caching scheme yields on an average 15% smaller number of non-stale results.

State 2: At time $t = 60$, 10% of the peers join. In a little more than 1000 system wide queries, $pbbfs$ adapts itself to the sudden addition of peers. The naive result caching scheme requires over 6000 system wide queries before it catches up with $pbbfs$ in terms of the mean number of non-stale results.

For comparison, we have shown number results actually available in the system in state 1 and 2 in *results-state1* and *results-state2* respectively. These results show that result caching in conjunction with $pbbfs$ is not only more bandwidth efficient, but is also capable of handling the stale-results problem effectively.

6 Related Work

In the past few years research on P2P systems has received a lot of attention. Several research efforts have been targeted at improving the performance of P2P search [16, 3, 8]. These papers suggest enhancements over the naive flooding-based bfs algorithm by fine-tuning their search schemes based on measurement studies conducted on user characteristics, distribution of files among peers, etc. For example *Routing Indices* in [3] exploits the locality of the queries and the uneven distribution of different categories of files among peers. In addition, several traces and case studies of the Gnutella networks have highlighted the importance of result caching for Gnutella networks [14]. Suggestions have been made to implement caching as a part of Gnutella protocol [7]. But none to our knowledge has addressed the short-cut effect and exploited the capability-awareness in P2P topology construction and P2P search techniques.

Several have pointed out that peer heterogeneity would be a major stumbling block for Gnutella [10, 12]. Solutions based on super-peer architectures have been proposed in [5] to alleviate the problem of peer heterogeneity. The super-peer architecture can be viewed as a hierarchical topology with $numHL = 2$. Our work not only generalizes $numHL$ to arbitrary values, promoting multi-tier layered sparse topology over the hierarchical topology, but also provides an analytical model that yields the desired degree information to precisely construct capability-aware overlay topologies.

7 Conclusion

The key problems that have plagued a Gnutella like P2P systems are *Peer Heterogeneity* and its *bandwidth consumption*. Most of the P2P developments today treat all peers as equal, and thus ignore variations in their capabilities. Utilizing such inherent differences among peers can significantly improve the performance of the P2P system.

We have proposed simple yet effective multi-tier capability-aware topologies for improving the P2P search performance. Such topologies can be realized using simple bootstrapping and do not impose high construction or maintenance costs. The main contributions of this paper are two folds. First, we propose techniques to structure overlay topologies taking peer heterogeneity into account so as to ensure that the performance of the overlay network is not hindered by less powerful peers. Second, we develop a capability-aware search technique that further enhances the search performance. We also demonstrate the combined benefit of integrating the probabilistic broadening search technique with result caching. Finally, our design and techniques being simple and pragmatic can be easily incorporated into existing systems like Gnutella.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. http://www.firstmonday.dk/issues/issue5_10/adar, 2003.

[2] F. S. Annexstein, K. A. Berman, and M. Jovanovic. Latency effects on reachability in large-scale p2p networks. In *ACM Symposium on Parallel Algorithms and Architectures*, Crete Island, Greece, 2001.

[3] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of International Conference on Distributed Computing Systems*, July 2002.

[4] Gtk-Gnutella. The graphical unix gnutella client. http://gtk-gnutella.sourceforge.net/, 2003.

[5] F. S. Inc. Super-peer architectures for distributed computing. http://www.fiorano.com/whitepapers/ superpeer.pdf, 2002.

[6] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimization by simualated annealing. *Science, Number 4598, 13 May 1983*, 1983.

[7] LimeWire. Improving gnutella protocol: Protocol analysis and research proposals. http://www9.limewire.com/ download/ivkovic_paper.pdf, 2002.

[8] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *16th annual ACM International Conference on supercomputing*, 2002.

[9] E. P. Markatos. Tracing a large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[10] S. R. Qin Lv and S. Shenker. Can heterogeneity make gnutella scalable? In *Proceedings of the first International Workshop on Peer-to-Peer Systems*, 2002.

[11] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale p2p systems and implications for system design. In *IEEE Internet Computing Journal, vol. 6, no. 1, 2002.*

[12] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, University of Washington, 2001.

[13] A. Singh. Mini Project I. http://www.cc.gatech.edu/ aameek/projects/mps/mp1.html, 2002.

[14] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. http://www-2.cs.cmu.edu/k̃unwadee/research/p2p/paper.html, 2001.

[15] M. Srivatsa, B. Gedik, and L. Liu. Improving peer to peer search with multi-tier capability-aware overlay topologies. Technical report, Georgia Institute of Technology, 2003.

[16] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *22nd International Conference on Distributed Computing Systems (ICDCS'03)*, July 2002.

Appendix A

Let N denote the total number of nodes in the system, $numHL$ denote the number of heterogeneity levels, $maxHL$ denote the highest (most powerful) heterogeneity level, $minHL$ denote the lowest (weakest) heterogeneity level and C_i denote the capability of a node at level i ($0 \leq i \leq numHL - 1$). Let f_i denote the fraction of nodes at level i, d_i denote the degree of any node at level i. d_i consists of three components, namely, d_i^{up}, d_i^{down} and d_i^{in} such that $d_i^{up} + d_i^{down} + d_i^{in} = d_i$; d_i^{up} denotes the number of edges from a node at level i to nodes at level $i+1$, d_i^{down} denotes the number of edges from a node at level i to nodes at level $i-1$ and d_i^{in} denotes the number of edges from a node at level i to other nodes at the same HL level i. Note that $d_{numHL-1}^{up} = 0$ and $d_0^{down} = 0$. Also, d_i^{up} and d_{i+1}^{down} are not independent since the total number of edges from nodes at level i to level $i+1$ should equal the total number of edges from nodes at level $i+1$ to level i. Hence, $\forall i, 0 \leq i \leq numHL - 2$, the following equation holds:

$$f_i * d_i^{up} = f_{i+1} * d_{i+1}^{down} \quad (1)$$

We first discuss the relationship between the load on a node and its coverage. We then show that the coverage of any given node can analytically be modeled as a function of its heterogeneity level and $initTTL$, the initial value of Time-to-Live of the query.

Let p be any node in the system. Assume that queries are issued with $TTL = initTTL$. Let $X(p)$ denote the set of nodes that would be visited by a query Q from node p with scope $initTTL$. By definition, the coverage of node p (denoted as $coverage(p)$) equals $|X(p)|$. Symmetrically, when any node $q \in X(p)$ issues a query, the query would reach node p. However, the presence of short-cuts in the topology destroys this kind of symmetry. For instance, referring Figure 1, we find that node 4 may not be reachable from node 1 in two hops while node 1 is reachable from node 4 in two hops.

From the above discussion, we observe two facts. First, the coverage of a node p gives a good estimate of the number of nodes whose queries need to be processed by node p. Assuming that each node issues requests to the system that follows a Poisson Process with mean λ requests per second, the load on node p can be estimated by:

$$load(p) = \lambda * coverage(p) \quad (2)$$

Second, uniformly ignoring the short-cut effect in measuring the load on every node ensures that we indeed get a good estimate for the *variance* of load distribution.

We compute the coverage of a node at a given HL in rounds, where each round corresponds to a particular value of the query's TTL. In each round, we compute the number of nodes at each HL that would receive the query. We then eliminate those nodes that would have received the query in earlier rounds. Finally, only those nodes that received the query for the first time in the current round contribute to the number of nodes that would receive the query in the next round. This process is repeated until all rounds are exhausted (until TTL expires).

Let $COV(hl, ttl)$ denote the number of *new* nodes at level *hl* that would be reached by a query Q with $Q.TTL = ttl$. A node q is a new node if it has not received any duplicate of the query Q with TTL $> ttl$. We define COV recursively with the following base case: If node p issues a query Q, then node p is the only new node that received the query with $TTL = initTTL$. Therefore,

$$COV(hl, initTTL) = \begin{cases} 1 \; if \; hl = HL(p) \\ 0 \; otherwise \end{cases} \quad (3)$$

Let $REQ(hl, ttl)$ denote the total number of queries with $TTL = ttl$ that would hit a given heterogeneity level *hl* ($0 \leq hl < numHL$). We can compute $REQ(hl, ttl)$ using COV as follows:

$$\begin{aligned} REQ(hl, ttl) &= COV(hl+1, ttl+1) * d_{hl+1}^{down} \quad (4) \\ &+ COV(hl, ttl+1) * d_{hl}^{in} \\ &+ COV(hl-1, ttl+1) * d_{hl-1}^{up} \end{aligned}$$

where $d_{numHL}^{down} = 0$ and $d_{-1}^{up} = 0$. Note that in the formula to compute $REQ(hl, ttl)$ we have not accounted for the fact that a node p would not forward the query to the node from whom it actually received the query. This is because the following portion of the analysis eliminates those nodes that have already received a query.

Among $REQ(hl, ttl)$ queries, some of them reach nodes that have already received the query; while the rest reaches new nodes. The number of nodes that have NOT received the query prior to $Q.TTL = ttl$, denoted by $NOTRCVD(hl, ttl)$, can be computed as follows:

$$NOTRCVD(hl, ttl) = N * f_{hl} - \sum_{i=ttl+1}^{initTTL} COV(hl, i) \quad (5)$$

Hence, the probability that a query Q with $Q.TTL = ttl$ reaches a new node at level *hl* is:

$$PRNEW(hl, ttl) = \frac{NOTRCVD(hl, ttl)}{N * f_{hl}} \quad (6)$$

Hence, the number of queries Q that are forwarded to new nodes at level *hl* when $Q.TTL = ttl$ can be computed by:

$$REQNEW(hl, ttl) = PRNEW(hl, ttl) * REQ(hl, ttl) \quad (7)$$

Now, $REQNEW(hl, ttl)$ queries are directed towards $NOTRCVD(hl, ttl)$ nodes at level *hl* when $Q.TTL = ttl$. We have to estimate the number of distinct nodes (among $NOTRCVD(hl, ttl)$ nodes) that indeed receive the query request. For given *hl* and *ttl*, define a binary random variable

Y_q for every node $q \in NOTRCVD(hl, ttl)$, as follows:

$$Y_q = \begin{cases} 1 \; if \; q \in COV(hl, ttl) \\ 0 \; otherwise \end{cases} \quad (8)$$

Note that for the simplicity of notation, we have overloaded NOTRCVD and COV as a set as well as the number of nodes. Define a random variable Y as, $Y = \sum_{q \in NOTRCVD(hl,ttl)} Y_q$. Now, the expected value of Y, namely $E[Y]$, is the number of distinct new nodes that receive the query at level hl and at $Q.TTL = ttl$. Hence,

$$COV(hl, ttl) = E[Y] = \sum_{q \in NOTRCVD(hl,ttl)} E[Y_q] \quad (9)$$

For any node $q \in NOTRCVD(hl, ttl)$, $E[Y_q] = 1 \times Pr(Y_q = 1) + 0 \times Pr(Y_q = 0) = Pr(Y_q = 1)$. Now, $Pr(Y_q = 0)$ is the probability that none of the $REQNEW(hl, ttl)$ queries reach node q. Also, the probability that any request in $REQNEW(hl, ttl)$ reaches node q is $\frac{1}{NOTRCVD(hl,ttl)}$ (a request is equally likely to hit any of the nodes in the set $NOTRCVD(hl, ttl)$). Hence for any $q \in NOTRCVD(hl, ttl)$,

$$\begin{aligned} Pr(Y_q = 1) &= 1 - Pr(Y_q = 0) \\ = \; 1 - \left(1 - \frac{1}{NOTRCVD(hl, ttl)}\right)^{REQNEW(hl,ttl)} \end{aligned} \quad (10)$$

Therefore, we can compute the expected number of new nodes that receive the query at level hl and $Q.TTL = ttl$ from equation 9:

$$COV(hl, ttl) = NOTRCVD(hl, ttl) * Pr(Y_q = 1) \quad (11)$$

This completes the recursive definition of the COV function. The Equations 4 to 11 recursively define the COV function with the base case in Equation 3. Finally, the coverage of a query from node p is computed as,

$$coverage(p) = \sum_{hl=0}^{numHL-1} \sum_{ttl=0}^{initTTL} COV(hl, ttl) \quad (12)$$

Observe that in our analytical model, the coverage of a node p depends only on the heterogeneity level of the node p and the initial value of TTL. Therefore, by computing the coverage of one node at each heterogeneity level from $0, 1, \cdots, numHL - 1$, we can get a good estimate of the load on any node in the system. Finally, given the load on every node p in the system and the capability of the node, which is captured by the capacity of the heterogeneity level of the node, denoted by $C_{HL(p)}$, one can compute the variance of load distribution, namely, $Var\left(\frac{load(p)}{C_{HL(p)}}\right)$ over all nodes p.

Note that the set of recursive equations listed above, can be easily converted into an iterative algorithm of complexity O($numHL \times initTTL$) that accepts d_i^{up} and d_i^{in} for all $0 \leq i \leq numHL - 1$ (recall that Equation 1 constraints d_i^{down}) as inputs and outputs the variance of load distribution. One can resort to optimization techniques like simulated annealing [6] to obtain near optimal solutions that minimize the variance of load distribution.

Observe that one could extend this model to accommodate the short-cut effect by defining a round as the shortest possible time required for a hop, $rt = CL(maxHL, maxHL)$, and approximating any $CL(x, y)$ ($0 \leq x, y < numHL$) as an integral multiple of round time rt. Now, using a similar technique as described above, we could write recursive equations for $COV(hl, rn, ttl)$, where $COV(hl, rn, ttl)$ denotes the number of new nodes at heterogeneity level hl that receive the query in round number rn with the query's TTL being ttl. Note that the maximum number of rounds required is $numR = CL(maxHL, maxHL)/CL(minHL, minHL) * initTTL$. However, we chose not to do so since, this computation being O($numHL \times numR \times initTTL$) is prohibitively expensive especially since $numR$ could be very large. An interesting alternative would be to decrease the granularity of the round time at the cost of increasing inaccuracy of the model.