

Scaling Unstructured Peer-to-Peer Networks With Multi-Tier Capacity-Aware Overlay Topologies

Mudhakar Srivatsa, Buğra Gedik and Ling Liu

College of Computing, Georgia Institute of Technology

{mudhakar, bgedik, lingliu}@cc.gatech.edu

Abstract

The peer to peer (P2P) file sharing systems such as Gnutella have been widely acknowledged as the fastest growing Internet applications ever. The P2P model has many potential advantages due to the design flexibility of overlay networks and the server-less management of cooperative sharing of information and resources. However, these systems suffer from the well-known performance mismatch between the randomly constructed overlay network topology and the underlying IP layer topology for packet routing. This paper proposes to structure the P2P overlay topology using a capacity-aware multi-tier topology to better balance load at peers with heterogeneous capacities and to prevent low capacity nodes from downgrading the performance of the system. An analytical model is developed to enable us to construct and maintain capacity-aware overlay topologies with good node connectivity and better load balance. To study the benefits and cost of the multi-tier capacity aware topology with respect to basic and advanced routing protocols, we also develop a probabilistic broadening scheme for efficient routing, which further utilizes capacity-awareness to enhance the P2P routing performance of the system. We evaluate our design through simulations. The results show that our multi-tier topologies alone can provide eight to ten times improvements in the messaging cost, two to three orders of magnitude improvement in terms of load balancing characteristics, and seven to eight times lower topology construction and maintenance costs when compared to Gnutella's random power-law topology.

1 Introduction

With applications such as Gnutella [7], KaZaA [9], and Freenet [6], 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. P2P systems can be broadly classified into two categories: Structured and Unstructured P2P systems. The primary focus of structured (distributed hash table based) P2P systems is on precisely locating a given object (identified by some unique ID) within a bounded number of hops. However, the main focus of unstructured P2P systems (Gnutella [7], KaZaA [9]) is to support approximate search and partial answers in a

typical distributed file sharing application. Many agree that unstructured P2P systems are more suitable for mass-market file sharing applications and thus there are increasing interests for mechanisms that can make unstructured P2P systems scalable [2].

Most of the decentralized unstructured P2P overlay networks, such as Gnutella and KaZaA, share two unique characteristics. First, the topology of the overlay network and the placement of files within the network are largely unconstrained. Second, queries are flooded via a *broadcast-styled routing (bsr)* algorithm across the overlay network with limited scope. Upon receiving a query, each peer sends a list of all content matching the query to the query originator node. In the *bsr* scheme, a query Q is specified by a quadruplet: $\langle originator, keywords, ID, TTL \rangle$, 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 has seen this query before by checking 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).

There are several serious performance problems with the use of such a random topology (RT) and its random broadcast based routing protocol. First, this approach does not distinguish peer heterogeneity in terms of their computing and communication capacity (such as access bandwidth, CPU, Disk I/O). It has been observed in [14] 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. Further, a weak peer on a path between two powerful peers throttles the throughput between the powerful peers. Second, the random topology combined with flooding-based routing is clearly not scalable since the load on each peer grows linearly with the total number of queries in the network, which in turn grows with the size of the system.

Much research has been dedicated to improving the performance of Gnutella-like P2P systems. Most of these research efforts can be classified into two categories: the search algorithm for P2P routing and the overlay topology. Most research efforts in the first category focus on improving the flooding-based search algorithm in Gnutella by introducing a guided search procedure with different biases (based on query latency, messaging bandwidth, node degree, the number of results returned, etc) [17]. However, most of these proposed routing or search algorithms disregard the natural peer heterogeneity present in most P2P systems, and more importantly the potential performance difficulty caused by the randomly constructed overlay topology.

Most of the work on improving the performance of the overlay topology for unstructured P2P systems can be categorized into two subdivisions: semantic-based and capacity-based. Several semantic overlay topologies [4] assume global knowledge of the semantic grouping of the shared files or the search queries. However, they do not address scalability of the P2P system with respect to physical IP-networks. It has been observed in [14] that peers are highly diverse in terms of their network resources and their participation times. Hence, constructing an overlay network randomly results in unstable and less powerful peers hindering the system's performance. Further, disrespecting peer heterogeneity overloads weaker peers, which in turn suffocates the performance of the P2P system. Unfortunately, there are only a couple of research efforts so far have devoted to improving topology of the overlay network using capacity-awareness.

The first such effort is the super-node architecture that is built into a popular P2P file sharing application – Kazaa [9]. It classifies nodes into two classes – the strong (high capacity) nodes and the weak (low capacity) nodes and enforces capacity-based restrictions on overlay network: (1) it disallows connections between any two weak nodes, and (2) it permits a weak node to be connected to exactly one strong node. A *supernodes* is a specially designated node that has higher bandwidth connectivity and it maintains an index of pointers to the data stored at each of its associated low capacity nodes. All queries are routed to supernodes. Several report that the supernode approach appears to offer better scaling than Gnutella. Many Gnutella clients have now implemented the supernode scheme, however its scalability has neither been measured or analyzed formally.

The second effort towards capacity-awareness is the recent proposal in [2], which uses the satisfaction based topology adaptation algorithm to ensure that most nodes are within constant reach of the strong nodes. However, the proposed topology adaptation scheme is ad-hoc in terms of maintenance cost and formation of small world groups and needs to pay the cost of maintaining a large number of neighbor nodes for those nodes that do not reach the level of satisfaction desired.

In this paper, we propose capacity-based techniques to structure the overlay topology with the aim of improving the overall utilization and performance by developing a number of system-level facilities.

- We propose a overlay network scheme with *Multi-Tier Capacity Aware Topologies*, capacityaiming at improving the network utilization, reducing the search latency, and improving the fault tolerance of the P2P system. We leverage the idea of exploiting node heterogeneity but making the selection of supernodes and the construction of the topology more generic. We present three classes of overlay topologies that can be viewed as an enhancement-cum-generalization over the super-node architecture. First, we permit multiple peer classes (as against just two classes in a super-node architecture), and second, we analyze and evaluate several interesting capacity-based restrictions on node connectivity.

- We develop *An analytical model* that enables us to construct multi-tier network-connection aware topologies such that the number of lookup queries served by a peer is commensurate with its capacity. Our analytical model ensures that the nodes with higher capacities are not overloaded and the nodes with low capacities stay connected.
- *Capacity guided bootstrap and topology maintenance service* that enables us to efficiently construct and maintain the overlay topology. In comparison with Gia [2], our techniques largely reduce the risk of disconnecting the overlay topology and incur significantly lower messaging overhead than Gia’s satisfaction-based topology adaptation algorithm.
- *A Probabilistic-Selective* routing technique, which employs a capacity-aware routing algorithm. On one end of the spectrum, our thiscapacity-aware routing algorithm represents a capacity-guided random walk [12] and on the other end of the spectrum, it represents the broadcast-styled flooding algorithm traditionally used by Gnutella.

We evaluate our design through simulation-based experiments. The results show that our multi-tier topologies alone can provide eight to ten times improvements in terms of messaging cost, two to three orders of magnitude improvement in terms of load balancing characteristics, and allows seven to eight times lower topology construction and maintenance costs when compared to Gnutella’s random power-law topology. We believe that efforts on bridging the gap (mismatch) between overlay networks and the underlying Internet topology will bring P2P services beyond pure “best effort” and closer to serious applications with respectable quality of service requirements.

2 Multi-Tier Overlay Topologies

Unstructured P2P networks such as Gnutella face a common problem – nodes can quickly become overloaded in the presence of high aggregate query rate, and the situation may be aggravated as the number of nodes in the system increases. As a consequence, the system utilization decreases dramatically. Therefore, our first design objective is to make the unstructured P2P networks more scalable by promoting the Capacity-Aware Topologies (CAT), which exploits peer heterogeneity with respect to bandwidth connectivity, CPU, and Disk I/O. Our second design objective is to enable the multi-tier topologies to scale well with increasing number of nodes in the system. To achieve such scalability we strive to design the CAT construction and maintenance algorithms with two aims. First, we want to avoid overloading any of the nodes by explicitly taking into account of their capacity constraints. Second, we want to keep nodes with low capacities stay connected.

We organize this section as follows. First we introduce the concept of heterogeneity level (HL) and then formally introduce the three classes of multi-tier capacity-aware topologies. To provide a qualitative comparison among

the three types of topologies, we develop a set of metrics to characterize their performance in terms of messaging bandwidth, load variance, query capacity, query latency, and fault-tolerance in terms of node connectivity. We then present an analytical model and algorithms for construction and maintenance of three classes of multi-tier topologies.

2.1 CAT: Capability-Aware Topologies

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 (HLs)* 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 $|HL(i) - HL(j)| \leq 1$.*

The fundamental question that first arises is: How to classify nodes into heterogeneity levels? In general, one could model the capability of a node as a *vector* consisting of several components including access network bandwidth, CPU speed, main memory size, disk access latency, etc. The capability vector is obtained based on the amount of resources a node is willing to *contribute* to the P2P system. Based on the capability vectors obtained from a statistically sampled set of nodes, one could cluster them into groups based on some similarity metric. For example, one could use the inverse of the Euclidian distance between two capability vectors as their similarity. It is observed in Gnutella [14] that nodes show several orders of magnitude difference in their access network bandwidth (slow dial-up connection to DSL connections to Cable connections to high speed LAN connections). Given the magnitude of difference in the access network capacities, one would anticipate that a node with a 1 Mbps DSL connection would be willing to contribute significantly more network bandwidth than a node with a 25Kbps dial-up connection. Hence, employing a simple k-mean based clustering algorithm would suffice to classify nodes into distinct classes. In the following sections of this paper, we assume that the set of node classes are given to us, and that each node class is associated with a scalar capability C (presumably based on access network bandwidth).

Classifying nodes into capability classes is vital for the performance of the P2P system for at least the following reasons: (i) Weak nodes are prevented from becoming hot-spots or bottle-necks that throttle the performance of the system, (ii) Avoiding bottle-necks decreases the average query response time as perceived by an end user, and (iii) From the perspective of the P2P system designer, *load balanced architectures* improve the overall throughput of the system. For instance, we can use this notion of heterogeneity level to avoid a weak node from suffocating the performance of the system as follows. Given the fact that we construct the overlay network by only connecting nodes of comparable *HLs*, we can reduce the load on weak nodes by constructing topologies such that the connectivity of

the nodes increases with their HL . Hence, a powerful node would maintain more connections and would receive more queries on an average when compared to a weak node. In short, our capability-aware overlay topologies achieve significant performance gains by getting more work done by the powerful nodes and reducing the workload on the weaker nodes.

2.2 Three Classes of Multi-Tier Topologies

We first formally define three classes of overlay topologies: *Hierarchical Topology*, *Layered Sparse Topology* and *Layered Dense Topology*. We characterize an overlay topology by the *type* of connections maintained by each node in the system. The significance of these classes will become more apparent when we explore their performance characteristics and topology construction and maintenance costs. Second, we present an analytic model for designing such capability-aware multi-tier topologies. Finally, we provide simple and pragmatic techniques to realize such topologies.

We now formally define and analyze the performance characteristics of these three classes of overlay topologies.

Definition Multi-Tier Topologies: Let $HL(p)$ denote the heterogeneity level of peer p . Let $maxHL$ denote the class of peers at the maximum heterogeneity level. A multi-tier topology is defined as a weighted, undirected graph $G:\langle V, E \rangle$ where V is the set of nodes and E is the set of edges such that the following conditions are verified:

1. Connectivity Preserving Rule: $\forall p \in V$, if $HL(p) < HL(maxHL)$ then $\exists q \in V$ such that $HL(q) = HL(p) + 1 \wedge (p, q) \in E$
2. $\forall (p, q) \in E$, assuming $HL(p) \leq HL(q)$ without loss of generality, we have
 - (a) Hierarchical Topology: $p, q \in maxHL \vee (HL(q) = HL(p) + 1 \wedge \nexists q' \neq q \text{ s.t. } (HL(q') \geq HL(p) \wedge (p, q') \in E))$
 - (b) Sparse Topology: $p, q \in maxHL \vee HL(q) = HL(p) + 1$
 - (c) Dense Topology: $p, q \in maxHL \vee HL(q) = HL(p) + 1 \vee HL(q) = HL(p)$

Informally, a hierarchical topology is a pure *tree-like* topology, except that the nodes in the highest HL are strongly connected (but not completely connected). An example 3-tiered hierarchical topology is shown in Figure 1. A sparse topology improves the fault-tolerance and reduces the diameter of the hierarchical topology by permitting a node to maintain connections with more than one node at its immediate higher level, as illustrated in Figure 2. A dense topology is equivalent to the basic multi-tier topology that adds no further restrictions on the connections between

nodes; see Figure 3. Note that constraint (1) in the definition of the topologies is very important to maintain the *connectivity* of the overlay network; while constraint (2) is responsible for the differences observed in the performance trends of these three classes of multi-tier overlay topologies.

2.3 Performance Characterization of Three Multi-Tier Topologies

2.3.1 Performance Metrics

We characterize the goodness of an overlay topology using the following five metrics.

Amortized messaging bandwidth: Amortized messaging bandwidth is measured as the ratio of the total *capacity-aware bandwidth (CAB)* consumed by a query to the number of results obtained. Capacity-Aware Bandwidth (CAB) is defined as the ratio of the total number of messages sent/received by a node to the node’s capacity. Note that in a homogeneous environment it would be reasonable to measure total bandwidth as the total number of messages sent by a node in the system. However in a heterogeneous environment, we have to distinguish between a message sent by a powerful node and a message sent by a weak node. Hence, we derive CAB by normalizing the total number of messages sent by a node with its capacity. Strictly speaking, we should measure bandwidth as number of messages per unit time. However, we just measure the number of messages (normalized by capacity) per query as bandwidth because the queries are associated with a mean query rate measured in *Queries Per Second (QPS)*.

Load Variance: Load on a node is measured as the ratio of the total number of messages sent/received by a node to its capacity, i.e., the amount of CAB expended by a node in the system. We characterize load distribution of a heterogeneous P2P system by the variance of load over all nodes in the system. Low load variance indicates that all nodes are loaded in proportion to their capabilities; this shows that our overlay topologies indeed solve the peer heterogeneity problem. Note that amortized messaging bandwidth is a per-query CAB requirement, while load is a per-node CAB requirement.

Coverage: Number of nodes that receive a broadcast query whose initial TTL was set to *initTTL*. Coverage is an

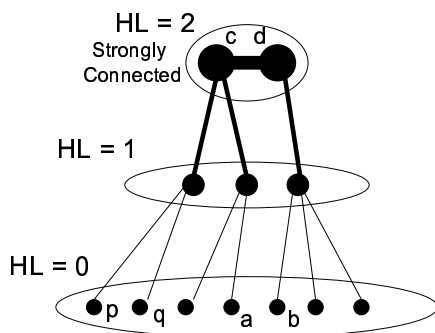


Figure 1: Hierarchical Topology

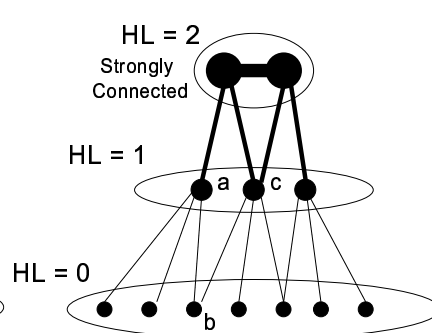


Figure 2: Sparse Topology

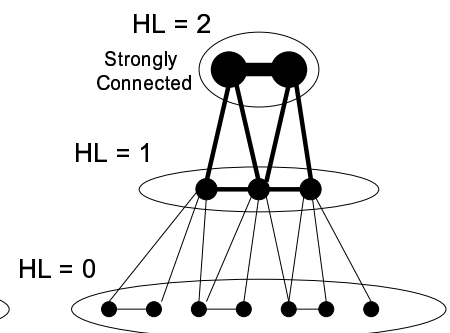


Figure 3: Dense Topology

indicator of the number of results that can be obtained for a search query. Coverage is a vital measure because, we should ensure that our efforts to classify nodes into classes and restricting their connectivity should not weaken the connectivity between different classes, or in the worst case, disconnect the overlay topology into multiple classes.

Amortized Latency: Ratio of total response time for a query to the number of results obtained. The amortized latency measure captures the response time for a search query as perceived by an end-user.

Fault-Tolerance: Fault-tolerance is measured as the fraction of the number of results obtained when a random set of nodes fail. In a fault-tolerant design, the performance of the overlay topology is not significantly affected by the failure of some random nodes in the system. Further, our measurement of fault-tolerance indicates the ability of our overlay topology to defend itself against certain kinds of Denial-of-Service (DoS) attacks by some malicious nodes in the system. For example, a denial of service attack by a node that does not process or route queries is equivalent to the failure of that node.

2.3.2 Performance Trends

Having formally defined the overlay topologies, we now discuss in depth their performance trends with respect to various performance metrics listed in section 2.3.1.

Amortized messaging 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 (recall Section 1). 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 broadcast-styled algorithm is used. However the increase in the number of duplicate queries in the sparse and the dense topology (due to redundant paths in the overlay topology) increases their amortized messaging bandwidth requirement. In comparison to a random topology, we achieve a huge bandwidth savings because our capability-aware overlay topologies ensure that the number of messages required to be handled by a node is commensurate with its capability. This largely reduces the load on weak nodes, and consequently, reduces the capability-aware bandwidth (CAB) requirement of our overlay topologies significantly.

Load Variance: 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 (or the dense) topology would display much better load distribution since the restriction on the type of connections maintained by a node is much weaker than that of a hierarchical topology (see Section 2.4). Nevertheless, our multi-tier topologies, by their very structure, ensure that the load on a node is commensurate with its capability, and hence show remarkably better load distribution when compared to a

random topology that is agnostic to peer heterogeneity; more importantly, uniform load distribution guarantee that our capability-aware overlay topologies do not overload the weak nodes.

Coverage: A hierarchical topology's tree-like structure increases the diameter of the overlay network, and 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, and thereby brings down the diameter of the overlay network. Also, by the structured nature of sparse and dense topologies, they reduce the number of duplicate queries, and hence achieve higher coverage, when compared to a random topology.

Amortized Latency: Latency (response time as perceived by an end-user) is much lower in the multi-tier topologies primarily because they largely avoid weak nodes from appearing on a overlay path between two powerful nodes. Assuming that the communication latency between two nodes is constrained by the weaker of the two nodes, the response time (and consequently amortized latency) in a random topology would be much higher than a multi-tier topology. Note that in any interactive system (directly involving the end-user) it is very crucial that the response time of the system remains sufficiently small.

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. 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 [14] 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. Based on our discussion in this section, we draw the following conclusions: (i) A hierarchical topology is best suited for minimizing the amortized messaging bandwidth consumption for any query Q since it largely eliminates duplicate queries (or redundant paths on the overlay topology), (ii) The dense topology is the best choice for minimizing load variance since it does not impose any heavy constraints on a node's neighbors, and (iii) The sparse and dense topologies show significantly higher fault-tolerance than the tree-like hierarchical topology. Further, in section 2.5 we show that the sparse topology is the best choice for minimizing topology construction and maintenance costs. Finally, based on our experimental results, we promote the sparse topology as the best overlay topology since it performs reasonably well with respect to most performance metrics.

2.4 Constructing CATs: Analytical Model

Having presented three classes of multi-tier capability-aware overlay topologies and analyzed their performance trends with respect to various performance metrics, we now turn our attention to constructing such overlay topologies. In this section, we present an analytical model that provides directives to construct an overlay topology that minimize the variance of CAB over all nodes in the system. This not only ensures that all nodes contribute equal amounts of CAB to the P2P system (thereby, eliminating bottle-necks in the system), but also significantly reduces the amortized messaging bandwidth requirement. We characterize the overlay topologies by its classes and the number of neighbors maintained by the nodes in each class. We present our model assuming a broadcast styled routing algorithm is used. However, the model can be easily modified to accommodate various other routing techniques including the iterative deepening technique [17] and the random walk technique [2]. It is important to note that our capability-aware overlay topologies reduce the per-query CAB requirement for most routing algorithms known in the literature.

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 . Note that $d_{numHL-1}^{up} = 0$ and $d_0^{down} = 0$.

The first observation that we make is that 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 derive the CAB requirement 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 CAB requirement on every node p in the system and its capability one can compute the variance of CAB, namely, $Var(CAB(p))$ over all nodes p . We resort to optimization techniques like simulated annealing [10] to obtain near optimal solutions for the degree parameters d_i^X that minimize the variance of CAB.

We estimate the CAB requirement at node p by counting the number of nodes whose queries would have to be processed by node p . For every query Q received by node p with $Q.TTL > 0$, the total number of messages sent/received by node p on query Q is $d_{HL(p)}$; if $Q.TTL$ equals zero, the node p expends its bandwidth only on

receiving the query Q . We first estimate the number of nodes at each HL that would receive a query Q sent by some node. We use this information to deduce the CAB requirement on every node in the system.

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

$$coverage_{lev}(hl, initTTL) = \begin{cases} 1 & \text{if } hl = lev \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

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

$$\begin{aligned} request_{lev}(hl, ttl) &= coverage_{lev}(hl + 1, ttl + 1) * d_{hl+1}^{down} \\ &+ coverage_{lev}(hl, ttl + 1) * d_{hl}^{in} \\ &+ coverage_{lev}(hl - 1, ttl + 1) * d_{hl-1}^{up} \end{aligned} \quad (3)$$

where $d_{numHL}^{down} = 0$ and $d_{-1}^{up} = 0$.

Among $request_{lev}(hl, ttl)$ 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 $notrcvd_{lev}(hl, ttl)$, can be computed as follows:

$$notrcvd_{lev}(hl, ttl) = N * f_{hl} - \sum_{i=ttl+1}^{initTTL} coverage_{lev}(hl, i) \quad (4)$$

Hence, the probability that a query Q with $Q.TTL = ttl$ reaches a new node at level hl is $prnew_{lev}(hl, ttl) = \frac{notrcvd_{lev}(hl, ttl)}{N * f_{hl}}$. Also, the number of queries Q that are forwarded to new nodes at level hl when $Q.TTL = ttl$ can be computed by $reqnew_{lev}(hl, ttl) = prnew_{lev}(hl, ttl) * request_{lev}(hl, ttl)$.

Now, $reqnew_{lev}(hl, ttl)$ queries are directed towards $notrcvd_{lev}(hl, ttl)$ nodes at level hl when $Q.TTL = ttl$. We have to estimate the number of distinct nodes (among $notrcvd_{lev}(hl, ttl)$ nodes) that indeed receive the query request. One can show that the number of distinct nodes that receive the query request at level hl with $Q.TTL = ttl$ is given by Equation 5,

$$coverage_{lev}(hl, ttl) = notrcvd_{lev}(hl, ttl) * \left(1 - \left(1 - \frac{1}{notrcvd_{lev}(hl, ttl)} \right)^{reqnew_{lev}(hl, ttl)} \right) \quad (5)$$

This completes the recursive definition of the *coverage* function. The Equations 3 to 5 recursively define the *coverage* function with the base case in Equation 2. Finally, the CAB requirement on some node p is computed

as the total number of messages to be sent/received by nodes at level $HL(p)$ divided over the number of nodes in level $HL(p)$ and normalized by capability $C_{HL(p)}$. Before that, we categorize the queries into those received with $Q.TTL > 0$ and $Q.TTL = 0$.

$$nummsg_{Q.TTL>0}(HL(p)) = d_{HL(p)} * \sum_{lev=0}^{numHL-1} \left(N * f_{lev} * \sum_{ttl=1}^{initTTL} coverage_{lev}(HL(p), ttl) \right) \quad (6)$$

and

$$nummsg_{Q.TTL=0}(HL(p)) = \sum_{lev=0}^{numHL-1} N * f_{lev} * coverage_{lev}(HL(p), 0) \quad (7)$$

Hence, one can obtain $CAB(p)$ from Equation 8

$$CAB(p) = \lambda * \frac{nummsg_{Q.TTL>0}(HL(p)) + nummsg_{Q.TTL=0}(HL(p))}{N * f_{HL(p)} * C_{HL(p)}} \quad (8)$$

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 constrains d_i^{down}) as inputs and outputs the variance of load distribution. One can resort to optimization techniques like simulated annealing [10] to obtain near optimal solutions (for d_i^X) that minimize the variance of load distribution. Further, it is straightforward to incorporate scenarios wherein the rate at which a node n issues queries depends on its heterogeneity level $HL(n)$; say a node n issues queries at rate $\lambda_{HL(n)}$ and quite possibly, $\lambda_{hl} > \lambda_{hl-1}$.

2.5 Topology Construction and Maintenance

Having developed an analytical model for structuring the overlay topology, we explore pragmatic techniques to construct and maintain such CATs. The multi-tier topology construction consists of two main components: (i) bootstrapping a new node, and (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 network, we require the bootstrap server to provide it with an appropriate

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 1: Bootstrapping node p

set of nodes q (depending on $HL(p)$) according to Table 1 and the node degree information required to be maintained by node p according to the topology degree parameters, namely $d_{HL(p)}^X$, $X \in \{up, down, in\}$. Note that the optimal value for d_i^X may not be an integer. To fix this problem, nodes at level i maintain $d_i^X - \{d_i^X\}$ connections with

probability $1 - \{d_i^X\}$ and $1 + d_i^X - \{d_i^X\}$ connections with probability $\{d_i^X\}$, where $\{d_i^X\}$ denotes the fraction part of d_i^X .

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 access network 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. Assume that a node n 's life time is exponentially distributed with mean $\frac{1}{\mu_{HL(n)}}$. One can show that probability that a node n loses a connection with a node m because of node m leaving the system is $\frac{\mu_{HL(m)}}{\mu_{HL(n)} + \mu_{HL(m)}}$. Hence, probability that a node n loses a connection with another node at the same level during its life time is $\frac{1}{2}$, while that with a node at a lower level than node n is greater than $\frac{1}{2}$ and that with a node at a higher level than node n is much lesser than $\frac{1}{2}$.

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. Recall that a node only needs to maintain the required number of connections with other nodes at the same or higher levels. Observe that a hierarchical topology has higher maintenance cost than a sparse topology because the failure of any node would entirely disconnect the sub-tree rooted at that node. Also, observe that the maintenance cost of a dense topology is higher than that of a sparse topology because a node in a dense topology would have to maintain connections with other nodes at the same level (the probability of loosing such a connection in a node's lifetime is about $\frac{1}{2}$). Clearly, the sparse topology emerges as the winner with respect to topology maintenance cost.

Discussion. An implicit assumption in dividing peers into capability-aware classes is that each class is significantly populated. Although, neither our analytical model nor topology construction and maintenance algorithms put forth such a requirement, we experimentally observed that it is in general hard to construct and maintain topologies wherein some peer class is scantily populated (<2% of the total population). Under such scenarios, we had to merge a thinly populated peer class with one if its adjacent classes.

Observe that at one extremity, wherein all the peers are homogeneous, our overlay topology falls back into a random topology. But our node degree restrictions guarantee that our overlay topology does not follow a power-law distribution; and hence it does not suffer from weak connectivity. Another special case of our capability-aware

topologies is the super-node architecture, which is a two-tiered hierarchical topology. From our analysis and experimentation we observed that an enhanced super-node topology (a two-tiered sparse topology) performs significantly better with respect to several performance metrics.

3 CAR: Capability-Aware Routing

3.1 Design Ideas

Recall the broadcast styled routing (*bsr*) technique 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. Several threshold based iterative algorithms have been proposed to improve the performance of the routing algorithm. For example, 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) [17]. 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 routing 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 [14]. 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 selective routing (psr)* 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 routing 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 selective routing algorithm comprises of three major components: *Ranking neighbor peers, Pro-*

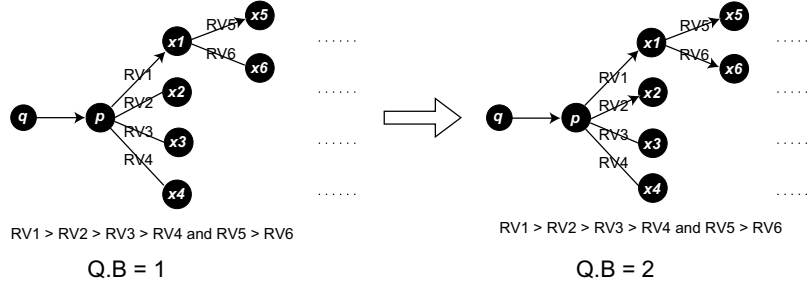


Figure 4: Probabilistic Selective Algorithm: Illustration

cessing 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 [16] for the algorithm details.

3.2 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 access 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.

In addition to using each metric independently as a ranking criterion, one could use a weighted combination of the above metrics 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 (or comparable) performance. However, the connectivity information (Max Degree) is inbuilt in our multi-tier topologies; hence Max Results and Max Rate show better performance on capability-aware topologies. Hence, without loss of generality, in the rest of the paper we use Max Results as the metric for all discussions.

3.3 Probabilistic-Selective Routing

Query processing using the probabilistic selective routing 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 4 illustrates our algorithm at peer p for $Q.B = 1$ and $Q.B = 2$ with ranking values $RV1 > RV2 > RV3 > RV4$ and $RV5 > RV6$.

3.3.1 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 $X = \{x_1, x_2, \dots, x_n\}$ as $x_1 \succ x_2 \succ \dots \succ x_n$. 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, \dots, x_b at time t' ($t' > t$). Consequently, the ranking measures of peers $x_{b+1}, x_{b+2}, \dots, x_n$ 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 $x_{b+1}, x_{b+2}, \dots, x_n$ are not considered subsequently.

To capture the dynamics of the P2P network we modify the *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*.

3.4 Discussion

In several respects, one could view the probabilistic-selective routing algorithm (*psr*) as a generalization of the biased random-walk [12] and the basic broadcast-styled algorithm. For example, with $minB = maxB = 1$, the *psr* algorithm reduces to a biased random-walk; with $minB = maxB = \infty$, the *psr* algorithm reduces to a broadcast-styled algorithm. In fact, the *psr* algorithm enjoys the benefits of both the biased random-walk and the broadcast-styled algorithm. For popular documents, *psr* is equivalent to a biased random-walk, and hence consumes significantly lesser network bandwidth than a broadcast-styled algorithm; for unpopular (rare) documents, the *psr* algorithm is almost equivalent to a broadcast-styled algorithm, and hence incurs significantly lower latency than the biased random-walk algorithm.

4 Experimental Results

We have performed three sets of simulation-based experiments. The first set compares the three classes of multi-tier topologies against a random topology using our performance metrics. The second set evaluates our model for constructing multi-tier capability-aware topologies against a random topology. The third set studies the effective-

<i>Parameter</i>	<i>Description</i>	<i>Default</i>
N	Number of Nodes	10,000
HL	Number of Peer Classes	3
C	Node capability	8:4:1
$Maxd$	Maximum Peer Degree	10
$Mind$	Minimum Peer Degree	1
Dd	Degree Distribution (Random/Power Law)	Power Law

Table 2: Network Generator

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

Table 3: Document Generator

<i>Parameter</i>	<i>Description</i>	<i>Default</i>
THR	Result threshold	32

Table 4: Search Simulator

ness of probabilistic routing in handling vastly heterogeneous nodes and trade-offs in probabilistic-selective routing algorithm.

4.1 Simulation Set Up

We implemented our simulator using a discrete event simulation [5] model. The end users send queries to the system at an exponentially distributed rate λ . The latency of a link from node n to node m is modeled as being inversely proportional to $\min(C_{HL(n)}, C_{HL(m)})$. The processing time of a query at a node is modeled as a constant; especially since the amount of computation power available to nodes is almost homogeneous. Our simulation setup comprises of three main modules: the Network Generator, the Document Generator, and the Routing Simulator. Tables 2, 3 and 4 present the major tunable parameters used in these modules. In all these modules we chose the parameter settings from several observations made on the real Gnutella network [14, 2].

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 20%, 70% and 10% 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 $\alpha = 1$). 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).

Routing Simulator. The routing simulator implements one of the following routing algorithms: broadcast styled routing and probabilistic-selective routing.

4.2 Evaluation of CATs

We compare the performance of a hierarchical, sparse, dense and random overlay topology using five performance metrics: Amortized messaging bandwidth, Load Distribution, Coverage, Amortized Latency, and Fault Tolerance using the broadcast style routing algorithm with $initTTL$ equal to 7. We conclude this section with an experiment that shows the importance of identifying the correct number of heterogeneity levels (peer classes) in obtaining good performance.

Amortized messaging bandwidth. Figure 5 presents the amortized messaging bandwidth consumption on various topologies. For $N = 10,000$ nodes, the hierarchical topology consumes the least amortized messaging bandwidth since its tree-like structure minimizes the number of duplicate queries. Also, the multi-tiered sparse and dense topologies reduce the CAB requirement on nodes by about **6 to 10 times** when compared to a random topology because they require that the weak nodes send/receive far fewer messages when compared to strong (more capable) nodes. This clearly demonstrates our claim that capability-aware topologies indeed reduce the amortized messaging bandwidth requirement significantly.

Load Distribution. Figure 6 shows the variation of load distribution among the four overlay topologies that are normalized with respect to a $N = 1,000$ node random topology. For $N = 10,000$ the sparse and the dense topology show **80 to 100 times** lower variance, while a hierarchical topology shows 10 times lower variance. The hierarchical topology pays for the strict constraints it imposes on the overlay topology ($d_i^{up} = 1$ and $d_i^{in} = 0$), while the random topology pays for the anarchy in its overlay topology. Uniform load distribution properties of capability-aware topologies ensure that the weak nodes do not become bottle-necks and throttle the performance of the system.

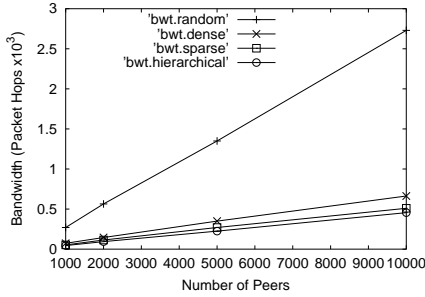


Figure 5: Amortized messaging bandwidth

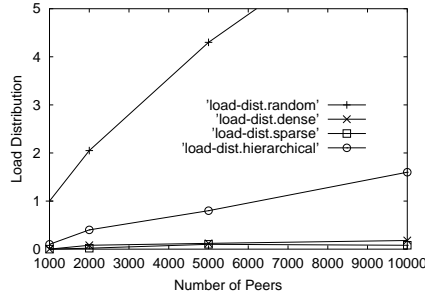


Figure 6: Variance of load/capability

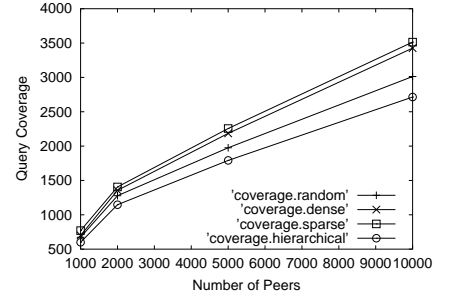


Figure 7: Query Coverage

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 dense and sparse topologies show 20% more coverage because, by their structured nature they largely reduce the number of duplicate queries; while the hierarchical topology shows 15% lower coverage because of its increased diameter. Table 5 shows the variation of coverage with different HL s 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). Further, this property of *fairness* acts as an important motivation for the high capability nodes to do more work for the system.

Amortized Latency. Figure 8 shows the amortized latency (equivalently, the response time experienced by an end-user) of the four overlay topologies. Recall that we had, for simplicity, modeled the latency of a link from node n to node m to be inversely proportional to $\min(C_{HL(n)}, C_{HL(m)})$. In a random topology, the presence of many weak nodes on the paths between two powerful nodes is mainly responsible for its latency to be about twice as large as that of capability-aware topologies. Further, among the capability-aware topologies, the hierarchical topology shows about 5% and 12% lower latency than the sparse and the dense topologies; one could attribute this trend to the increasing *randomness* in the topology from a hierarchical topology to the sparse and the dense topologies.

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 5: Coverage vs HL for $N = 10,000$

Fault Tolerance. We study the fault-tolerance of the four topologies under two conditions: *Uniform Faults* and *Non-Uniform Faults*. Figure 9 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 because of its delicate tree-like structure; while the sparse and the dense topologies exhibit only 2-3% lower fault-tolerance because they assign more responsibility to higher level nodes; recall that in a random topology all nodes are treated as the same and hence are assigned equal responsibilities.

Figure 10 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 4:3:1 [2, 14]. For $N = 10,000$ nodes, the sparse and dense topology show about 4-5% more fault-tolerance than the random topology and the hierarchical topology shows 20% improvement as against the uniform faults case. This improvement is primarily because the multi-tiered topologies assign *more responsibility* to more capable nodes which are *less likely to fail* (higher level nodes).

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

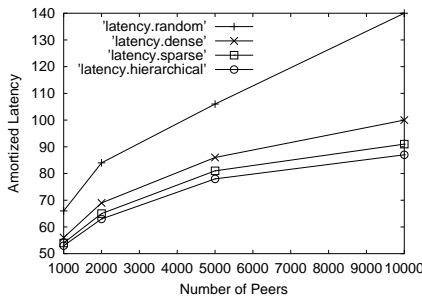


Figure 8: Amortized Latency

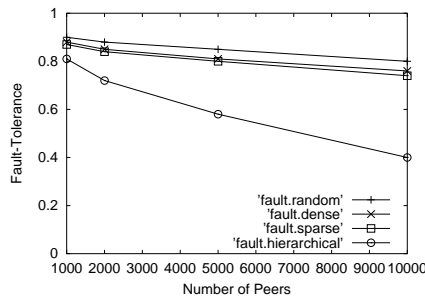


Figure 9: Fault Tolerance with uniform faults

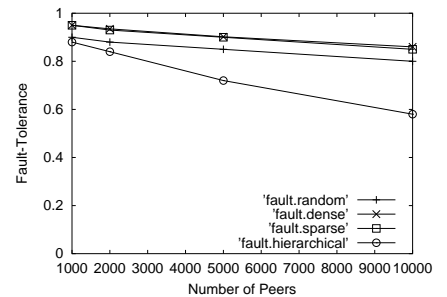


Figure 10: Fault Tolerance with non-uniform faults

Topology Type	Coverage	Amortized Bandwidth	Amortized Latency
Hierarchical	0.97	3.06	0.96
Sparse	0.99	3.49	1.16
Dense	0.99	3.33	1.12
Topology Type	Fault-Tolerance (non-uniform faults)	Load Distribution	
Hierarchical	0.96	21.0	
Sparse	0.82	11.0	
Dense	0.81	9.0	

Table 6: Importance of choosing correct $numHL$

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 $numHL = 2$ (by merging the top two peer classes) and $numHL = 3$. Table 6 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, **3.5 times** more bandwidth (CAB), 16% more latency, 18% lesser fault-tolerant, and **11 times** more variance in load distribution than a sparse topology with $numHL = 3$. Nevertheless, observe that the results obtained for $numHL = 2$ is better than a random topology because the former is *more* capability-aware.

4.3 Topology Construction and Maintenance

In this section we evaluate our techniques to realize a multi-tier capability-aware topology. First, we evaluate the efficacy of our bootstrapping algorithm in constructing an overlay network that accurately reflects the results obtained from our analytical model. Second, we study the goodness of our topology maintenance algorithm along two dimensions: (i) performance in terms of its messaging overhead, and (ii) efficacy in terms of the connectivity of the overlay topology.

Bootstrapping. Table 7 shows the *absolute error* and the *relative error* (within brackets) in the node loads obtained using our bootstrap algorithm and the results obtained from our analytical model. Absolute error in the CAB requirements on nodes is computed as $Var(CAB_{computed} - CAB_{experimental})$, where $CAB_{computed}$ denotes the CAB requirement on a node as obtained from the analytical model and $CAB_{experimental}$ is the load on the same node as obtained from our simulation results. Relative error is computed as $Var(\frac{CAB_{computed} - CAB_{experimental}}{CAB_{experimental}})$. We compute the bootstrapping error along three dimensions, namely, the number of nodes, the number of levels and fraction of nodes at a randomly chosen level. In our experiment, we incrementally added one node at a time using our bootstrap algorithm to construct a 10,000 node three-tiered topology. The error margins show the deviation in the CAB variance of a multi-tier topology constructed using our bootstrapping strategy as against the optimal value obtained from our analytical model. Hence, low error margins indicate that our bootstrap algorithm is capable of

Parameter	1	3	5	7	10
# nodes $\times 10^3$	1.2 (0.00400)	1.4 (0.00356)	2.0 (0.00523)	3.3 (0.00671)	4.9 (0.00576)
# levels $\times 1$	1.03 (0.00252)	1.09 (0.00214)	1.4 (0.00332)	3.2 (0.00299)	4.4 (0.00278)
# fraction of nodes $\times 0.1$	5.8 (0.01434)	3.2 (0.00742)	1.2 (0.00219)	2.9 (0.00347)	6.7 (0.00421)

Table 7: Bootstrapping Error

constructing topologies that are near optimal with respect to minimizing CAB variance. It would be incorrect to entirely attribute this level of accuracy in topology construction to the bootstrapping algorithm; instead, it is the simplicity in the design of our overlay topologies that makes it possible to efficiently and precisely construct them.

Topology Maintenance. We measured deviation from the optimal CAB variance values (from our analytical model) while attempting to maintain our multi-tier topologies. We observed that the errors due to topology maintenance are almost equivalent to the bootstrapping errors; see Table 7.

Figure 11 shows the topology maintenance cost versus the rate at which nodes join or leave the system. The *bootstrap-server* technique sends a request to the bootstrap server each time a node needs to open new connections. For obvious reasons, the bootstrap-server technique is the cheapest but incurs heavy overheads on the bootstrap server. In the *distributed-maintenance* technique a node maintains a cache of other nodes at levels higher than itself. Note that such a cache can be constructed easily as nodes are anyway in constant interaction with each other. When a node n needs to open a connection to say, some node at level $HL(n) + 1$, it pings those nodes at level $HL(n) + 1$ in its cache and opens a connection with one of them. Observe from Figure 11, the dense topology pays for maintaining volatile links with nodes at the same level, while a sparse or a hierarchical topology benefit because the nodes have to maintain their connections only with strictly higher level nodes (see section 2.5). Note that the maintenance cost for hierarchical topologies remains low, although they run into the risk of disconnecting an entire sub-tree on the failure of a single node; this is because our experiments let the root node of such a disconnected sub-tree to directly contact the bootstrap server to bootstrap itself (and consequently the entire sub-tree) into the system.

We also measured the effectiveness of the topology construction and maintenance algorithm by measuring the *connectedness* of the overlay topology. The disconnectedness of the overlay topology is measured as the number of disconnected components in the overlay topology. Figure 12 shows the variation of disconnectedness of our capability-aware topology (CAT) and a random topology with time. Observe that the connectedness of multi-tier topologies is significantly higher than a random topology. It has been observed that a random topology formed by Gnutella is in fact a power-law topology and a few powerful nodes are responsible for keeping the overlay topology connected. On contrary, our structured overlay topologies are systematically constructed and hence maintain a highly connected overlay topology.

4.4 Evaluation of CAR

In this section we present a detailed evaluation of our capability-aware routing algorithm. We first show the benefits of CAR on both a random topology and a multi-tier topology. Second, we compare our routing algorithm with other routing algorithms popularly known in literature, such as, iterative deepening (*rdfs*) [17] and broadcast-styled routing (*bsr*). Third, we show the ability of our routing algorithm to adapt itself to the dynamics of the P2P network.

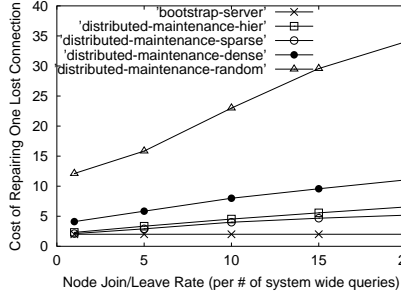


Figure 11: Topology Maintenance Cost

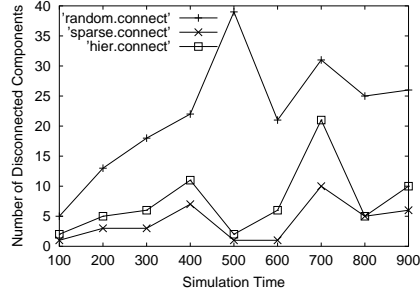


Figure 12: Topology Connectivity

Performance Enhancements by CAR. Figure 13 shows the mean bandwidth (CAB) expended for the execution of a query. For a random topology, CAR shows about 30% reduction in bandwidth, while for a capability-aware multi-tier topology, CAR shows about 35% reduction in network bandwidth. The capability-aware query routing algorithm ensures that the query is delivered to nodes that are more capable of obtaining results for the query. Further, in a capability-aware topology, the queries are not only routed through powerful nodes in terms of their network bandwidth, but also selectively routed to nodes that are capable of generating more results for the query. However, one should keep in mind that the bandwidth improvements obtained by CAR is at the cost of introducing a skew in the load distribution; however, from our experiments we observed that typically CAR limits the CAB variance to less than twice of the optimal value obtained from our analytical model.

Comparison of Routing Algorithms. Table 8 shows the performance of several routing algorithms over a random topology and a three-tiered sparse topology for several values of the search threshold. The performances of the routing algorithms are measured in terms of amortized messaging bandwidth (based on CAB) and amortized latency (shown within brackets). Note that PSR denotes our probabilistic-selective routing algorithm, ITR-DEEP denotes the iterative-deepening technique [17], BIASED-RW denotes a biased random-walk algorithm [12] and BSR denotes the broadcast-styled routing algorithm. In general the capability-aware topologies (CAT) consume about 5-7 times lower amortized messaging bandwidth than the random topologies (RT). Observe that the biased random-walk algorithm, in general, consumes the least amortized messaging bandwidth. For low values of the search threshold the *psr* algorithm is close to the biased random-walk algorithm. However, for higher values of threshold, the biased random-walk algorithm imposes enormous latency; however, *psr* incurs much lower latency, at the cost of increased amortized messaging bandwidth.

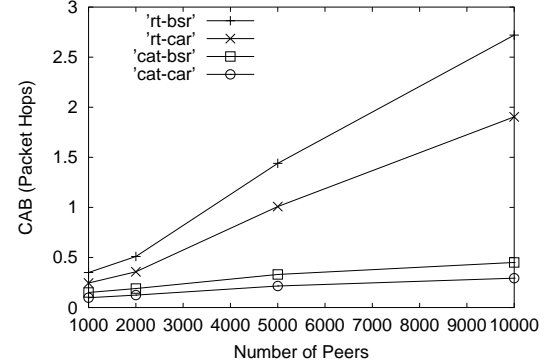


Figure 13: CAR Benefits

Search THR	CAT-PSR	CAT-ITR-DEEP	CAT-BIASED-RW	CAT-BSR	RT-PSR	RT-ITR-DEEP	RT-BIASED-RW	RT-BSR
1	2.11 (7.40)	1.75 (1.75)	1.45 (1.82)	5242 (24.03)	7.53 (10.53)	6.33 (2.54)	4.62 (3.69)	9031 (35.87)
5	7.50 (8.41)	8.53 (8.87)	7.03 (7.95)	5242 (24.03)	35.15 (12.65)	42.17 (13.74)	20.53 (15.07)	9031 (35.87)
10	15.63 (12.73)	17.94 (13.73)	14.00 (20.83)	5242 (24.03)	68.39 (18.21)	78.97 (19.83)	56.74 (30.85)	9031 (35.87)
20	31.01 (18.23)	36.34 (20.01)	27.05 (50.34)	5242 (24.03)	130.63 (26.74)	150.33 (30.15)	104.98 (61.32)	9031 (35.87)
30	50.93 (23.06)	62.93 (24.62)	40.83 (100.53)	5242 (24.03)	200.73 (32.53)	240.93 (33.83)	154.35 (123.65)	9031 (35.87)

Table 8: Comparison of Routing Algorithms: $N = 10,000$, $numHL = 3$ and $initTTL = 7$

Note that in the above experiments the average number of results returned by the CAT-BSR is about 25. We observed that when the threshold is set significantly higher than the average number of results return by BSR, the threshold-based algorithms (PSR, ITR-DEEP and BIASED-RW) perform poorer than BSR. This is because the threshold-based algorithms were designed with a 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 (recall the discussion in section 3). Also, the focused/biased nature of the PSR algorithm is responsible for its performance to be better than that of the ITR-DEEP algorithm.

5 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 [17, 3, 12]. These papers suggest enhancements over the naive flooding-based 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 [15]. Suggestions have been made to implement caching as a part of Gnutella protocol [11].

Several have pointed out that peer heterogeneity would be a major stumbling block for Gnutella [13, 14]. Solutions based on super-peer architectures have been proposed in [8] 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-tiered 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.

Chawathe *et al* [2] suggest the use of dynamic topology adaptation that puts most nodes within short reach of

high capacity nodes. Their topology adaptation scheme defines a level of satisfaction and ensures that high capacity nodes are indeed the ones with high degree and that low capacity nodes are within short reach of higher capacity ones. However, it constructs and maintains topology using ad hoc mechanisms which not only increases the topology maintenance cost (to maintain the level of satisfaction for every node) but also runs into the risk of building disconnected topologies, each of whose components are like the *small world groups* observed in Gnutella. Also, the performance gains reported in [2] are primarily due to the fact that they use one hop replication and allow nodes to maintain as many as 40-60 neighbors. From well-known observations on Gnutella, it is observed that even the powerful nodes maintain only about 10 neighbors. Hence, we believe that maintaining such a large number of connections on behalf of one application is unreasonable. Further, our topologies are structured precisely using the analytical model and incur very little construction and maintenance costs. In addition, our techniques guarantee that the overlay topology is well-connected and avoid the formation of small-world groups.

6 Conclusion

The key problem that has plagued a Gnutella like P2P systems is *Peer Heterogeneity*. We have proposed simple yet effective multi-tier capacity-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. There are three main contributions in this paper. First, we propose techniques to structure overlay topologies taking peer heterogeneity into account; such capacity aware topologies ensure that the performance of the P2P system is not hindered by less powerful peers. Second, We developed an analytical model to enable us to construct and maintain capacity-aware overlay topologies with good node connectivity and better load balance. Third, we proposed a probabilistic routing algorithm that further reduces the bandwidth consumption of our P2P system. We used extensive simulation-based experiments and mathematical analysis to construct and evaluate the goodness of capacity-aware topologies and routing algorithms over random topologies and broadcast-styled routing algorithms that disregard peer heterogeneity. 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] Y. Chawathe, S. Ratnaswamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM 2003*, 2003.
- [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] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University, October 2002., 2002.
- [5] G. S. Fishman. Discrete-event simulation. Springer Series in Operations Research.
- [6] Freenet. The free network project. <http://freenet.sourceforge.net/>, 2000.
- [7] Gnutella. The gnutella home page. <http://gnutella.wego.com/>, 2002.
- [8] F. S. Inc. Super-peer architectures for distributed computing. <http://www.fiorano.com/whitepapers/superpeer.pdf>, 2002.
- [9] Kazaa. Kazaa home page. <http://www.kazaa.com/>, 2003.
- [10] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 1983.
- [11] LimeWire. Improving gnutella protocol: Protocol analysis and research proposals. http://www9.limewire.com/download/ivkovic_paper.pdf, 2002.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. <http://www-2.cs.cmu.edu/~kunwadee/research/p2p/paper.html>, 2001.
- [16] 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.
- [17] 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.