

Large Scaling Unstructured Peer-to-Peer Networks with Heterogeneity-Aware Topology and Routing

Mudhakar Srivatsa, *Student Member, IEEE*, Buğra Gedik, *Member, IEEE*, and
Ling Liu, *Senior Member, IEEE*

Abstract—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, including high flexibility and serverless management. However, these systems suffer from the well-known performance mismatch between the randomly constructed overlay network topology and the underlying IP-layer topology. This paper proposes to structure the P2P overlay topology using a heterogeneity-aware multitier topology to better balance the load at peers with heterogeneous capacities and to prevent low-capability nodes from throttling the performance of the system. An analytical model is developed to enable the construction and maintenance of heterogeneity-aware overlay topologies with good node connectivity and better load balance. We also develop an efficient routing scheme, called probabilistic selective routing, that further utilizes heterogeneity-awareness to enhance the routing performance. We evaluate our design through simulations. The results show that our multitier topologies alone can provide eight to 10 times improvement in the messaging cost, two to three orders of magnitude improvement in terms of load balancing, and seven to eight times lower topology construction and maintenance costs when compared to Gnutella's random power-law topology. Moreover, our heterogeneity-aware routing scheme provides further improvements on all evaluation metrics, when used with our heterogeneity-aware overlay topologies.

Index Terms—Peer-to-peer systems, overlay topology, overlay routing, node heterogeneity, load balancing.

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. We see two forces pushing distributed middleware technologies that allow decentralized peer-to-peer applications to scale to a large and continuously growing community of users. First, there is an increasing demand for Internet users to collaborate and share information across multiple administrative domains. Second, P2P middleware makes it possible to harvest an immense amount of underutilized resources over the Internet in a cooperative manner.

Unlike traditional distributed computing, P2P networks aggregate large number of computers and possibly mobile or handheld devices, which join and leave the network frequently. Nodes in a P2P network, called peers, play a variety of roles in their interaction with other peers. When accessing information, they are clients. When serving information to other peers, they are servers. When forwarding information for other peers, they are routers. This new breed of systems creates application-level virtual networks with their own overlay topology and routing protocols. An overlay topology provides mechanisms to create and maintain the connectivity of an individual node

to the network by establishing network connections 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 administration costs, and transparent and dynamic operability, there are some well-known problems with most of the current P2P systems. For the P2P paradigm to be adopted widely, the technology must meet several challenges. We describe the general functioning of unstructured P2P networks before discussing their drawbacks that form the source of these challenges.

Decentralized, unstructured P2P overlay networks, such as Gnutella, 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 contents matching the query to the query originator node. In the *bsr* scheme, a query Q is specified by a quadruplet $\langle \text{originator}, \text{keywords}, \text{ID}, \text{TTL} \rangle$, where $Q.\text{originator}$ is the query originator, $Q.\text{keywords}$ is the list of user supplied keywords, $Q.\text{ID}$ is a unique query identifier, and $Q.\text{TTL}$ is the Time-to-Live of the query. The query originator assigns the query Q a unique ID ($Q.\text{ID}$) and sets the scope of the query $Q.\text{TTL}$ to initTTL . When a peer p receives a query Q from any neighbor peer q , peer p checks if it is a *duplicate* query (a node stores all identifiers $Q.\text{ID}$ received in the last max_delay time units for this purpose, where max_delay is set to be larger than the time it takes for a query to complete

• M. Srivatsa and L. Liu are with the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332. E-mail: {mudhaker, lingliu}@cc.gatech.edu.

• B. Gedik is with the IBM T.J. Watson Research Center, 1133 Westchester Avenue, White Plains, NY 10604. E-mail: bgedik@us.ibm.com.

Manuscript received 20 July 2004; revised 1 Apr. 2005; accepted 16 Sept. 2005; published online 26 Sept. 2006.

Recommended for acceptance by R. Schlichting.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0176-0704.

```

PROCESSQUERY(Query  $Q$ , Peer  $p$ , Peer  $q$ )
(1) if  $isDuplicate(Q.ID)$ 
(2)   Drop the query  $Q$ 
(3)   return
(4)    $X = \{x_1, x_2, \dots, x_n\} \leftarrow Neighbors(p) - \{q\}$ 
(5)   if  $Q.TTL = 0$ 
(6)     Drop the query  $Q$ 
(7)   else
(8)      $Q.TTL \leftarrow Q.TTL - 1$ 
(9)     Forward  $Q$  to peers in  $X$ 
(10)   $LR$ : Local Results - Matching files in peer  $p$  for query  $Q$ 
(11)   $LR \leftarrow p.matches(Q.keywords)$ 
(12)  Send  $LR$  to peer  $q$ 

```

Fig. 1. Query processing.

its broadcast). If Q is a duplicate, peer p drops the query; 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 1 to all its neighbors (except peer q). Fig. 1 provides a sketch of query processing in broadcast-style routing.

There are several serious performance problems with the use of a random topology (RT) and a broadcast-based routing protocol. First, this approach does not distinguish peers in terms of their computing and communication capabilities (such as access bandwidth, CPU, disk I/O). It has been observed in [15] 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. Second, the random topology combined with a broadcast-based routing scheme 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. As a result, there is a need to develop topologies that do not share the shortcomings of random topologies but retain their flexibility and maintainability. Moreover, in order to best utilize such topologies, we need to develop intelligent query routing mechanisms that exhibit low messaging cost.

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: 1) P2P search and routing algorithms and 2) P2P overlay topologies. 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.) [20], [3]. 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 impediment 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 capability-based. 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 peer heterogeneity. It has been observed in [15] 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. Unfortunately, there are only a couple of research efforts so far, that are devoted to improving topology of the overlay network using heterogeneity-awareness.

The first such effort is the supernode architecture that is built into a popular P2P file sharing application Kazaa [7]. It classifies nodes into two classes, namely, strong (high capability) nodes and weak (low capability) nodes and enforces capability-based restrictions on the 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 *supernode* is a specially designated node that has higher bandwidth connectivity and maintains an index of pointers to the data stored at each of its associated low capability nodes. All queries are routed via the supernodes. Several authors have reported that the supernode approach appears to offer better scalability than the random topology. Many Gnutella clients have now implemented the supernode scheme; however, its generalizations have neither been studied nor analyzed formally.

The second effort toward heterogeneity-awareness is the recent proposal in [2], which uses a satisfaction-based topology adaptation algorithm to ensure that most nodes are within the reach of the strong nodes. However, the proposed topology adaptation scheme is ad hoc in terms of its maintenance cost and the formation of small world groups. Further, it requires nodes that have not reached the desired level of satisfaction to maintain a large number of neighboring nodes.

In this paper, we propose capability-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 an overlay network scheme with *Multitier Heterogeneity-Aware Topologies*, aiming at improving the network utilization, reducing the search latency, and improving the fault tolerance of the P2P system. We leverage node heterogeneity by 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 supernode architecture.
- We develop *an analytical model* that enables us to construct multitier network-connection-aware topologies such that the number of lookup queries served by a peer is proportional to its network capacity.
- *Capability guided bootstrap and topology maintenance service* 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* employs a heterogeneity-aware routing algorithm. On one end of the spectrum, our heterogeneity-aware routing algorithm represents a capability-guided random walk [10] 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 multitier topologies

alone can provide eight to 10 times improvements in terms of messaging cost, achieve two to three orders of magnitude improvement in terms of load balancing characteristics, and incur 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 the 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.

We organize the rest of this paper as follows: Section 2 introduces our overlay topologies and presents a detailed analysis with respect to various metrics. In Section 3, we present our search technique, describing in detail its different variations and policies. The overlay topologies and the search techniques are then evaluated in Section 4, showing the effectiveness of these techniques in comparison with other existing approaches. We present related work in Section 5 and conclude the paper in Section 6.

2 MULTITIER HETEROGENEITY-AWARE OVERLAY TOPOLOGIES

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

We organize this section as follows: First, we introduce the concept of heterogeneity class (*HC*) and, then, formally introduce the three classes of multitier heterogeneity-aware topologies. To provide a qualitative comparison among the three types of topologies, we develop a set of metrics that characterize their performance in terms of messaging bandwidth, load variance, query capability, query latency, and fault-tolerance. We then present an analytical model and algorithms for construction and maintenance of multitier topologies.

2.1 HAT: Heterogeneity-Aware Topologies

We classify nodes into *Heterogeneity Classes (HCs)* based on their *capabilities*, with class zero used to denote the *least powerful* set of nodes. Peers at the same heterogeneity class are assumed to be *almost* homogeneous in terms of their capability. The key idea is to ensure that two nodes whose *HCs* vary significantly are not directly connected in the overlay network. So, we *do not allow a connection between two nodes i and j if $|HC(i) - HC(j)| > 1$* , where $HC(x)$ denotes the heterogeneity class of node x .

One fundamental question that first arises is how to classify nodes into heterogeneity classes. 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 [15] 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 25 Kbps 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 determined by a method like this 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: 1) Weak nodes are prevented from becoming hot-spots or bottlenecks that throttle the performance of the system, 2) avoiding bottlenecks decreases the average query response time as perceived by an end user, and 3) 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 class to prevent 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 *HCs*, we can reduce the load on weak nodes by constructing topologies such that the connectivity of the nodes increases with their *HC*. Hence, a powerful node would maintain more connections and would receive more queries on average when compared to a weak node. In short, our heterogeneity-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 Multitier 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 their topology construction and maintenance costs. Second, we present an analytical model for designing such heterogeneity-aware multitier topologies. Finally, we provide simple and pragmatic techniques to realize such topologies.

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

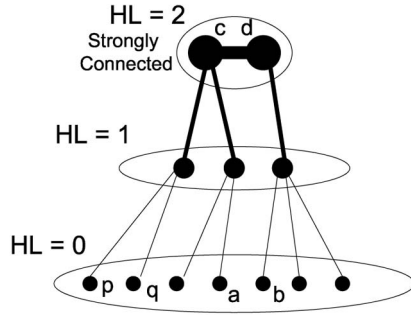


Fig. 2. Hierarchical topology.

Definition 2.1. Multitier Topologies: Let $HC(p)$ denote the heterogeneity class of peer p . Let $maxHC$ denote the highest heterogeneity class. A multitier 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 Connectivity Preserving Rule (CPR) and Connection Restriction Rule (CRR) are satisfied:

CPR: $\forall p \in V$, if $HC(p) < maxHC$ then $\exists q \in V$ such that $((HC(q) = HC(p) + 1) \wedge ((p, q) \in E))$.

CRR: $\forall (p, q) \in E, |HC(p) - HC(q)| \leq 1$.

Hierarchical Topology. Hierarchical topology is a pure tree topology, except that the nodes in the highest HC are strongly connected. An example hierarchical topology is shown in Fig. 2. Below, we provide a formal definition of a hierarchical topology.

Definition 2.2: Hierarchical topology. Let $CL(x, y)$ denote the communication latency between two peers at heterogeneity classes x and y , respectively, and weight

$$w(p, q) = CL(HC(p), HC(q)).$$

A hierarchical topology is defined as a multitier topology $G : \langle V, E \rangle$, such that the following additional condition is verified:

If edge $(p, q) \in E$, where $HC(p) \leq HC(q)$, then it has weight $w(p, q)$ and

$$(HC(p) = HC(q) = maxHC)$$

$$\forall \neg(\exists qt \text{ s.t. } q)((qt \neq \wedge (HC(p) \leq HC(qt)) \wedge ((p, qt) \in E))).$$

This condition states that in a hierarchical topology, two peers at the highest HC may be connected; if a peer does not belong to the highest HC , then the peer can only be connected to exactly one node in its immediate higher HC .

Sparse Topology. The second class of overlay topology is Sparse Topology. Sparse topology permits a node to maintain connections with more than one node at its immediate higher class, as illustrated in Fig. 3. To be concise, we leave out the formal definition of the sparse topology. Informally, a sparse topology is defined as a multitier topology (CRR and CPR are satisfied) where connections between two nodes in the same HC are not allowed, except for the nodes at the $maxHC$ level.

Dense Topology. Dense topology is the third class of multitier overlay topology considered in this paper. An example dense topology is shown in Fig. 4. Informally, a dense topology is a generic multitier topology where no

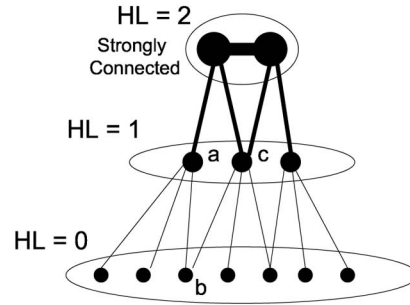


Fig. 3. Sparse topology.

further restrictions are put on the connections between the nodes, other than CRR and CPR that are part of the multitier topology definition.

2.3 Performance Characterization of Three Multitier 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 *Heterogeneity-Aware Vandwidth (HAB)* consumed by a query to the number of results obtained. *Heterogeneity-Aware Bandwidth (HAB)* is defined as the ratio of the total number of messages sent/received by a node to the node's capability. 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 HAB by normalizing the total number of messages sent by a node with its capability. Strictly speaking, we should measure bandwidth as the number of messages per unit of time. However, we just measure the number of messages (normalized by capability) 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 mean (averaged over queries) amount of HAB 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. Note that amortized messaging bandwidth is a per-query HAB expenditure, while load is a per-node HAB expenditure.

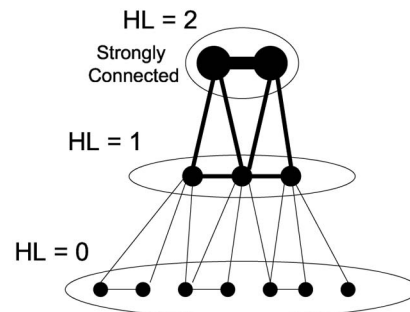


Fig. 4. Dense topology.

Coverage. Defined as the number of nodes that receive a broadcast query whose initial TTL was set to $initTTL$, coverage is an 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 in classifying nodes into classes and restricting their connectivity should not weaken the connectivity between different classes or, in the worst case, disconnect the overlay topology.

Amortized Latency. Defined as the ratio of total response time for a query to the number of results obtained, amortized latency captures the response time for a search query as perceived by an end-user. For example, let us consider the following two scenarios: In the first scenario, we get 10 answers in five seconds and in the second scenario we get one answer in one second. Note that answers are received over a period of time. This implies that, on average, two answers are received per second in the former scenario, which is an improvement over the latter scenario, where we get one answer in one second.

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 subset of nodes in the system. Further, our measurement of fault-tolerance indicates the ability of an overlay topology to defend itself against certain kinds of Denial-of-Service (DoS) attacks by the 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 give an in-depth discussion of their performance trends with respect to various performance metrics listed in Section 2.31.

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 HC). Hence, this topology consumes minimum aggregate bandwidth when a broadcast-style algorithm is used. However, the increase in the number of duplicate queries in sparse and dense topologies (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 heterogeneity-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, significantly reduces the heterogeneity-aware bandwidth (HAB) requirement of our overlay topologies.

Load Variance. In a hierarchical topology, due to its *tree-like* structure, the nodes at the highest HC are likely to be more heavily loaded (relative to their capability) than the nodes at lower classes, thereby making the load distribution somewhat skewed. Sparse (or 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 multitier topologies, by their very structure, ensure that the load on a node is commensurate with its capability and, hence, show remarkably better load

TABLE 1
Summary: Overlay Topologies

| Topology Type | Amortized Bandwidth | Load Variance | Coverage Coverage | Amortized Latency | Fault Tolerance |
|---------------|---------------------|---------------|-------------------|-------------------|-----------------|
| Random | very large | very poor | medium | very high | good |
| Hierarchical | low | poor | low | medium | poor |
| Sparse | low | good | high | low | good |
| Dense | low | good | high | low | good |

distribution, as compared to a random topology that is agnostic to peer heterogeneity. More importantly, the uniform load distribution property of our heterogeneity-aware overlay topologies guarantees that the weak nodes are not overloaded.

Coverage. A hierarchical topology's tree-like structure increases the average distance between nodes on 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 average distance between nodes on 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, as compared to a random topology.

Amortized Latency. Latency (response time as perceived by an end-user) is much lower in the multitier topologies primarily because they largely prevent 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 in a multitier 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 *subtree* 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 *more responsibility* to higher class nodes, thereby making them marginally weaker than in 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 [15] that nodes with powerful network connections stay longer in the system, contrasting with weak network connections (like dial-up users). Hence, the probability of failure of a higher-class node (because of it leaving the system) is much lower than that of a lower-class node. Under this assumption of *nonuniform faults*, the multitier 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 1 shows a summary of our discussion in this section: 1) A hierarchical topology is best suited for minimizing the amortized messaging bandwidth consumption for any query Q since it largely eliminates duplicate queries, 2) the dense topology is the best choice for minimizing load variance since it does not impose heavy constraints on a node's neighbors, and 3) 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 the 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 HATs: Analytical Model

Having presented three classes of multitier heterogeneity-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 minimizes the variance of HAB over all nodes in the system. We characterize an overlay topology by its classes and the number of neighbors maintained by the nodes in each class. We present our model assuming a broadcast-style routing algorithm is used. However, the model can be easily modified to accommodate various other routing techniques, including the iterative deepening technique [20] and the random walk technique [2]. It is important to note that our heterogeneity-aware overlay topologies reduce the per-query HAB expenditure for most routing algorithms known in the literature.

Let N be the total number of nodes in the system, $numHC$ be the number of HCs, C_i be the capability of a node at class i ($0 \leq i \leq numHC - 1$), f_i be the fraction of nodes at class i , and d_i be the average degree of nodes at class 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 average number of edges from a node at class i to nodes at class $i + 1$, d_i^{down} denotes the average number of edges from a node at class i to nodes at class $i - 1$, and d_i^{in} denotes the average number of edges from a node at class i to other nodes at the same HC class i . Note that $d_{numHC-1}^{up} = 0$ 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 class i to class $i + 1$ should equal the total number of edges from nodes at class $i + 1$ to class i . Hence, $\forall i$, $0 \leq i \leq numHC - 2$, the following equation holds:

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

We derive the HAB expended by every node p as a function of its heterogeneity class, the initial value of TTL, and the degree parameters d_i^X ($0 \leq i \leq numHC - 1$ and $X \in \{up, down, in\}$). Given the HAB expended by every node p in the system and its capability, one can compute the variance of HAB, namely, $Var(HAB(p))$ over all nodes p . We resort to optimization techniques like simulated annealing [8] to obtain a near-optimal solution for the degree parameters d_i^X that minimize the variance of HAB.

We estimate the HAB 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 $d_{HC(p)}$; 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 HC that would receive a query Q sent by some node. We use this information to deduce the HAB expended by every node in the system. Let $coverage_{lev}(hc, ttl)$ denote the number of *new* nodes at class hc that would be reached by a query Q with $Q.TTL = ttl$ when

some node r at class 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,

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

Let $request_{lev}(hc, ttl)$ denote the total number of query messages with $TTL = ttl$ that is received by some node at heterogeneity class hc ($0 \leq hc < numHC$). We can compute $request_{lev}(hc, ttl)$ using *coverage* as follows:

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

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

Among $request_{lev}(hc, ttl)$ queries, some of them reach nodes that have already received the query, while the rest reach new nodes. The number of nodes that have *not* received the query prior to $Q.TTL = ttl$, denoted by $notrcvd_{lev}(hc, ttl)$, can be computed as follows:

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

Hence, the probability that a query Q with

$$Q.TTL = ttl$$

reaches a new node at class hc is

$$prnew_{lev}(hc, ttl) = \frac{notrcvd_{lev}(hc, ttl)}{N * f_{hc}}.$$

Also, the number of queries Q that are forwarded to new nodes at class hc when $Q.TTL = ttl$ can be computed by $reqnew_{lev}(hc, ttl) = prnew_{lev}(hc, ttl) * request_{lev}(hc, ttl)$.

Now, $reqnew_{lev}(hc, ttl)$ queries are directed toward

$$notrcvd_{lev}(hc, ttl)$$

nodes at class hc when $Q.TTL = ttl$. We have to estimate the number of distinct nodes (among $notrcvd_{lev}(hc, ttl)$ nodes) that indeed receive the query request. One can show that the number of distinct nodes that receive the query request at class hc with $Q.TTL = ttl$ is given by (5):

$$\begin{aligned} coverage_{lev}(hc, ttl) &= \\ notrcvd_{lev}(hc, ttl) &* \left(1 - \left(1 - \frac{1}{notrcvd_{lev}(hc, ttl)} \right)^{reqnew_{lev}(hc, ttl)} \right). \end{aligned} \quad (5)$$

Note that the parenthesized part of (5) is the probability that a new node at class hc will receive at least one query request with $Q.TTL = ttl$. This completes the recursive definition of the *coverage* function. Equations (3) to (5) recursively define the *coverage* function with the base case in (2). Finally, the HAB expended by some node p is computed as the total number of messages to be sent/received by nodes at class $HC(p)$ divided over the number of nodes in class $HC(p)$ and normalized by capability

$C_{HC(p)}$. Before that, we categorize the queries into those received with $Q.TTL > 0$ and $Q.TTL = 0$. We use λ to denote the rate of query generation and τ to denote the time duration of interest.

$$\begin{aligned} nummsg_{Q.TTL>0}(HC(p)) &= d_{HC(p)} * \tau \\ &* \sum_{lev=0}^{numHC-1} \left(N * f_{lev} * \lambda * \sum_{ttl=1}^{initTTL} coverage_{lev}(HC(p), ttl) \right) \end{aligned} \quad (6)$$

and

$$\begin{aligned} nummsg_{Q.TTL=0}(HC(p)) &= \\ \tau * \sum_{lev=0}^{numHC-1} (N * f_{lev} * \lambda * coverage_{lev}(HC(p), 0)). \end{aligned} \quad (7)$$

Hence, one can obtain $HAB(p)$ from (8) as

$$\begin{aligned} HAB(p) &= \\ \frac{nummsg_{Q.TTL>0}(HC(p)) + nummsg_{Q.TTL=0}(HC(p))}{N * f_{HC(p)} * C_{HC(p)}}. \end{aligned} \quad (8)$$

Note that the set of recursive equations listed above can be easily converted into an iterative algorithm of complexity $O(numHC \times initTTL)$ that accepts d_i^{up} and d_i^{in} for all $0 \leq i \leq numHC - 1$ (recall that (1) constrains d_i^{down}) as inputs and outputs the variance of load distribution. One can resort to optimization techniques, such as simulated annealing [8], to obtain near optimal solutions (for d_i^X) that minimize the load variance. Further, it is straightforward to incorporate scenarios wherein the rate at which a node n issues queries depends on its heterogeneity class $HC(n)$; say, a node n issues queries at rate $\lambda_{HC(n)}$.

2.5 Constructing and Maintaining Heterogeneity-Aware Multitier Topologies

Multitier topology construction consists of two main components: bootstrapping a new node and maintaining the topology in the presence of node departures and node failures. We first describe the algorithm for bootstrapping a new node using the basic bootstrap service common to most existing P2P systems. We then discuss an enhanced bootstrap service that not only provides support for incorporating dynamic changes in the number of tiers (HC s), but also facilitates certification of the heterogeneity classes (HC s) declared by new peers upon their entry to the multitier topology-based overlay network.

Topology Construction. The structure of a multitier overlay topology $G : \langle V, E \rangle$ with $numHC$ tiers is defined by d_i^{up} , d_i^{down} , and d_i^{in} for $0 \leq i \leq numHC - 1$. Also, $d_{numHC-1}^{up} = 0$ and $d_0^{down} = 0$ for all three classes of multitier topologies. Let $neighbors(p)$ denote the set of neighbor peers of node p . The following formula holds: $neighbors(p) = \{q | (p, q) \in E\}$ and $|neighbors(p)|$ is, on average, equal to

$$d_{HC(p)}^{up} + d_{HC(p)}^{down} + d_{HC(p)}^{in}.$$

Pseudocode of the algorithm for bootstrapping a new node p in a multitier topology is shown in Fig. 5. We assume that the bootstrap servers maintain a cache of recently joined nodes. Note that for a hierarchical topology, $d_i^{up} = 1$ and $d_i^{in} = 0$ for

```

MULTITIERBOOTSTRAP(Node p)
(1) Let in = 0, up = 0
(2) repeat
(3)   {Obtain entry point peers from the bootstrap server}
(4)   E ← BootstrapServer(p)
(5)   for i = 0 to |E| - 1
(6)     if up < d_{HC(p)}^{up} and HC(p) + 1 = HC(E_i)
(7)       {Add node p as a neighbor of node E_i and vice versa}
(8)       Neighbors(p) ← Neighbors(p) ∪ {E_i}
(9)       Neighbors(E_i) ← Neighbors(E_i) ∪ {p}
(10)      up ← up + 1
(11)     if in < d_{HC(p)}^{in} and HC(p) = HC(E_i)
(12)       Neighbors(p) ← Neighbors(p) ∪ {E_i}
(13)       Neighbors(E_i) ← Neighbors(E_i) ∪ {p}
(14)      in ← in + 1
(15) until in ≥ d_{HC(p)}^{in} and up ≥ d_{HC(p)}^{up}

```

Fig. 5. Bootstrapping a new node p in a multitier topology.

$0 \leq i \leq numHC - 2$; for a sparse topology, $d_i^{in} = 0$ for $0 \leq i \leq numHC - 2$; and for a dense topology, there are no such constraints. Further, note that d_i^X ($X \in \{in, up, down\}$) need not be integers. We tackle this problem as follows: Denoting the fraction part of Y as $[Y]$, with a probability of $[d_i^X]$, the node maintains $d_i^X - [d_i^X] + 1$ connections of type X and with a probability $1 - [d_i^X]$, the node maintains $d_i^X - [d_i^X]$ connections of type X .

We build the topology incrementally. A new node p proactively opens connections to an existing node q that is at the same or at a higher class than node p ($HC(q) \geq HC(p)$). However, the connections from a node p to a node q at a lower heterogeneity class than node p ($HC(q) < HC(p)$) are initiated by node q rather than node p . The key motivation for this rule stems from the common observation that a node with larger bandwidth is likely to stay connected to the system for a longer period of time. Since nodes at higher classes have larger bandwidth, the probability that a node would lose connection to a node at higher class (because of the higher class node leaving the system) is quite low. This reduces the topology maintenance costs discussed below. Assume that a node n 's lifetime is exponentially distributed with mean $\frac{1}{\mu_{HC(n)}}$. One can show that probability that a node n loses a connection with a node m because of node m leaving the system is

$$\frac{\mu_{HC(m)}}{\mu_{HC(n)} + \mu_{HC(m)}}.$$

Hence, probability that a node n loses a connection with a node at a lower class than node n is greater than $\frac{1}{2}$ and that with a node at a higher class than node n is much smaller.

Topology Maintenance. Once a node has joined the overlay network, it may lose connections with some or all of its neighbors due to various reasons, including faults in the underlying network or the 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 neighbor peers can provide a set of peers to which peer p can potentially connect. If peer p indeed loses all of its neighbors, then it can do an IP-level expanding-ring search to locate other nodes in the overlay network or contact one of the bootstrap servers to obtain a new entry point peer. Another important feature of the multitier topology is that in our maintenance scheme, a

TABLE 2
Bootstrap Service

| Topology Type | $\{E\} \leftarrow bootstrap(p)$ | $\{E\} \leftarrow bootstrap(p)$ |
|----------------|--|---------------------------------|
| | $HC(p) < maxHC$ | $HC(p) = maxHC$ |
| Hierarchical | $HC(E) = HC(p) + 1$ | $HC(E) = maxHC$ |
| Layered Sparse | $HC(E) = HC(p) + 1$ | $HC(E) = maxHC$ |
| Layered Dense | $HC(E) = HC(p) \vee HC(E) = HC(p) + 1$ | $HC(E) = maxHC$ |

node only needs to maintain the required number of connections with other nodes at the same or higher HC . A node does not actively respond when it loses a connection with a lower class node. The nodes at the lower HC tiers are the ones responsible for maintaining their connections with the higher HC nodes. Observe that maintaining a sparse or a dense topology is much easier than a hierarchical topology. Since nodes in sparse or dense topologies maintain multiple connections to other nodes at higher classes (and at same class), a node gets completely disconnected only if it loses all its connections.

Enhancing Bootstrap Services. We have discussed how to construct heterogeneity-aware multitier topologies using the basic bootstrap services commonly used in many existing decentralized P2P systems. One enhancement to the basic bootstrap service is to let the bootstrap server maintain a cache of live nodes for every HC . If a new node p with $HC(p) = maxHC$ contacts the bootstrap server, the server replies with a set of nodes whose heterogeneity class is $maxHC$ irrespective of the concrete topology structure, be it hierarchical, sparse, or dense. If the new node $HC(p) < maxHC$, then the bootstrap server will return a list of entry nodes $\{E\}$ that are in the immediate higher HC of the node p , i.e., $HC(E) = HC(p) + 1$. For dense topology structure, the bootstrap server can also return those entry nodes that are at the same HC tier as the new node p . We summarize the responses of the enhanced bootstrap server upon entry of a node p in Table 2. The second and third columns in the table enforce constraints on the HC of the entry point peers returned by the bootstrap server. This enhanced bootstrapping approach has at least two advantages: First, when the number of heterogeneity classes ($numHC$) of the participating peers changes (due to technological advancements), only the bootstrap servers are required to be aware of such changes in order to construct topologies in accordance with the new heterogeneity classes. Second, and more importantly, the bootstrap servers can guard the system from malicious peers. For example, the bootstrap servers can test a peer's capability upon its entry to the system and provide a *certificate* to the new peer indicating its HC . With this enhanced scheme, a connection is established between two nodes p and q only if they are mutually convinced about each other's capability.

Observe that, at one extreme, 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; hence, it does not suffer from weak connectivity. Another special case of our heterogeneity-aware topologies is the supernode architecture, which is a two-tiered hierarchical topology. From our analysis and experimentation, we observed that an enhanced supernode topology (a two-tiered sparse topology) performs significantly better with respect to several performance metrics.

3 HAR: HETEROGENEITY-AWARE ROUTING

Recall the broadcast-styled routing (*bsr*) technique used in most Gnutella-like P2P systems today: A 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) [20]. Each query is associated with another parameter $Q.threshold$, specifying the number of results required. The query originator iterates over the query's $initTTL$, going 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 percent) are free-riders [1] and that about 90 percent of documents are shared by about 10 percent of the nodes [15]. 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 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 heterogeneity-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 three major components: *ranking neighbor peers*, *processing a query using the ranking information*, and *maintaining the rankings up-to-date*. We discuss each of them in detail below.

3.1 Ranking Neighbor Peers

One of the critical components of the probabilistic broadening search technique is to forward a query to only the best subset of neighbor peers instead of all the neighbor peers. There are several mechanisms to rank the set of neighbor peers and to select a subset of the best neighbors. A simple approach is to use the past performance of peers in serving queries as a measure to rank them. In a P2P system, peers are constantly interacting with their neighbors as a part of their query and response forwarding responsibility. The results of a query retrace the path to the query originator; a peer p would receive results of a query Q from its neighbors ($Neighbors(p)$) and, in turn, peer p forwards these results toward the query originator of Q . Hence, a peer can collect information about its neighbors in terms of the number of results obtained from them in the past. Note that the results obtained through a peer p not only include the matching files offered at peer p , but also include the results obtained from all those peers to whom the peer p forwarded the query.

Below, we provide a number of metrics to rank the *effectiveness* of a peer:

- **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. In other words, one can expect such a peer to be an active and powerful participant in the peer community.
- **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 parameters to obtain a more sophisticated ranking metric. The ranking G of a peer p can be obtained as

$$G_p = w_1 * Degree(p) + w_2 * Results(p) + w_3 * Rate(p),$$

where $w_1 + w_2 + w_3 = 1$.

It is important to note that the dynamic nature of peers joining and departing the network makes the above metrics change continuously. This can happen as peers join, leave, or update their shared file index. We provide a robust technique to dynamically update the neighbor rankings over time (see Section 3.3 for details).

Concretely, our ranking algorithm proceeds as follows: When a query is forwarded from peer q to peer p , let the set of neighbors of peer p be $X = \{x_1, x_2, \dots, x_n\}$, to which peer p can potentially forward the query, and $x_i \neq q$ for $1 \leq i \leq n$. We can compute the ranking value for each neighbor peer x_i as $RV_i = G_i / \sum_{j=1}^n G_j$, where G_i is the goodness metric of x_i obtained by measuring the chosen metric, be it a simple metric or a weighted formula of a set of simple metrics. Note that peer q is excluded from the set X ($q \notin X$). Also, RV_i ($1 \leq i \leq n$) are normalized such that $\sum_{j=1}^n RV_j = 1$ holds. $RV_j > RV_k$ ($1 \leq j, k \leq n$) implies that neighbor x_j is a better choice for obtaining results for a query than neighbor x_k . A pseudocode of the algorithm is given in Fig. 6.

It is vital to note that the ranking metrics discussed above subsume that peers are noncooperating nodes. Studies on live Gnutella networks [15] reveal that the P2P network

```

RANKPEERS(Peers X)
(1) Rank: Final ranking order of peers in the neighbor set X
(2) Let  $X = \{x_1, x_2, \dots, x_n\}$ 
(3)  $G_i$ : the ranking measure of peer  $x_i$ 
(4)  $w_1, w_2, w_3$ : Weights for metrics on Degree, Results, and Rate
(5)  $RV_i$ : Normalized Ranking Measure of peer  $x_i$ 
(6)  $G \leftarrow \{G_i \mid 1 \leq i \leq n, G_i = w_1 * Degree(x_i) + w_2 * Result(x_i) + w_3 * Rate(x_i)\}$ 
(7) { Compute normalized ranking of peers in X }
(8)  $S \leftarrow \sum_{j=1}^n G_j$ 
(9) for  $i = 1$  to  $n$ 
(10)  $RV_i = G_i/S$ 
(11) RankList  $\leftarrow$  Sort X in decreasing order of  $RV_i$ 
(12) return RankList

```

Fig. 6. Ranking a set of peers X .

contains a small percentage of peers that act *like servers*, while most of the other peers act predominantly *like clients*. If all peers are cooperating, different set of ranking schemes should be used.

From our experiments, we observed that in a random topology, all the metrics suggested above have very similar performance. However, for any of the layered topologies, the connectivity factor (Max Degree) is embedded in their structure; hence, Max Results and Max Rate show improved performance over Max Degree. Without loss of generality, in the rest of the paper we use Max Results as the metric for all discussions.

3.2 Selective Routing

The query processing using the selective routing search technique consists of three steps: 1) the job done by the query originator to control the iteration and the breadth parameter, 2) the job done by the peer p that receives a query Q from the query originator or another peer, which involves computing the ranking metric for each candidate neighbor peer of p , and 3) the job of p upon receiving the results from its B neighbors for the query Q . Fig. 8 provides the pseudocode sketching these query processing steps in detail.

Concretely, the query originator initializes the breadth parameter ($Q.B$) of a query Q to $initB$. Every peer that receives the query ranks its neighbors and forwards Q to its best $Q.B$ neighbors. The query originator waits for the results; if the number of results obtained is not satisfactory, it reissues the query Q with its breadth parameter $Q.B$ incremented by 1. We use $maxB$ to denote the maximum value of the breadth parameter. Fig. 7 illustrates our algorithm at peer p for $Q.B = 1$ and 2 with $RV_1 > RV_2 > RV_3 > RV_4$ and $RV_5 > RV_6$.

Pseudocode of our algorithm is provided in Fig. 8. The procedure $PROCESSQUERY(Q, p, q)$ is executed when a query Q is received by peer p from peer q . Note that peer p forwards query Q to its best $Q.B$ neighbors as against all of its neighbors (as in a pure *bsr* scheme). The procedure $PROCESSRESPONSE(R, Q, p, x_i)$ is executed when response R for a query Q is returned from neighbor peer x_i to the peer p . Peer p first updates the ranking metric G_i based on the results returned by peer x_i and, then, it forwards response R toward the query originator of Q .

3.3 Maintaining Rankings Up-to-Date

As nodes join and leave the overlay network, a peer p has to update the rankings of its neighbors. Say, at time instant t , a peer p ranks its neighbors $X = \{x_1, x_2, \dots, x_n\}$

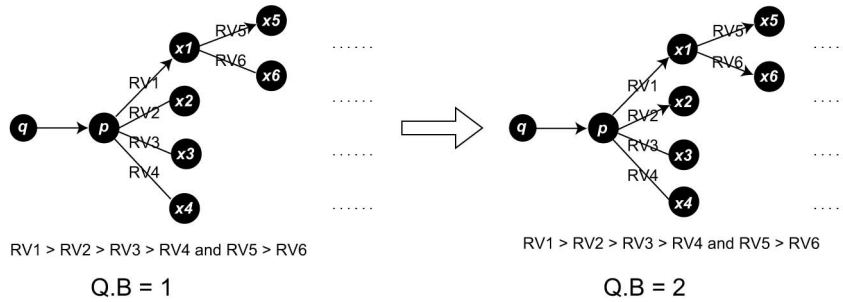


Fig. 7. Selective routing algorithm: illustration.

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 peer p gets results only from the top-ranked b neighbors: x_1, x_2, \dots, x_b . Thus, the goodness metric G_i will be updated for only these b peers; however, the goodness measures of peers $x_{b+1}, x_{b+2}, \dots, x_n$ are not updated at this time since peer p did not forward queries to peers $x_{b+1}, x_{b+2}, \dots, x_n$. Hence, changes to the file indices or the overlay topology itself (due to peer joins/leaves) of peers accessible through neighbors $x_{b+1}, x_{b+2}, \dots, x_n$ are not considered in the subsequent ranking computations. In other words, a peer, once having ranked its neighbors, gets stuck with the same ranking order simply because it does not explore the rest of its neighbors to see if they can return better results *now*.

In order to capture the dynamics of the entire P2P network, we modify the *neighbor selection step* of our algorithm in Fig. 8 as follows: Instead of selecting *deterministically* the best $Q.B$ peers, peer p select $Q.B$ peers *probabilistically*, that is, each of the neighbor peers x_j is

```
QUERYORIGINATOR(Query Q, Threshold thr)
```

- (1) $Q.B \leftarrow \text{init}B$
- (2) $R \leftarrow \emptyset$
- (3) **repeat**
- (4) PROCESSQUERY($Q, Q.\text{originator}, \text{nil}$)
- (5) $R \leftarrow R \cup \text{Response for query } Q$
- (6) $Q.B \leftarrow Q.B + 1$
- (7) **until** $|R.\text{results}| \geq Q.\text{threshold} \vee Q.B > \text{max}B$
- (8) **return** $R.\text{results}$

```
PROCESSQUERY(Query Q, Peer p, Peer q)
```

- (1) $X = \{x_1, x_2, \dots, x_n\} \leftarrow \text{Neighbors}(p) - \{q\}$
- (2) LR : Local Results - Matching files in peer p for query Q
- (3) **if** $Q.TTL = 0$
- (4) Drop the query Q
- (5) **else**
- (6) $Q.TTL \leftarrow Q.TTL - 1$
- (7) {Neighbor Selection Step}
- (8) **if** $Q.B \leq n$
- (9) $RX \leftarrow \text{RANKPEERS}(X)$ {Rank peers in X }
- (10) Forward Q to peers $RX_1, RX_2, \dots, RX_{Q.B}$
- (11) **else**
- (12) Forward Q to peers x_1, x_2, \dots, x_n
- (13) $LR \leftarrow p.\text{matches}(Q.\text{keywords})$
- (14) Send LR to peer q

```
PROCESSRESPONSE(Response R, Query Q, Peer p, Peer x_i)
```

- (1) Update G_i {Update Goodness Measure of peer x_i }
- (2) Forward the response R towards $Q.\text{originator}$

Fig. 8. Query processing.

selected with a probability that is equal to its normalized rank value RV_j . Hence, most of the queries get routed through the neighbors who have performed well before. At the same time, by probabilistically sending the query to some inferior neighbors, peer p can figure out if they can provide us better results now. Pseudocode of the modified neighbor selection algorithm is shown in Fig. 9.

Summary. In several respects, one could view the probabilistic-selective routing algorithm (*psr*) as a generalization of the biased random-walk [10] and the basic broadcast-style algorithm. For example, with $\text{min}B = \text{max}B = 1$, the *psr* algorithm reduces to a biased random-walk; with $\text{min}B = \text{max}B = \infty$, the *psr* algorithm reduces to a broadcast-style 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 less network bandwidth than a broadcast-style algorithm; for unpopular (rare) documents, the *psr* algorithm is almost equivalent to a broadcast-style algorithm and, hence, incurs significantly lower latency than the biased random-walk algorithm.

3.4 Probabilistic Selective Routing with Result Caching

It is observed by many authors [16], [17] 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. This leads one to believe that *caching query results* would be very effective for a Gnutella-like overlay network.

Result caching in a Gnutella network simply refers to *caching query results*. In a flooding-based querying system, a peer sees many queries and its responses as a part of its

```
SELECTNEIGHBORS(Query Q, Peers X)
```

- (1) SX : Probabilistically selected peers from X
- (2) Let $X = \{x_1, x_2, \dots, x_n\}$
- (3) $RV \leftarrow \text{RANKPEERS}(X)$
- (4) $SX \leftarrow \emptyset$
- (5) **if** $Q.B \geq n$
- (6) $SX \leftarrow X$
- (7) **else**
- (8) **for** $i = 1$ to $Q.B$
- (9) $x \leftarrow \text{Select from } X \text{ a peer } x_j \text{ with probability } RV_j$
- (10) $SX \leftarrow SX \cup \{x\}$
- (11) $X \leftarrow X - \{x\}$
- (12) $RV \leftarrow \text{RANKPEERS}(X)$
- (13) { SX contains the selected neighbors from X }
- (14) **return** SX

Fig. 9. Maintaining ranking up-to-date.

TABLE 3
Network Generator

| Parameter | Description | Default |
|-----------|---|-----------|
| N | Number of Nodes | 10,000 |
| HC | 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 |

routing responsibility. This can be used to build caches that are aimed at providing faster responses and reducing the aggregate bandwidth consumption. Although extensive research has devoted to caching techniques such as Web caching, the highly dynamic nature of the network warrants that the caching technique be capable of adapting itself to the dynamics of the system. Hence, handling the stale results proves to be a major hurdle in designing a caching system for a highly decentralized and dynamic P2P system like Gnutella. In this section, we show that one can handle this problem effectively using the *psr* technique.

The key motivation for using *psr* with result caching stems from the fact that both of these techniques share a common design philosophy. A general result caching scheme [11] can be summarized briefly as follows: When a peer p processes a query Q forwarded by peer q , it makes a 5-tuple cache entry: $\langle Q.keywords, q, Q.TTL, R, t \rangle$, where q is the neighbor peer from which the query Q was forwarded, R denotes the query results, and t denotes the time stamp for measuring staleness. This scheme can be easily integrated with *psr* as follows: In fact, result caching can be viewed as a fine-grained ranking system. For a given query Q , a peer p can estimate the number of results that could be obtained through each of its neighbors using its result cache. Concretely, the ranking metric *Max Results* used in the selective routing scheme can be computed as *aggregates* of the cached results. Hence, the techniques used to maintain peer rankings up-to-date in the *psr* scheme can be effectively adapted to solve the stale-results problem in 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 simplicity of a *psr*-based solution to the stale results problem motivates us to use result caching and benefit from improved performance in the presence of bursty traffic.

TABLE 4
Document Generator

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

TABLE 5
Search Simulator

| Parameter | Description | Default |
|-----------|------------------|---------|
| THR | Result threshold | 32 |

4 EXPERIMENTAL RESULTS

In this section, we present three sets of simulation-based experiments. The first set compares the three classes of multitier topologies against a random topology using our performance metrics. The second set evaluates our model for constructing multitier heterogeneity-aware topologies against a random topology. The third set studies the effectiveness of the probabilistic selective routing scheme in handling vastly heterogeneous nodes.

4.1 Simulation Setup

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 using a randomly chosen node in the system. The latency of a link from node n to node m is modeled as being inversely proportional to $\min(C_{HC(n)}, C_{HC(m)})$. 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 3, 4, and 5 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 [15], [2]. All the results presented in this paper have been averaged over five simulation runs.

Network Generator. We use our multitier bootstrap algorithm and construct the overlay network incrementally by adding one node at a time. For our simulation, we use 20 percent, 70 percent and 10 percent of the peers for the corresponding $HC = 0, 1$ 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 nonuniform distribution of documents among peers (Db of 20 percent-80 percent means that 20 percent of the peers hold about 80 percent of the documents).

Routing Simulator. The routing simulator implements one of the following routing algorithms: broadcast-style routing, iterative deepening, biased random-walk, and probabilistic-selective routing.

4.2 Evaluation of HATs

We compare the performance of hierarchical, sparse, dense, and random overlay topologies 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 classes (peer classes) in achieving improved performance.

Amortized Messaging Bandwidth. Fig. 10 presents the amortized messaging bandwidth consumption on various topologies. For $N = 10,000$ nodes, the hierarchical topology consumes the least amortized messaging bandwidth since

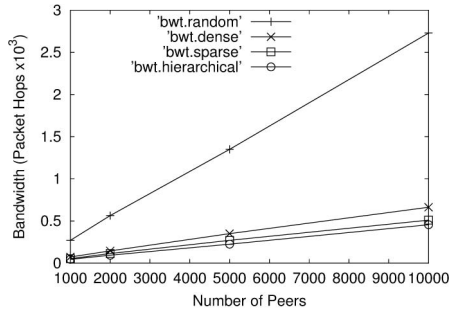


Fig. 10. Amortized messaging bandwidth.

its tree-like structure minimizes the number of duplicate queries. Also, the multitiered sparse and dense topologies reduces the HAB expenditure by about **six to ten times** as compared to a random topology because they require that the weak nodes send/receive far fewer messages when compared to strong (more capable) nodes.

Load Distribution. Fig. 11 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 load variance, while a hierarchical topology shows 10 times lower load variance. The hierarchical topology pays for the strict constraints it imposes on the overlay topology ($d_i^{up} = 1$ and $d_i^{in} = 0$). Uniform load distribution properties of heterogeneity-aware topologies ensure that the weak nodes do not become bottlenecks and throttle the performance of the system.

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 class of peers. Fig. 12 shows the average coverage of a peer in the

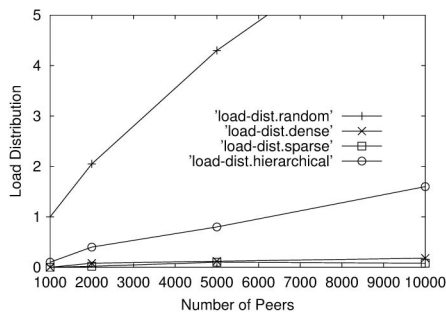


Fig. 11. Variance of load/capability.

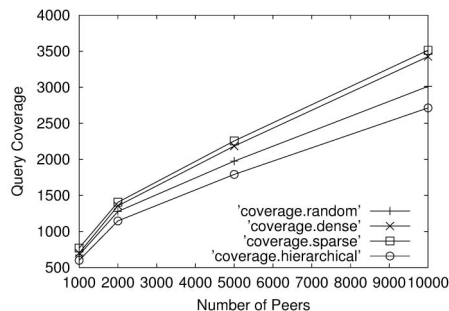


Fig. 12. Query coverage.

TABLE 6
Coverage versus HC for $N = 10,000$

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

network, with scope $initTTL = 7$. For $N = 10,000$ nodes, the dense and sparse topologies show 20 percent more coverage because their structured nature largely reduces the number of duplicate queries; the hierarchical topology shows 15 percent lower coverage because of its increased mean node-to-node distance. Table 6 shows the variation of coverage with different HC s of peers that are normalized with respect to $HC = 0$ peers. One key conclusion drawn from this table is that the extra work done by the higher HC 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. Fig. 13 shows the amortized latency for 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 $\min(C_{HC(n)}, C_{HC(m)})$. 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 heterogeneity-aware topologies. Further, among the heterogeneity-aware topologies, the hierarchical topology shows about 5 percent and 12 percent 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.

Fault-Tolerance. We have studied the fault-tolerance of the four topologies under two conditions: *Uniform Faults* and *Nonuniform Faults*. Fig. 14 shows the quality of the results obtained when a random 10 percent 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 percent lower fault-tolerance than the random topology because of its delicate tree-like structure; the sparse and the dense topologies exhibit only 2-3 percent lower fault-tolerance because they assign more responsibility to higher-class nodes; recall that in a random topology, all nodes are treated alike and, hence, are assigned equal responsibilities.

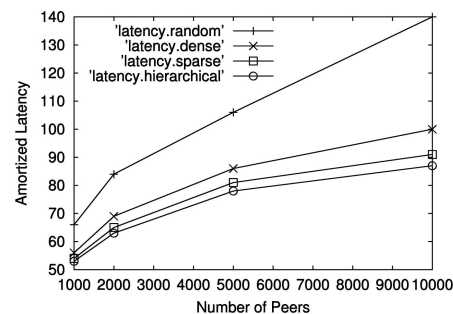


Fig. 13. Amortized latency.

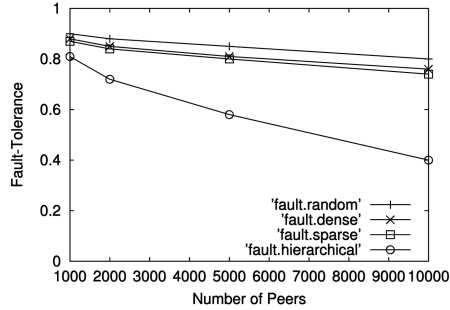


Fig. 14. Fault tolerance with uniform faults.

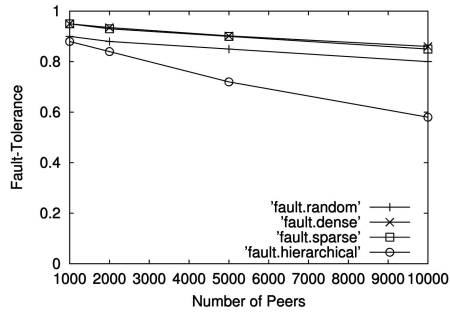


Fig. 15. Fault tolerance with nonuniform faults.

Fig. 15 shows the fault-tolerance of the topologies under nonuniform faults with 10 percent of failed peers, where the probability of peer failure at $HC = 0, 1$, and 2 are in the ratio 4:3:1 [2], [15]. For $N = 10,000$ nodes, the sparse and dense topology show about 4 percent-5 percent higher fault-tolerance than the random topology and the hierarchical topology shows 20 percent improvement over the uniform faults case. This improvement is primarily because the multitiered topologies assign *more responsibility* to more capable nodes that are *less likely to fail* (higher-class nodes).

Importance of Choosing Correct $numHC$. To demonstrate the importance of determining the correct value of $numHC$, we compare the performance of a system with an incorrect value for $numHC$ 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 $numHC = 2$ (by merging the top two peer classes) and $numHC = 3$. Table 7 shows ratios of the performance measures obtained for $numHC = 2$ to those obtained for $numHC = 3$. For instance, a sparse topology with $numHC = 2$ shows 1 percent less coverage, **3.5 times** more

TABLE 7
Importance of Choosing Correct $numHC$

| 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 8
Bootstrapping Error

| 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) |
| # classes $\times 1$ | 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) |

bandwidth (HAB), 16 percent more latency, 18 less fault-tolerance, and **11 times** more variance in load distribution than a sparse topology with $numHC = 3$.

Importance of Choosing Sizable Peer Classes. In this experiment, we illustrate that having peer classes with very small number of peers may not be quite beneficial for the performance of the system. Let us assume that there are 10 genuine peer classes (1 to 10), with peer class 1 being the least powerful set of peers. But, the number of peers in class i ($1 \leq i \leq 10$) follows a Zipf-like distribution with parameter $\alpha = 1$. Note that, by the properties of the Zipf distribution, the number of class 1 peers would be roughly 10 times that of the class 10 peers. We construct one overlay topology with $numHC = 10$. We also construct a second overlay topology with $numHC = 3$, wherein class zero comprises peers in class 1, class 1 comprises peers in class {2, 3, 4}, and class 2 comprises peers in class {5, 6, 7, 8, 9, 10}. Table 9 shows the ratios of the performance measures obtained with $numHC = 10$ to those obtained with $numHC = 3$. For instance, a sparse topology with $numHC = 10$ exhibits 23 percent less coverage, 41 percent more bandwidth, 40 percent more latency, 23 percent less fault-tolerance, and 7.7 times more variance in load distribution than a sparse topology with $numHC = 3$. It is clear from Table 9 that it is vital to form peer classes such that each class has a reasonably large population of peers. Recall that our analytical model helps us estimate the HAB expenditure of every node in the system. Thus, one can use the analytical model to hypothetically compare the effect of using different ways of categorizing peers into classes.

4.3 Topology Construction and Maintenance

In this section we evaluate our techniques to construct and maintain a multitier heterogeneity-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: 1) performance in terms of its messaging overhead, and 2) efficacy in terms of the connectivity of the overlay topology.

Bootstrapping. Table 8 shows the *absolute error* and the *relative error* (within brackets) in the node loads obtained using our bootstrap algorithm and the results obtained from

TABLE 9
Importance of Choosing Sizable Peer Classes

| Topology Type | Coverage | Amortized Bandwidth | Amortized Latency | Fault Tolerance | Load Variance |
|---------------|----------|---------------------|-------------------|-----------------|---------------|
| Hierarchical | 0.80 | 1.12 | 1.08 | 0.93 | 12.0 |
| Sparse | 0.77 | 1.41 | 1.40 | 0.77 | 7.7 |
| Dense | 0.79 | 1.25 | 1.26 | 0.75 | 8.1 |

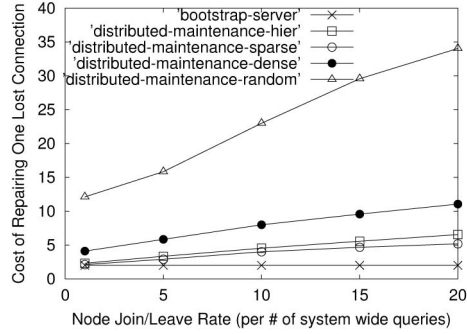


Fig. 16. Topology maintenance cost.

our analytical model. Absolute error in the HAB requirements on nodes is computed as

$$E[(HAB_{computed} - HAB_{experimental})^2],$$

where $E[X]$ denotes the expected value of X , $HAB_{computed}$ denotes the HAB expended by a node as obtained from the analytical model, and $HAB_{experimental}$ is the load on the same node as obtained from our simulation results. Relative error is computed as

$$E\left[\left(\frac{HAB_{computed} - HAB_{experimental}}{HAB_{experimental}}\right)^2\right].$$

We compute the bootstrapping error along three dimensions, namely, the number of nodes, the number of classes, and the fraction of nodes at a randomly chosen class. In our experiment, we incrementally added one node at a time using our bootstrap algorithm to construct a 10,000-node three-tiered topology. Achieving low error margins indicates that our bootstrap algorithm is capable of constructing topologies that are near optimal with respect to minimizing HAB variance.

Topology Maintenance. We measured deviation from the optimal HAB variance values (from our analytical model) while attempting to maintain our multitier topologies. We observed that the errors due to topology maintenance are almost equivalent to the bootstrapping errors; see Table 8.

Fig. 16 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 classes higher than itself. When a node n needs to open a connection to say, some node at class $HC(n) + 1$, it pings those nodes at class $HC(n) + 1$ in its cache and opens a connection with one of them. Observe from Fig. 16, the dense topology pays for maintaining volatile links with nodes at the same class, while a sparse or a hierarchical topology benefits because the nodes have to maintain their connections only with strictly higher class 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 subtree on the failure of a single node; this is because our experiments let the root node of such a disconnected subtree directly contact the bootstrap server to bootstrap itself (and, consequently, the entire subtree) into the system.

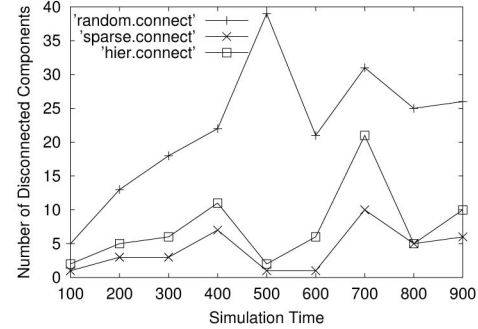


Fig. 17. Topology connectivity.

We also measured the effectiveness of the topology construction and maintenance algorithm by measuring the *disconnectedness* of the overlay topology. The disconnectedness of the overlay topology is measured as the number of disconnected components in the overlay topology. Fig. 17 shows the variation in disconnectedness of our heterogeneity-aware topology (HAT) and a random topology with time. Observe that the disconnectedness of multi-tier topologies is significantly lower (better) than a random topology. It has been observed that a random topology formed by Gnutea is, in fact, a power-law topology and a few powerful nodes are responsible for keeping the overlay topology connected. On the contrary, our structured overlay topologies are systematically constructed and, hence, maintain a highly connected overlay topology.

4.4 Evaluation of HAR

In this section, we present a detailed evaluation of our heterogeneity-aware routing algorithm. First, we show the benefits of HAR on both a random topology and a multitier topology. Second, we compare our routing algorithm with other routing algorithms popularly known in literature, such as iterative deepening (*rdfs*) [20] and broadcast-style routing (*bsr*). Third, we show the ability of our routing algorithm to adapt itself to the dynamics of the P2P network.

Performance Enhancements by HAR. Fig. 18 shows the mean bandwidth (HAB) expended for the execution of a query. For a random topology, HAR shows about 30 percent reduction in bandwidth, while for a heterogeneity-aware multi-tier topology, HAR shows about 35 percent reduction in network bandwidth. The heterogeneity-aware query routing algorithm ensures that the query is delivered to nodes that are more capable of obtaining results for the query. However, one should keep in mind that the bandwidth improvements obtained by HAR are at the cost of introducing a skew in the load distribution; however,

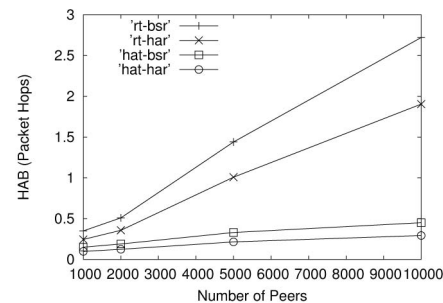


Fig. 18. HAR benefits.

TABLE 10
Comparison of Routing Algorithms: $N = 10,000$, $numHC = 3$, and $initTTL = 7$

| Search THR | HAT-PSR | HAT-ITR-DEEP | HAT-BIASED-RW | HAT-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) |

from our experiments, we observed that HAR typically limits the HAB variance to less than twice the optimal value obtained from our analytical model.

Comparison of Routing Algorithms. Table 10 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 HAB) and amortized latency (shown within brackets). Note that PSR denotes our probabilistic-selective routing algorithm, ITR-DEEP denotes the iterative-deepening technique [20], BIASED-RW denotes a biased random-walk algorithm [10] and BSR denotes the broadcast-style routing algorithm. In general, the heterogeneity-aware topologies (HAT) 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. Also, the focused/biased nature of the PSR algorithm ensures that it performs better than the ITR-DEEP algorithm.

Note that, in the above experiments, the average number of results returned by the HAT-BSR is about 25. We observed that when the threshold is set significantly higher than the average number of results returned by BSR, the threshold-based algorithms (PSR, ITR-DEEP, and BIASED-RW) perform more poorly 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).

Probabilistic Selective Routing with Result Caching. Now, we compare the performance of the search techniques with result caching enabled. A simple LFU-based cache (with a fixed size) was used to evaluate the search techniques with respect to their bandwidth consumption. Fig. 19 shows the plot between bandwidth and threshold. Similarly, the vertical and horizontal lines indicate the results for the *bsr* scheme. Note that when the search threshold is set to the average number of results returned by *bsr* with caching, *psr* shows about 20 percent lower bandwidth consumption than that of iterative deepening.

We also measured the ability of our result-caching scheme to respond to changes in the network topology in the form of peer joins or departures. Referring to Fig. 20, *psr*-stable is the performance of the *psr* algorithm under *steady* conditions; that is when peers join or leave the network randomly at a *steady* rate. We simulated about one join/leave for every 10 queries [20]. Fig. 20 shows that *psr* indeed adapts itself well to small and continuous changes. In fact, the standard deviation in the average number of results obtained is less than 4 in *psr*-stable. Also optimal-stable represents the average number of results obtained by an optimal algorithm on a topology *snapshot*.

We also performed experiments where a larger fraction of peers leave or join the network. We present an anecdotal description of one such experiment. We have measured time in terms of the number of queries executed on the network. *psr*-fluctuate shows the performance of *psr* under the following scenario:

State 1: At time $t = 0$, 20 percent of the peers leave the network. Observe that after a steep fall at $t = 0$, *psr* adapts and stabilizes by $t = 20$. In other words, within 2,000 system-wide queries, *psr* adapts itself to the sudden loss of peers.

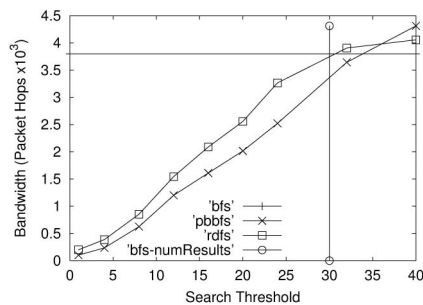


Fig. 19. Result-caching with *psr*.

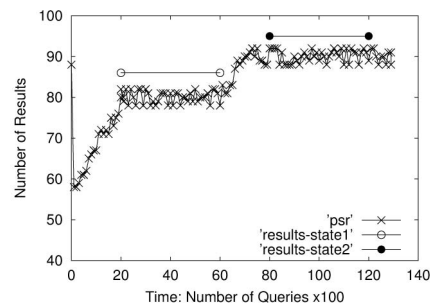


Fig. 20. Adaptability of result-caching when used with *psr*.

State 2: At time $t = 60$, 10 percent of the peers join. Around $t = 70$, *psr* has adapted itself to the new scenario. Therefore, in a little more than 1,000 systemwide queries, *psr* adapts itself to the sudden addition of peers.

For comparison, we have shown *optimal-fluctuate1*, the average number of query results obtained when 20 percent of peers have failed (State 1) using the deterministic strategy, and *optimal-fluctuate2*, the average number of query results after 10 percent of peers have joined (State 2) using the deterministic strategy.

By combining *psr* with result caching, we observe the following from our experiments:

- Result caching increases the number of results obtained while consuming less network bandwidth.
- The probabilistic selective routing is very closely tied to result caching. We observe that with result caching, the *psr* algorithm outperforms the iterative deepening scheme for most threshold values.
- The *psr* algorithm is able to respond effectively to changes in the P2P overlay network.

5 RELATED WORK

In the past few years, research on P2P systems has received a lot of attention. Several research efforts have targeted improving the performance of P2P search [20], [3], [10]. 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, the search schemes proposed in [20], [10] exploit the fact that a typical user is satisfied with a small number of queries, while *Routing Indices* in [3] subsumes 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 [17]. Suggestions have been made to implement caching as a part of Gnutella protocol [9]. But none to our knowledge has exploited the heterogeneity-awareness in P2P topology construction and P2P search techniques.

Several authors have pointed out that peer heterogeneity would be a major stumbling block for Gnutella [13], [15]. Active Virtual Peers (AVPs) [12] introduces the notion of special peers that are responsible for enabling flexibility and adaptability through self-organization. AVP provides an easy to use framework for deploying application-specific topology construction, maintenance and routing algorithms at other peers. Solutions based on superpeer architectures have been proposed in [6] to alleviate the problem of peer heterogeneity. The superpeer architecture can be viewed as a hierarchical topology with $numHC = 2$. Our work not only generalizes $numHC$ to arbitrary values, promoting multitier layered sparse topology over the hierarchical topology, but also provides an analytical model that yields the desired degree (number of connection) information to precisely construct heterogeneity-aware overlay topologies.

Chawathe et al. [2] suggest the use of dynamic topology adaptation that puts most nodes within a short reach of the high-capability nodes. Their topology adaptation scheme defines a level of satisfaction and ensures that high-capability nodes are indeed the ones with high degree. 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 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 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 using the offline solution obtained from our 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.

More recently, the distributed hash table (DHT)-based P2P systems have emerged as a more structured type of P2P network. Examples include Chord [18] and CAN [14]. On one hand, these systems implement a basic DHT functionality and guarantee to locate a *value* associated with a *key* within a bounded number of hops. On the other hand, these systems are complex and require maintaining a hard state. As a result, they have costly protocols for node joins/departures. In addition, these systems, to date, do not have feasible techniques for performing arbitrary keyword-based searches, which is vital to file-sharing applications. Several works, such as topology-aware CAN [14] and ExpressWay [19] attempt to handle heterogeneity in structured networks by ensuring that the neighbors of a node in the overlay network are, indeed, physically close to one another. Such techniques may be augmented into our heterogeneity-aware overlay topologies to further improve its performance.

6 CONCLUSION

The key problem that has plagued Gnutella-like P2P systems is *Peer Heterogeneity*. We have proposed simple yet effective multitier heterogeneity-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 have proposed techniques to structure overlay topologies taking peer heterogeneity into account; such heterogeneity-aware topologies ensure that the performance of the P2P system is not hindered by less powerful peers. Second, we have developed an analytical model to enable us to construct and maintain heterogeneity-aware overlay topologies with good node connectivity and better load balance. Third, we have proposed a probabilistic selective routing algorithm that further reduces the bandwidth consumption of the P2P system. We have used extensive simulation-based experiments and mathematical analysis to construct and evaluate the goodness of heterogeneity-aware topologies and routing algorithms over random topologies and broadcast-styled routing algorithms that disregard peer heterogeneity. Our experimental results show several orders of magnitude improvement in the bandwidth and the load variance metrics for the P2P system. Finally, our design and techniques, being simple and pragmatic, can be easily incorporated into existing systems, such as Gnutella.

ACKNOWLEDGMENTS

This research is partially supported by grants from the US National Science Foundation (NSF) Computer Systems

Research Program, NSF Information Technology Research Program, US Department of Energy Scientific Discovery through Advanced Computing Program, an IBM Shared University Research grant, an IBM Faculty Award, and a Hewlett-Packard Equipment Grant.

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," *Proc. ACM SIGCOMM*, 2003.
- [3] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," *Proc. Int'l Conf. Distributed Computing Systems*, July 2002.
- [4] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems," technical report, Computer Science Dept., Stanford Univ., Oct. 2002.
- [5] G.S. Fishman, *Discrete-Event Simulation*. Springer-Verlag, 2001.
- [6] "Super-Peer Architectures for Distributed Computing," F.S. Inc., <http://www.fiorano.com/whitepapers/superpeer.pdf>, 2004.
- [7] "Kazaa Home Page," <http://www.kazaa.com/>, 2003.
- [8] S. Kirkpatrick, C.D. Gellat, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, no. 4598, May 1983.
- [9] "Improving Gnutella Protocol: Protocol Analysis and Research Proposals," Limewire, http://www9.limewire.com/download/ivkovic_paper.pdf, 2002.
- [10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th Ann. ACM Int'l Conf. Supercomputing*, 2002.
- [11] E.P. Markatos, "Tracing A Large-Scale Peer to Peer System: An Hour in the Life of Gnutella," *Proc. Second IEEE/ACM Int'l Symp. Cluster Computing and the Grid*, 2002.
- [12] H.D. Meer, "Peer-to-Peer Programmability," *Programmable Networks for IP-Service Deployment*, A. Galis, S. Denazis, C. Brou, and C. Klein, eds., chapter 6, pp. 87-107. Artech House Books, 2004.
- [13] S.R. Qin Lv and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?" *Proc. First Int'l Workshop Peer-to-Peer Systems*, 2002.
- [14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. SIGCOMM Ann. Conf. Data Comm.*, Aug. 2001.
- [15] 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, Univ. of Washington, 2001.
- [16] A. Singh, "Mini Project I," <http://www.cc.gatech.edu/~aameek/Projects/mps/mp1.html>, 2002.
- [17] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," <http://www-2.cs.cmu.edu/~kunwadee/research/p2p/paper.html>, 2001.
- [18] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM Ann. Conf. Data Comm.*, Aug. 2001.
- [19] Z. Xu, M. Mahalingam, and M. Karlsson, "Turning Heterogeneity into an Advantage in Overlay Routing," *Proc. IEEE Infocom*, 2003.
- [20] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS '03)*, July 2003.



processing. He is a member of the IEEE.



Ling Liu is an associate professor at the College of Computing at the Georgia Institute of Technology. There, she directs the research programs in the Distributed Data Intensive Systems Lab (DiSL), examining research issues and technical challenges in building large scale distributed computing systems that can grow without limits. Dr. Liu and the DiSL research group have been working on various aspects of distributed data intensive systems, ranging from decentralized overlay networks, exemplified by peer-to-peer computing, data grid computing, to mobile computing systems and location-based services, sensor network computing, and enterprise computing systems. She has published more than 150 international journal and conference articles. Her research group has produced a number of software systems that are either open sources or directly accessible online, among which the most popular are WebCQ and XWRAPElite. Most of Dr. Liu's current research projects are sponsored by the US National Science Foundation, the US Department of Energy, the Defense Advanced Research Projects Agency, IBM, and Hewlett-Packard. She is on the editorial board of several international journals, such as the *IEEE Transactions on Knowledge and Data Engineering*, the *International Journal of Very Large Database Systems (VLDBJ)*, and the *International Journal of Web Services Research*. She has chaired a number of conferences as a program committee chair, a vice program committee chair, or a general chair, including the IEEE International Conference on Data Engineering (ICDE 2004, ICDE 2006, and ICDE 2007), the IEEE International Conference on Distributed Computing (ICDCS 2006), and the IEEE International Conference on Web Services (ICWS 2004). She is a senior member of the IEEE and a member of the IEEE Computer Society.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.



Mudhakar Srivatsa received the BTech degree from the Computer Science and Engineering Department of the Indian Institute of Technology, Madras, India, and is currently a PhD student in the College of Computing at the Georgia Institute of Technology. His research interests are in the security, reliability and performance of large scale distributed systems. He is a student member of the IEEE.