Improving Gnutella Network Performance Using Synthetic Coordinates

André Dufour and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
{adufour, ljilja}@cs.sfu.ca

Abstract— **In this paper, we examine the behavior of the Gnutella peer-to-peer file sharing network and propose a protocol modification to improve its performance. Gnutella exhibits sub-optimal performance in terms of message latency because its overlay topology does not match the underlying physical network. In order to characterize Gnutella's performance, we modified an existing Gnutella simulation framework developed for the *ns-2* network simulator to gather information about query and query hit propagation. We then modified the simulated protocol to use the Vivaldi synthetic coordinate system and to bias neighbor selection to favor nodes that are "close" in the Euclidean sense. Simulations with the adapted Gnutella protocol showed an improvement in both query and query hit propagation times.**

I. Introduction

With the advent of peer-to-peer (P2P) networks, the field of data communications has been radically altered. Since the popularization of the technology through the Napster file sharing network [1], P2P has grown to be the leading source of the Internet traffic [2]. Although file sharing is still its most popular application, P2P technology has found niches in distributed processing [3], [4], online chatting [5], and gaming [6], [7]. P2P networks rely on an application layer overlay for inter-node communication.

Gnutella is one of the most popular P2P file sharing protocols. It is an open standard protocol implemented by many vendors, such as Limewire and Bearshare. In addition to sharing content, thousands of nodes in the Gnutella network collaborate to forward control messages, such as queries, through the overlay topology. Unlike the Napster network [1], which employed a central server to mediate communication between peers, Gnutella is a distributed network. When first connecting to the network, new nodes (known as *servents*) contact a "bootstrap" server to obtain the addresses of a few connected peers. However, further communication is handled through the P2P overlay without relying on servers. Once a new node has the addresses of existing nodes, it attempts to connect to them by sending Gnutella *connect* messages [8]. Two connected nodes are called *neighbors*. Several connection attempts may be necessary because nodes may not be willing to accept new connections or may have left the network. Once a connection has been established, the new node begins sending periodic Gnutella *ping* messages. These probes (not to be mistaken for Internet Control Message Protocol (ICMP) "ping" messages) are used to search for other nodes willing to accept new connections. They are sent by the new node to all its neighbors, which, in turn, *flood* them to all their neighbors. This recursive flooding continues until the ping's time-to-live (TTL) field, which is decremented at each hop, reaches zero. Along the way, any node receiving a ping and willing to accept new connections responds with a *pong* message. The pong back-propagates to the originator of the ping, which may decide to attempt a connection with the pong's sender.

In order to locate content shared in the Gnutella network, nodes must send *query* messages. Queries contain the search criteria (a file name) and are flooded similarly to ping messages. When a node receives a query that matches a resource it shares, it responds with a *query hit* message, which is back-propagated to the query originator. The originator may then decide to download the file from the node that sent the query hit. This is done directly through an HTTP-like protocol [8] and the traffic does not pass through the P2P overlay network. Incidentally, this exchange represents a large portion of the Internet traffic [2].

The Gnutella overlay topology evolves as nodes learn the addresses of other nodes from bootstrap servers and from the nodes they are already connected to. This process does not consider the underlying physical topology and thus may lead to inefficient network utilization [9]. Nodes are as likely to connect to distant nodes as to close ones, which results in longer message latency.

An example of how the Gnutella network's topology may lead to inefficient use of network resources is shown in Fig. 1. All communications in the Gnutella network rely purely on the topological information known at the application layer. Hence, nodes may only send messages directly to their logical neighbors. If node 1 needs to retrieve content stored in node 2, its messages must first pass through node 5. This represents only two hops in the logical topology. In the physical topology, this corresponds to at least 5 hops: either {1, 6, 7, 5, 4, 2} or {1, 3, 4, 5, 4, 2}. The inefficiency is particularly egregious in the second path where the physical link between nodes 4 and 5 is traversed twice. If nodes 1 and 2 had elected to be neighbors in the logical topology, only a single physical hop would have been required and no links would have been traversed twice. This would have resulted in lower message latency and a more efficient use of network bandwidth.

It has been observed that less than 5% of Gnutella overlay links connect nodes that belong to the same autonomous

QShine'06 The Third International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks
August 7-9 2006, Waterloo, ON, Canada © 2006 ACM 1-59593-472-3/06/08…$5.00

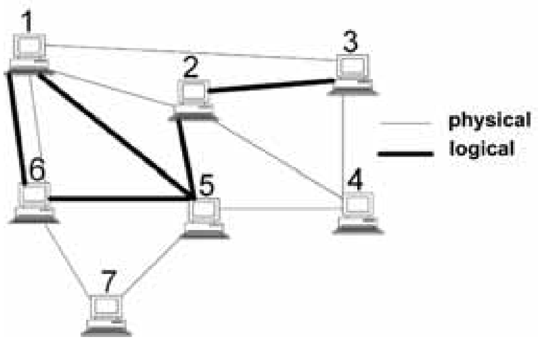

Fig. 1. Physical and logical topologies in P2P communication. The thin lines represent physical connections between hosts and the thick lines represent logical connections between neighbors in the P2P network.

system (AS) [9]. Autonomous systems, being controlled by a single administrative authority, imply a certain notion of locality. Intra-AS communication is often faster than communication between ASs. It is also less expensive for the network operators. Another potential indicator of locality is the domain name hierarchy [9]. The round-trip time (RTT) for communication between two hosts in the same domain is expected to be smaller than for hosts in distinct domains. Thus, if Gnutella hosts established neighbor relationships with hosts in the same domain, they would incur lower communication costs. Ripeanu and Foster [9] analyzed network entropy in order to test if the Gnutella network contains a notion of hierarchy. They define the entropy of a set C of size $|C|$ as

$$E\left(C\right)=\sum_{i=1}^{n}\left(-p_i\log\left(p_i\right)-\left(1-p_i\right)\log\left(1-p_i\right)\right), \qquad (1)$$

where p_i is the probability of randomly selecting a host with domain i and n is the number of distinct domain names [9]. They define the entropy of a network with $|C|$ nodes and k clusters as

$$E\left(C_1,C_2,\ldots,C_k\right)=\sum_{i=1}^{k}\frac{|C_i|}{|C_1|+|C_2|+\ldots+|C_k|}\times E\left(C_i\right). \qquad (2)$$

The entropy with clustering (2) around highly connected nodes in the Gnutella overlay was not lower than the entropy without clustering (1). Hence, Gnutella nodes cluster independently of the domain hierarchy [9].

Improving the way the Gnutella overlay uses the underlying physical infrastructure is desirable in order to achieve better network utilization and improve users' *quality of experience* (QoE). In this paper, we propose to optimize Gnutella's performance by using a neighbor selection algorithm that considers topological information at the physical level as it forms the P2P overlay. It employs the Vivaldi coordinate system [10] to assign to the nodes in the Gnutella network coordinates that predict their inter-node RTT. Based on this coordinate information, nodes select logical neighbors that are close in the physical topology. To compare the performance of the modified and the original Gnutella protocols, we developed the Gnutaldi simulator, which is based on reported work [11].

There are approaches that attempt to improve user QoE by optimizing the search and query propagation mechanisms [12], [13]. The approach proposed in this paper does not modify query forwarding behavior and deals instead with the network topology. While other network coordinate implementations exist in P2P networks, such as Chord [14], they have not been applied to improving the performance of the Gnutella network.

The Vivaldi coordinate system is discussed in Section II. We introduce a new neighbor selection algorithm in Section III. Its drawbacks are discussed in Section IV. In Section V, we discuss the implementation of the Gnutaldi simulator. Simulation results are given in Section VI. Finally, we conclude with Section VII.

II. The Vivaldi Coordinate System

The Gnutella network exhibits a mismatch between the physical and logical (overlay) topologies. This mismatch causes inefficient network utilization as peers establish *neighbor* relationships without considering the underlying physical network. It would be efficient to bias the selection of neighbors to favor nodes that are physically close, thus aligning the P2P overlay with the physical topology. This leads to an improved user QoE because queries and query hits propagate faster.

RTT is commonly employed as a measure of *closeness* in networks. It is the round-trip time it takes for a message to propagate from the sender to the receiver. In wireline networks, RTT is usually of the order of milliseconds. It is not feasible for every node in the Gnutella network to measure the RTT to every other node when evaluating prospective neighbors. This would lead to $O\left(N^2\right)$ ICMP echo messages for a N-node network. Even if that were not prohibitively costly, there is no single resource that stores all the addresses of Gnutella nodes. Hence, it would be impossible to determine all the participants in the network in order to measure a RTT to each one. Nevertheless, when a Gnutella node learns of a potential neighbor, it should be in a position to decide whether it should peer with that node without having to perform additional measurements. There are a number of *network coordinate systems* that provide such a facility.

Network coordinate systems (NCSs) assign coordinates to each node in the system and use distance (Euclidean, Manhattan, Mahanalobis) to estimate the physical distance (RTT) between nodes [10]. Consider a system where coordinates are represented as 4-dimensional vectors. There are two nodes: N_1 with coordinates $(2,3,4,7)$ and N_2 with coordinates $(4,\ 3,\ 1,\ 2)$. Using the Euclidean distance function, the RTT between the two nodes is estimated as

$$\begin{aligned} RTT &= \sqrt{(2-4)^2+(3-3)^2+(4-1)^2+(7-2)^2} \\ &= \sqrt{38}. \end{aligned} \qquad (3)$$

Many coordinate systems rely on fixed infrastructure nodes in order to calculate node coordinates. Global Network Positioning (GNP) [15], Network Positioning System (NPS) [16], and Lighthouse [17] use *landmark* or *beacon* nodes as reference points for coordinate calculation. Network participants derive their coordinates by measuring their RTT to landmarks.

While these systems predict RTTs between nodes with some success, their reliance on fixed infrastructure nodes makes them incompatible with the peer-to-peer paradigm.

There are also coordinate systems that are fully decentralized, with no dependence on infrastructure nodes. Practical Internet Coordinates (PIC) [18] does not rely on fixed landmark nodes. Nevertheless, it is over-sensitive to changing network conditions, which may make it unsuitable in dynamic networks [10] such as P2P. Vivaldi [10] is another decentralized coordinate system that is used by the Chord [14] P2P network's lookup algorithm. It is a system used to assign synthetic coordinates to nodes participating in a network. It does not rely on fixed infrastructure nodes and thus may be suitable for P2P networks. The Vivaldi algorithm employs the Euclidean distance between two nodes' coordinates to estimate the RTT between them (3).

A. *Error Minimization*

The error for a coordinate pair is defined as the difference between the predicted RTT (the coordinates) and the actual RTT [10]. The squared error function is

$$e = \left(R - \|x_i - x_j\|\right)^2 , \tag{4}$$

where R is the actual RTT while x_i and x_j are the two coordinates. If R_{ij} is the actual RTT value between nodes i and j, and x_i and x_j are defined as the coordinates of i and j, then the squared error E for the system is [10]

$$E = \sum_i \sum_j \left(R_{ij} - \|x_i - x_j\|\right)^2 . \tag{5}$$

The Vivaldi algorithm employs the squared error function because it is analogous to spring relaxation in a physical spring-mass system [10]. These associations follow if we consider a spring placed between each pair of nodes for which latency measurements exist [10]: the length of the spring models the distance between the nodes given their current coordinates; the spring rest position occurs when the coordinates predict the RTT with zero error; the potential energy of the spring is the square of its displacement from its rest position; the potential energy of the system is the squared-error function (5).

B. *Spring Relaxation and Coordinate Adjustment*

The Vivaldi algorithm seeks to minimize the potential energy of the spring system. Hence, it models the movements of the nodes under the forces applied by the conceptual springs between them [10].

Let F_{ij} be the force that node j exerts on node i. Hooke's law [19] states that the force is proportional to the spring's displacement from its rest position, and in the opposite direction. Hence, the force is

$$F_{ij} = \left(R_{ij} - \|x_i - x_j\|\right) \times u\left(x_i - x_j\right) , \tag{6}$$

where $\left(R_{ij} - |x_i - x_j|\right)$ is the magnitude of the spring displacement and $u\left(x_i - x_j\right)$ is a unit vector in the direction of the force (pushing i along a line connecting it to j, either closer or farther) [10]. If nodes have identical coordinates, $u\left(x_i - x_j\right)$ is defined as a unit vector in an arbitrary direction [10].

The actual RTT R_{ij} is not known. Hence, nodes adjust their coordinates in response to sampled RTT values learned from communication with other nodes. Based on these samples, nodes allow their coordinates to be "pushed" for a short time δ by the force (6) [10]. For a sample RTT r_{ij} between nodes i and j, node i will adjust its coordinates to

$$x_i = x_i + \delta \times \left(r_{ij} - \|x_i - x_j\|\right) \times u\left(x_i - x_j\right) . \tag{7}$$

C. *Vivaldi Algorithm*

With every message sent, nodes participating in the Vivaldi process append three additional values: their coordinates, their estimate of the error on those coordinates, and a timestamp so that the receiver can calculate the RTT. A node receiving a message will apply the Vivaldi algorithm [10]:

1) *Calculate the credence to be given to the new sample*: If the sample bears a high error, the receiving node will in response adjust its coordinates only slightly. Node movement is also conditioned by the local error. If a node has high local error, it will give more weight to reports from other nodes. The sample weight w [10] is defined as the ratio of the local and sample errors:

$$w = \frac{e_i}{e_i + e_j}, \tag{8}$$

where e_i and e_j are the local and sample errors, respectively.

2) *Calculate the relative error of the sample e_s*: Based on the RTT predicted by the coordinates and the measured RTT [10], calculate:

$$e_s = \frac{\left|\|x_i - x_j\| - RTT\right|}{RTT}. \tag{9}$$

3) *Update the local error*:

$$e_i = e_s \times c_e \times w + e_i \times \left(1 - c_e \times w\right) , \tag{10}$$

where c_e is a tuning parameter.

4) *Update the local coordinates*: Move the timestep δ by a constant proportion c_c of the sample weight (8). The recommended value $c_c = 0.25$ [10] is based on empirical observation. The suggested initial value is $\delta = 1$. The coordinates are updated as [10]

$$\delta = c_c \times w \tag{11}$$

$$x_i = x_i + \delta \times \left(RTT - \|x_i - x_j\|\right) \times u\left(x_i - x_j\right) . \tag{12}$$

The timestep δ employed by the Vivaldi algorithm helps achieve fast convergence and low oscillation by "trusting" nodes with relatively low error more than nodes with high error [10].

D. Vivaldi Accuracy

In a study involving 1,740 hosts in the Internet, the Vivaldi coordinate system (with two Euclidean dimensions and a height component) predicted RTT with a median relative error of 11% [10]. This result was achieved using Internet domain name servers as the nodes for which the RTT was to be predicted. Vivaldi's authors used the King [20] method to measure the actual RTT between each pair of nodes in the set. For example, to measure the RTT between *A* and *B*, the probing host first measures the RTT to *A*. It then requests that *A* resolves a domain served by *B* [10]. The difference between the two times is an estimate of the RTT between *A* and *B*. The RTT was continuously measured (100×10^6 times) over the course of a week and compared with the results produced by the Vivaldi algorithm. The errors observed were as low as GNP [15], which uses fixed infrastructure nodes to help assign synthetic coordinates [10].

III. Neighbor Selection

We present here the proposed neighbor selection algorithm in terms of the syntactic and behavioral modifications that it brings to the Gnutella protocol.

A. Syntactic Modifications

The proposed modification to Gnutella messages include Vivaldi coordinates. It was found that 2D Euclidean coordinates augmented with height vectors provide sufficiently accurate results [10]. We propose that these three coordinate components be inserted in Gnutella binary and text messages.

In addition to communicating nodes' coordinates, the Vivaldi algorithm requires the error of the estimated coordinates and a means to estimate the RTT (such as a "send timestamp"). This timestamp represents the time a message is sent and may employ a common time base used by all Gnutella nodes to synchronize when they join the network. Assuming that the latency is symmetrical, an estimate of the RTT is twice the difference between the timestamp and the time a message is received. We propose to include nodes' estimated coordinate error and send timestamp in all Gnutella messages. The modified structure of the Gnutella binary message header is shown in Table I. This information may be also encoded into text (connection sequence) messages by adding an additional field named *Vivaldi-Data*.

TABLE I

Modified header structure of Gnutella binary messages.

Octets	Description
0-15	Message globally unique identifier.
16	Payload type
17	TTL (time to live)
18	Hops
19-22	Payload length
23-30	Vivaldi *X* coordinate
31-38	Vivaldi *Y* coordinate
39-46	Vivaldi *height* coordinate
47-54	Vivaldi coordinate error
55-62	Vivaldi send timestamp

Gnutella servents ignore headers they do not support in the text messages exchanged during the connection phase [21]. Thus, the use of the additional header *Vivaldi-Data* does not pose a backwards compatibility problem. The proposed binary messages, however, will be incompatible with servents that do not implement proposed modifications. Hence, both servents involved in a connection must agree to use the Vivaldi enhancements during the connection phase when they exchange capability headers. If either servent does not support Vivaldi, then they cannot communicate using the modified protocol.

B. Behavioral Modifications

The proposed Vivaldi enhancements to the Gnutella protocol require modification of servents. The proposed modifications only operate in ultrapeer-to-ultrapeer communication: leaves and legacy Gnutella 0.4 servents do not employ the enhancements. In the two-tiered Gnutella hierarchy, peers were divided into ultrapeers and leaves to conserve the networking and processing resources of leaves [21]. Hence, the proposed modifications are not applied to leaves. We therefore avoid adding extra bytes to the messages sent to leaves and eliminate the burden of processing each message bearing Vivaldi coordinates. New protocol modifications do not apply to legacy peers because they employ an old version of the protocol.

1) Initialization: The coordinates of a node that joins the network are initialized to the origin $(0, 0, 0)$. The node's local error is set to 5×10^6 (an arbitrary large value). These initial values are used every time a node joins the network, regardless of whether it had previously been a participant. This is necessary because node coordinates will *drift* significantly over time due to the highly dynamic nature of the P2P overlay topology. Coordinates held previously by the node are of little use.

2) Coordinate Updates: When a node receives a Gnutella message bearing Vivaldi data, it will use the information to update its coordinates and local error according to the Vivaldi algorithm. It uses twice the difference between the current time and the timestamp of the received message as an estimate of the RTT to the sending node. Vivaldi information is "piggybacked" on Gnutella control traffic, and, hence, nodes' coordinates are continuously updated through the exchange of messages.

3) Message Forwarding: When a Gnutella servent forwards a message to one of its neighbors either through flooding (*ping* or *query*) or back-routing (*pong* or *query hit*), it includes its coordinates, timestamp, and error estimate in the message. The updated values included in the received messages replace the previous values recorded at a node. Nodes need to only consider the Vivaldi information of their immediate neighbors when updating their coordinates because the RTT estimate is only meaningful for nodes that are directly connected in the overlay topology.

4) Optimal Neighbor Selection: The topological mismatch between the overlay and physical topology is addressed by servents electing neighbors that are physically close. Thus, nodes need to judiciously select how to respond to *connect* messages they receive. The number of connections (a finite maximum) a Gnutella servent can accept is decided by the client implementation. It is a small number: of the order of tens rather than hundreds of connections [9]. If this maximum has not been reached, Vivaldi-modified Gnutella clients will accept any connection request, as in the case of standard clients. The receiving client stores an estimate of the distance to the node on the other side of the new connection. The estimate is based on the Euclidean distance to that node's coordinates. Every time a new message with new coordinates is received from that node, the distance estimate is updated. When a connection is terminated, the record for that distance estimate is purged. If a node reaches its maximum allowed number of connections and a new connection request is received, it will consider dropping an existing connection to accommodate the new request. It first estimates the distance to the requesting node based on its coordinates. It then searches its own connection records for the node estimated to be the farthest. If this node is farther than the requesting node, the existing connection is terminated and the new connection is accepted. Otherwise, the new connection request is rejected. Thus, the Gnutella nodes use the Vivaldi information to form neighbor relationships with nodes that are physically close.

A peer initiates a connection request when it has not reached its maximum number of connections. It should, therefore, attempt to connect to any peer it is aware of; no special logic is required in this situation. When a peer receives a positive response to a connection request it initiated, it records a distance estimate to the responding node. The new connection will be eligible for discard if the servent reaches the maximum number of allowed connections.

In summary, when a node has room for more connections, it initiates and accepts connections exactly as a usual Gnutella client would. When the maximum number of connections has been reached, it only accepts connections to peers that are closer than the peers to which it is connected.

IV. Costs and Risks of the Modifications

We discuss here the tradeoffs regarding costs and risks associated with the proposed modifications.

A. Costs of the Modifications

As with most protocol enhancements, the costs are mainly associated with increased message size and additional processing. By adding an additional forty octets to each Gnutella binary message, the header size will be tripled. Even if 32-bit values were used for the coordinates, error, and timestamp, an additional twenty octets would be added (almost doubling the header size). To keep this seemingly enormous cost in perspective, we recall that Gnutella control traffic volumes are orders of magnitude less than the associated file transfer activity, which has not been modified. Hence, even though the amount of control traffic increases, the overall effect in the network is virtually insignificant. The processing costs at the nodes are far more important. Each node must update its coordinates with every message received. Assuming a control traffic rate of 6 kbps [9] and knowing that the vast majority of messages (91%) [9] are queries with a minimum size of 25 bytes, we can estimate a lower bound for the number of coordinate updates as

$$\begin{aligned} U \geq \frac{6{,}000 \text{ bits/sec}}{25 \text{ bytes} \times 8 \text{ bits/byte}} &= 30 \text{ messages/sec} \\ &= 30 \text{ updates/sec.} \end{aligned} \quad (13)$$

While this is a fair number of operations, it is not unreasonable for today's fast processors.

B. Risks of the Modifications

The proposed modifications carry certain risks, in addition to the costs.

One of the most important risks is that the enhancements leave the network vulnerable to malicious nodes. A node that misreported its coordinates, especially with a very low reported error, could mislead other nodes and reduce the accuracy of their coordinates. Furthermore, a node could send connection requests with false coordinates engineered to be close to the target node's coordinates, and thus cause it to discard other legitimate connections. For this proposal to be safely deployed, it would have to be augmented with a trust management algorithm.

Another potential risk is that the dropping of existing connections and the resulting network instability may degrade the user experience and make it harder to locate content quickly.

One final risk worth noting is that by causing nodes to preferentially associate with nodes that are physically close, we may be fragmenting the Gnutella network and limiting nodes' search horizon. If distant nodes are always rejected in favor of closer ones, we may cause disjoint networks to form (one on each continent, for example). If some nodes have not filled all their available connection slots, they would still accept connections from distant servents. Thus, intercontinental connections would therefore still be possible, although less frequent. The performed simulations measure the time required to locate content, which is the ultimate indication of the network's success and the user's QoE.

V. The Gnutaldi Simulator

It is neither feasible nor desirable to implement modified Gnutella clients in a deployed network on any meaningful scale. Hence, we rely on network simulations. In order to observe the effects of the proposed neighbor selection algorithm on the performance of the Gnutella network, we developed a new network simulator: Gnutaldi (Gnutella + Vivaldi). Gnutaldi evolved from the GnutellaSim simulator [22] based on *ns-2* [23]. The choice of the *ns-2* simulator was motivated by a desire to capture packet-level details. It has the disadvantage of a fairly unscalable platform. *NS-2*-based simulations do not scale to the tens of thousands of nodes required to model a realistic Gnutella network. Nevertheless, even simulating small networks with tens of nodes provided a useful test of the proposed modifications.

The GnutellaSim simulator [22] provides a starting point for evaluating the performance of the existing Gnutella network. With only minor modifications, we customized it to collect statistics on query propagation times and the time required for nodes to receive query hits. Therefore, we could establish a baseline for comparison of Gnutella with the proposed modified version.

The Gnutaldi simulator models the modified Gnutella protocol and allows us to observe its characteristics by providing a platform to observe the effect of parameter modifications within the enhanced Gnutella protocol.

VI. Simulation Results

In order to test the modifications to the Gnutella protocol, we created a core synthetic network topology of 50 nodes using the BRITE [24] topology generator and Barabási-Albert model [25]. We then completed the sample network by attaching a random number of nodes (either 1 or 2) to each leaf (nodes with the smallest degree) in the core topology. A similar approach was proposed by the authors of GnutellaSim [11] and the hierarchical transit-stub model [26]. These nodes, newly attached to the leaves, contain the Gnutella servents. They are connected by slower links distributed according to observed peer bandwidths [27]. (Note the bandwidth of the nodes is of little importance here.) The latency for the peer access links is distributed uniformly between 2 and 6 milliseconds. The resulting network contained 92 nodes with 42 Gnutella servents. Because of the scalability limitations of the *ns-2* simulator, it was not possible to simulate networks with a large number of nodes because the simulation times (dominated by query flooding) increase quadratically with the number of nodes. It was possible to examine the behavior of the proposed algorithm even with a small number of network nodes.

We validated the operation of the neighbor selection algorithm and confirmed that connections are indeed dropped when more favorable ones are offered.

A. Convergence of Vivaldi Coordinates

In this subsection, we examine the convergence properties of the Vivaldi coordinate system implemented in the Gnutella protocol.

The experiments reported in the seminal Vivaldi publication [10] show that with as few as 8 neighbors, most nodes exhibited a relative RTT estimate error of less than 20%. This result was obtained using a set of 1,740 DNS servers considered to be rather stable nodes. In order to compare the proposed implementation, we used the Gnutaldi tool to simulate the 92-node network characterized by the parameters given in Table II.

The median relative RTT prediction error as a function of time for a stable network where nodes do not disconnect from the P2P overlay once they have joined is shown in Fig. 2. The neighbor selection algorithm is operating on all the Gnutella servents. The Vivaldi coordinates converge with a median relative error of approximately 5% in the steady-state. We consider the median error rather than the arithmetic mean (for example) because we initialize the error for new nodes to an extremely large value. This outlier value would disproportionately affect the evaluation if the mean were used. Even though the neighbor selection algorithm introduces additional instability in the network, as connections to distant nodes are replaced with connections to closer peers, Fig. 2 indicates that the Vivaldi coordinates converge adequately. Note that this is not a realistic situation: P2P networks are quite dynamic in nature. It is, nevertheless, a solid basis for comparing the implementation of Vivaldi in Gnutella with the Vivaldi experiments conducted on the stable DNS servers [10]. The fact that the coordinates converge with a smaller error than in the original experiments may be attributed to the greater connectivity of the simulated overlay network: the 8 connections allowed for each peer reach a larger proportion of the network than the equivalent 8 connections would in the DNS experiment with a network containing forty times as many nodes. Spikes such as the one observed near 400 seconds are not unexpected as the coordinates converge. When a node connects to a new node, it obtains information from different regions of the network and must adjust its coordinates. During this adjustment period, its median RTT prediction error is high. As the node adjusts its coordinates, the nodes to which it is connected will also have to adjust their coordinates as the Vivaldi spring system attempts to return to a state of mini-

TABLE II
Simulation Parameters for a Stable 42-Peer Network.

Nodes	92
Gnutella servents	42
Maximum number of connections per servent	8
Minimum node start time	0 sec
Maximum node start time	50 sec
Probability of going offline after a successful query	0%
Number of nodes with the desired content	12
Proportion of nodes sending queries	25%
Query interval	100 sec
Simulation time	1,500 sec

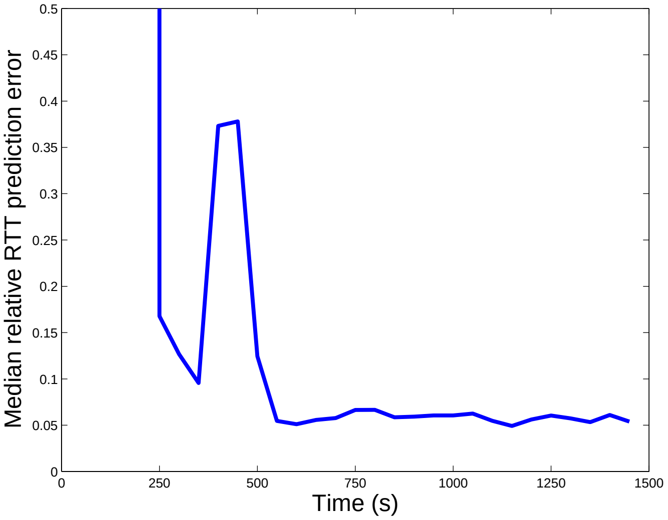

Fig. 2. Median relative RTT prediction error as a function of time for 92-node network with 42 stable Gnutella servents running the neighbor selection algorithm. Each node has up to 8 neighbors.

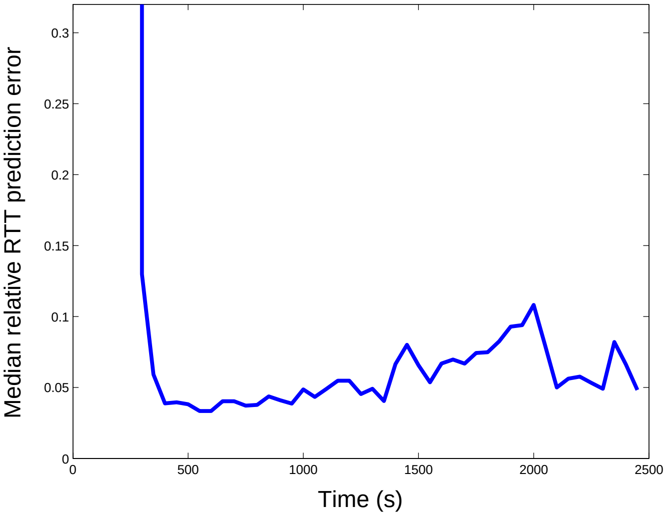

Fig. 3. Median relative RTT prediction error as a function of time for a 92-node network with 42 Gnutella servents running the neighbor selection algorithm. Each node has up to 8 neighbors. The peers have a 10% chance of leaving the network after a successful query.

TABLE III
SIMULATION PARAMETERS FOR A DYNAMIC 42-PEER NETWORK

Nodes	92
Gnutella servents	42
Maximum number of connections per servent	8
Minimum node start time (core overlay)	0 sec
Maximum node start time (core overlay)	50 sec
Earliest allowed node disconnection time	1,000 sec
Minimum new node start time	1,000 sec
Maximum new node start time	1,999 sec
Probability of going offline after a successful query	10%
Number of nodes with the desired content	12
Proportion of nodes sending queries	25%
Query interval	100 sec
Simulation time	2,500 sec

mal potential energy. The *domino* effect of these coordinate adjustments percolates through the network and the median RTT prediction error may rise quickly, only to fall as the coordinates converge based on the new information received from their neighbors.

The convergence of the Vivaldi coordinates for the 92-node network, but with peers joining and leaving the network, is shown in Fig. 3 The parameters are given in Table III. The error varies between 5% and 10%. The magnitude is similar to the error in the original Vivaldi experiments [10]. Evidently, the coordinates in the simulated dynamic network converge to within a moderate error when there is a core network of nodes with reliable coordinates. This is the case in the genuine Gnutella network, where new nodes encounter nodes that existed in the network for some time and had acquired reliable coordinates: the network does not come into existence all at once.

B. Performance Evaluation

In this subsection, we evaluate the performance improvements introduced by the neighbor selection algorithm implemented in the simulated Gnutella network. We consider

TABLE IV
CONNECTIVITY AT 200 SECONDS

Random seed	Average connections per servent		Standard deviation	
	Unmodified	Vivaldi	Unmodified	Vivaldi
7	6.5714	6.8571	2.2265	1.8784
12	7.0476	7.5238	1.3593	0.8136
17	6.7619	7.3333	1.8949	0.7303

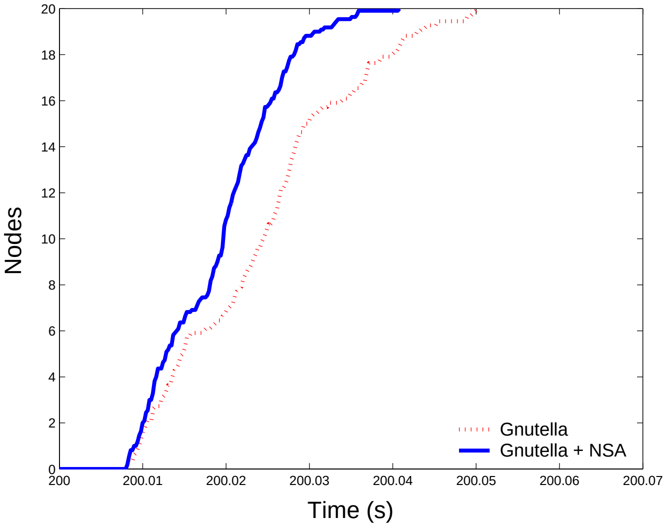

Fig. 4. Average number of nodes reached for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 100%. At this stage in the simulation, only 21 servents are online. At instant 200, when these queries originated, the median relative RTT prediction error is 497,760.252921.

two metrics: the average number of distinct nodes reached by queries and the average number of query hits received by the node that originated the query as functions of time. These quantities provide an indication of the users' QoE when using the Gnutella network. The goal is that queries reach nodes faster and that query hits are promptly received. In each simulation, one quarter of the Gnutella servents were randomly selected to generate queries at regular 100-second intervals. To ensure a valid comparison between the unmodified Gnutella simulation and the simulation using the neighbor selection algorithm, in both experiments the identical set of nodes was selected to send queries. We observed the average for both QoE metrics for every 100-second flight of queries.

The reported simulation results are based on the network described in Table III. These values approximate realistic peer behavior [22], to the extent that it is possible to capture it in a small simulated network. The unmodified Gnutella protocol and the Gnutella protocol augmented with the neighbor selection algorithm (NSA) are compared in each figure.

The performance of the neighbor selection algorithm and the unmodified Gnutella protocol for queries sent at instant 200 sec is compared in Figs. 4 and 5. They show the average number of unique nodes reached by queries and the average number of query hits received as functions of time. At instant 200, all nodes in the network are still very new, and their coordinates have not yet converged to values that reliably predict the RTT between servents. The median relative RTT

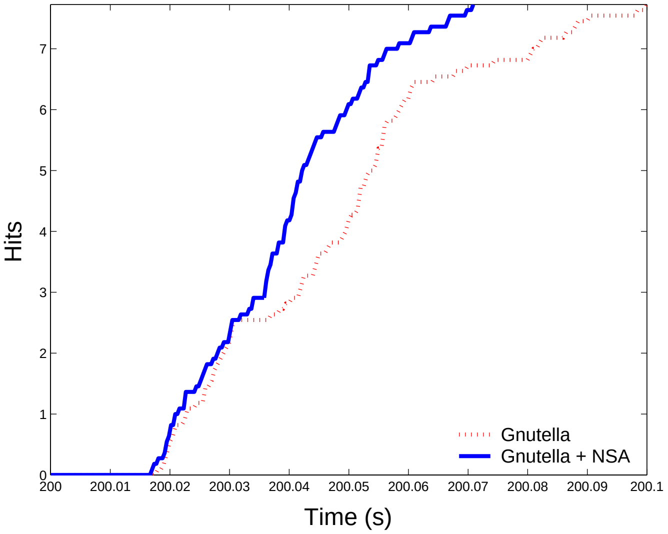

Fig. 5. Average number of query hits received for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 100%. At this stage in the simulation, only 21 servents are online. At instant 200, when these queries originated, the median relative RTT prediction error is 497,760.252921.

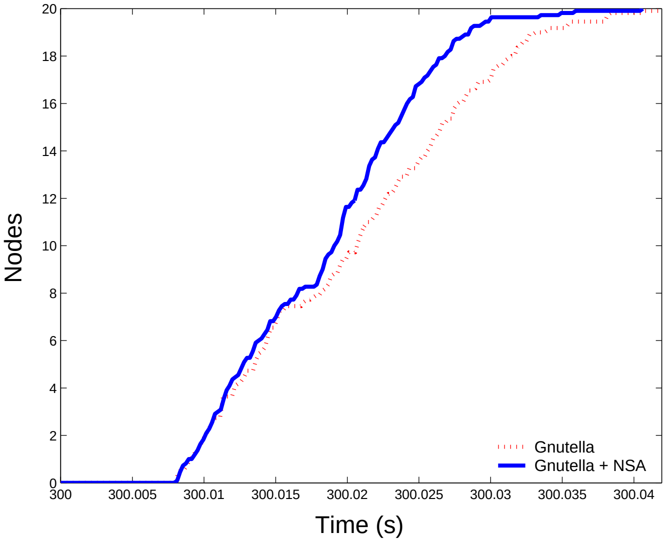

Fig. 6. Average number of nodes reached for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 10%. At this stage in the simulation, only 21 servents are online. At instant 300, when these queries originated, the median relative RTT prediction error is 0.122698.

prediction error at this instant is 497,760.252921. Hence, we did not expect any performance improvement due to the neighbor selection algorithm because any decisions made by peers based on their coordinates would be ill informed. However, both figures show that the neighbor selection algorithm outperforms the unmodified Gnutella protocol. Two repeated simulations, with random seeds equal to 12 and 17, confirm the result. Closer scrutiny of the connection dynamics in the simulations reveals that the neighbor selection algorithm increases connectivity in the network. Table IV shows the average number of connections per servent at instant 200 in the three simulation runs. It also captures the standard deviation of the number of connections. The results show that the neighbor selection algorithm (Vivaldi) increases connectivity in the network. This is not surprising, since the algorithmic modifications to Gnutella give new nodes a greater chance of finding a neighbor to connect to, since nodes with no available connections will consider the possibility of dropping existing connections to accept a request from a closer node. Although the connection success for the new node leads to a concomitant connection drop for another node, it may increase overall connectivity since the dropped node, by virtue of having been in the network longer, may have learned of many other nodes to which it may easily connect. This would not have been the case for the new node. Moreover, the standard deviation of the number of connections is much smaller with the modified Gnutella protocol, as shown in Table IV. This is again because new nodes may “steal” connections from existing nodes with many neighbors, leading to a tighter distribution of the number of connections. The greater connectivity and fewer outlier nodes (fewer bottleneck nodes with a very low number of connections) lead to the improved performance observed at 200 seconds, even though the median relative RTT prediction error is rather high.

The performance of the unmodified Gnutella protocol and the neighbor selection algorithm are compared based on the chosen two metrics: average number of nodes reached and average number of query hits received, as functions of time. The comparison is shown in Figs. 6 and 7. In both cases the neighbor selection algorithm outperforms the unmodified Gnutella protocol: queries traverse the network faster and query hits are received faster. The neighbor selection algorithm yields benefits even though the median relative RTT prediction error is above 10%. These results are similar to results reported for the stable 42-node network described in Table II. This is as expected: nodes are forbidden to leave the network because period prior to 1,000 seconds is used to build a stable Gnutella *core*.

The interval between 1,000 and 2,000 seconds most closely resembles real-world network dynamic conditions: nodes are joining and leaving the network at random intervals [28]. The results for times 1,400 and 1,900 seconds are shown in Figs. 8, 9, 10, and 11. Again, the neighbor selection algorithm outperforms the original Gnutella protocol. The results shown in Fig. 9 are slightly equivocal, since the first query hits are received faster without the modifications. However, the remaining query hits are received faster using the neighbor selection algorithm.

The neighbor selection algorithm leads to performance improvements of the order of tens of milliseconds. At 1,900 seconds, the originating nodes receive the last query hit on average 10 milliseconds earlier using the neighbor selection algorithm. It would appear that this is an insignificant difference that would make no difference to the users’ QoE. Is it worth the extra overhead involved in using the neighbor selection algorithm to achieve this meager improvement? Note

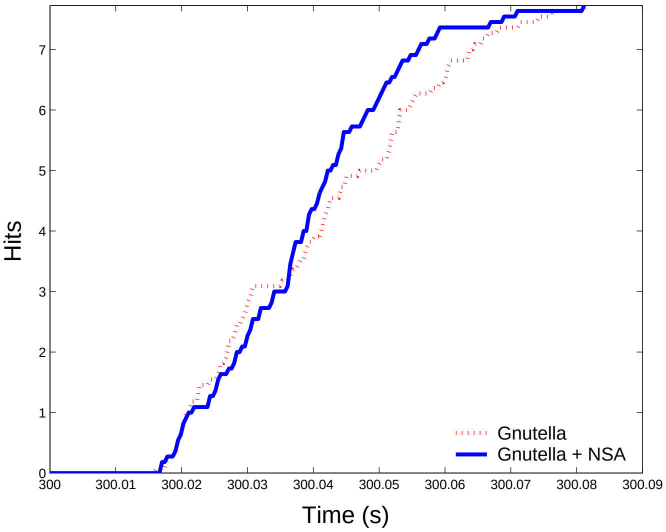

Fig. 7. Average number of query hits received for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 10%. At this stage in the simulation, only 21 servents are online. At instant 300, when these queries originated, the median relative RTT prediction error is 0.122698.

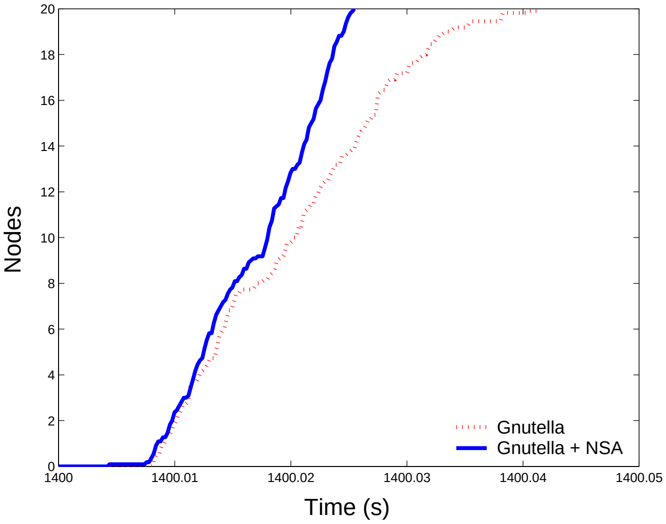

Fig. 8. Average number of nodes reached for queries sent with 42 dynamic Gnutella servents at instant 1,400. At this instant, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.051884.

that we simulated a very small network with only 92 nodes. The Internet has millions of nodes and Gnutella has of the order of 1 million participating hosts. End-to-end latencies in the Internet are much greater than those we simulated and may reach over hundreds of milliseconds, especially across AS boundaries. The potential for performance improvement based on locality is much greater than in simulation. We were able to observe a consistent performance improvement with only 42 servents illustrates that performance gains may be achieved by constructing an overlay that accounts for the underlying physical network. It is difficult to quantify the scale of the improvements the proposed modifications would yield in a real-world deployment. However, since they are quite measurable in a 92-node network, they are likely to positively impact users' QoE in a realistically sized deployment. Furthermore, an overlay that maps well to the underlying physical network would reduce the P2P traffic on service providers' networks, leading to reduced congestion and, ultimately, to better performance for network applications. While we have not modeled the transfer of content that users can initiate after a successful query, it would be significantly faster for users to download content from a node that is close. Use of Vivaldi coordinates provides the method to determine this proximity without explicit measurements. Due to the high traffic volumes of P2P transfers, downloading from a peer that is close in the physical topology would reduce network congestion by utilizing the physical network more efficiently.

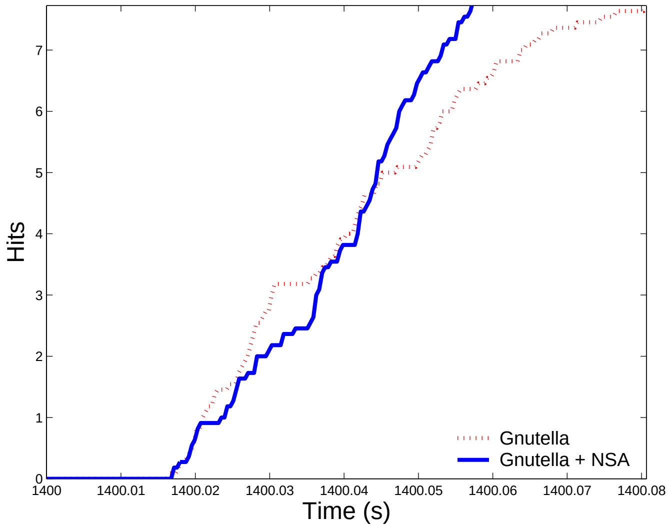

Fig. 9. Average number of query hits received for queries sent with 42 dynamic Gnutella servents at instant 1,400. At this instant, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.051884.

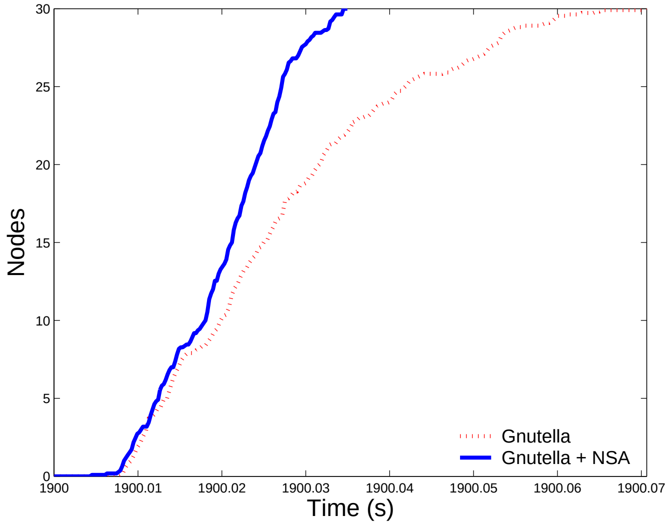

Fig. 10. Average number of nodes reached for queries sent with 42 dynamic Gnutella servents at instant 1,900. At this instant, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.084133.

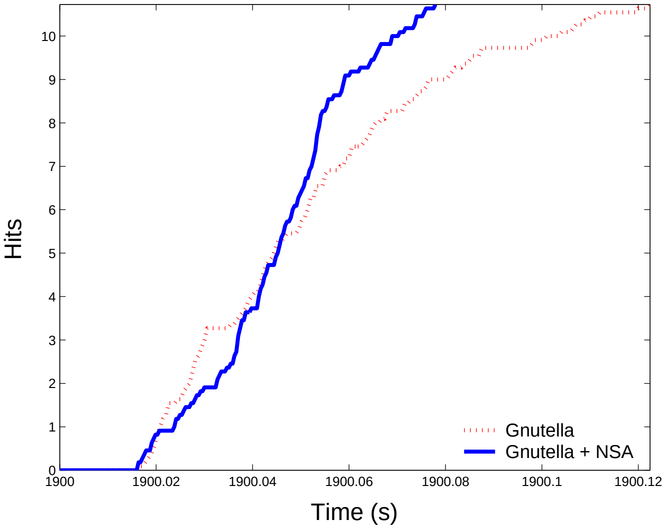

Fig. 11. Average number of query hits received for queries sent with 42 dynamic Gnutella servents at instant 1,900. At this instant, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.084133.

Simulation results show that the proposed neighbor selection algorithm to form the Gnutella overlay based on physical topology information achieves a greater QoE for users by reaching more nodes faster with queries and receiving query hits more quickly.

VII. Conclusions

In this paper, we proposed a neighbor selection algorithm for the Gnutella peer-to-peer network to improve the performance of queries and query hits and, ultimately, provide a better QoE to users. The algorithm employs the Vivaldi coordinate system in order to assign synthetic coordinates to each node participating in the P2P overlay. These coordinates are then used by a peer in the network to choose neighbors: according to the Vivaldi coordinates, nodes will preferentially select neighbors close to them in the physical topology.

The performance of the proposed algorithm was characterized using Gnutaldi, a new network simulator based on the existing GnutellaSim and *ns-2* simulators. We used Gnutaldi to simulate a small Gnutella network. The Vivaldi coordinates converged with a low error. We also observed that in a network using the neighbor selection algorithm, queries reached nodes faster and query hits were returned to the originator more quickly. The simulation results indicate that the proposed neighbor selection algorithm improves users' QoE.

References

[1] Napster. (2005) All the music you want. Any way you want it. [Online]. Available: http://www.napster.com.

[2] G. Bennett. (Dec. 30, 2003) Controlling P2P traffic. [Online]. Available: http://www.lightreading.com/document.asp?doc_id=44435.

[3] V. Pande, "Folding@home: advances in biophysics and biomedicine from world-wide grid computing," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO, Apr. 2005, pp. 101–107.

[4] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "Seti@home-massively distributed computing for seti," *Computing in Science & Engineering*, vol. 3, no. 1, pp. 78–83, Jan. 2001.

[5] Icq. (2005) Icq.com–community, people search and messaging service! [Online]. Available: http://www.icq.com/.

[6] A. El Saddik and A. Dufour, "Peer-to-peer suitability for collaborative multiplayer games," in *Proc. Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications*, Delft, Netherlands, Oct. 2003, pp. 101–107.

[7] A. El Saddik and A. Dufour, "Peer-to-peer communication through the design and implementation of xiangqi," in *Proc. International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, Aug. 2003, pp. 1309–1313.

[8] Gnutella. (2003) The annotated Gnutella protocol specification v0.4. [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[9] M. Ripeanu and I. Foster, "Mapping the Gnutella network," in *Proc. 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002, pp. 85–93.

[10] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *Proc. SIGCOMM'04*, Portland, OR, Aug. 2004, pp. 15–26.

[11] Q. He. (2005) Packet-level peer-to-peer (Gnutella) simulation. [Online]. Available: http://www.cc.gatech.edu/computing/compass/gnutella/usage.html.

[12] S. Jiang and X. Zhang, "FloodTrail: an efficient file search technique in unstructured peer-to-peer systems," in *Proc. IEEE GLOBECOM'03*, San Francisco, CA, Dec. 2003, pp. 2891–2895.

[13] M. W. Jiang, M. M. Morgado Costa, J. M. Almeida, and V. A. F. Almeida, "Using locality of reference to improve performance of peer-to-peer applications," in *Proc. ACM WOSP'04*, Redwood City, CA, Jan. 2004, pp. 216–227.

[14] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. K. F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[15] T. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM'02*, New York, NY, June 2002, pp. 170–179.

[16] T. Ng and H. Zhang, "A network positioning system for the Internet," in *Proc. of the 1st USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, CA, Mar. 2004, pp. 141–154.

[17] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *Proc. 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, Feb. 2003. pp. 278–291.

[18] M. Costa, M. Castro, A. Rowstron, and P. Key, "Pic: practical Internet coordinates for distance estimation," in *Proc. 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, Mar. 2004, pp. 178–187.

[19] H. Benson, *University Physics*, rev. ed. Hoboken, NJ: John Wiley & Sons, 1995, p. 286.

[20] K. Gummadi, S. Saroiu, and S. King, "Estimating latency between arbitrary Internet end hosts," in *Proc. SIGCOMM'02*, Pittsburgh, PA, Nov. 2002, pp. 5–18.

[21] T. Klingberg and R. Manfredi. (2002) Gnutella protocol development. [Online]. Available: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.

[22] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto, "Mapping peer behavior to packet-level details: a framework for packet-level simulation of peer-to-peer systems," in *Proc. 11th IEEE/ACM International Symposium on MASCOTS*, Orlando, FL, Oct. 2003, pp. 71–78.

[23] ns-2. (2005) The network simulator. [Online]. Available: http://www.isi.edu/nsnam/ns/.

[24] A. Medina, I. Matta, and J. Byers, "BRITE: a flexible generator of Internet topologies," in *Proc. MASCOTS*, Cincinnati, OH, Aug. 2001, pp. 346–353.

[25] A. Barabasi, R. Albert, and H. Jeon, "Scale-free characteristics of random networks: the topology of the World Wide Web," *Physica A*, vol. 281, no. 1-4, pp. 69–77, June 2000.

[26] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM'02*, San Francisco, CA, Mar. 1996, pp. 594–602.

[27] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 2002, pp. 156–170.

[28] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, Apr. 2003, pp. 2188–2198.