OVERLAY INFRASTRUCTURE SUPPORT FOR INTERNET APPLICATIONS

By

ZHAN ZHANG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2007

To my family

ACKNOWLEDGMENTS

I am grateful for the help I have received in writing this dissertation. First of all, I would like to thank my advisor Prof. Shigang Chen for his guidance and support throughout my graduate studies. Without the numerous discussions and brainstorms with him, the results presented in this thesis would never have existed.

I am grateful to Prof. Liuqing Yang, Prof. Randy Chow, Prof. Sartaj Sahni, and Prof. Ye Xia for their guidance and encouragement during my years at the University of Florida (UF).

I am thankful to all my colleagues in Prof. Chen's group, including Liang Zhang, MyungKeun Yoon, Ying Jian, and Ming Zhang. They provide valuable feedback for my research. I would like to thank many people in the Computer and Information Science and Engineering (CISE) Department for their help in my research work.

Last but not least, I am grateful to my family for their love, encouragement, and understanding. It would be impossible for me to express my gratitude towards them in mere words.

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

OVERLAY INFRASTRUCTURE SUPPORT FOR INTERNET APPLICATIONS

By

Zhan Zhang

May 2007

Chair: Shigang Chen
Major: Computer Engineering

Overlay networks have gained a lot of popularity, and are now being considered in many application domains such as content distribution, conferencing, gaming, etc. These applications rely on the support from underlying overlay infrastructures. How to design overlay infrastructures to satisfy requirements from different applications, such as resource-sharing systems and application-level multicast, is largely an open problem. My dissertation develops several techniques in the design of overlay infrastructure to facilitate Internet applications.

A distributed hybrid query scheme is proposed for resource-sharing systems. It combines Markov random walks and peer clustering to achieve a better tradeoff. The scheme has a short response time for most of the queries that belong to the same interest group, while still maintaining a smaller network diameter. More important, we propose a totally distributed clustering algorithm, which means better resilience to network dynamics.

A new incentive scheme is proposed to support cooperative overlays, in which the amount of a node can benefit is proportional to its contribution, malicious nodes can only attack others at the cost of their own interests, and a collusion cannot gain advantage by cooperation. Furthermore, we have designed a distributed authority infrastructure and a set of protocols to facilitate the implementation of the scheme in peer-to-peer networks, which have no trusted authorities to manage each peer's history information.

Moreover, we propose two capacity-aware multicast systems that focus on host heterogeneity, any source multicast, dynamic membership, and scalability. We extend Chord and Koorde to be capacity-aware. We then embed implicit degree-varying multicast trees on top of the overlay network and develop multicast routines that automatically follow the trees to disseminate multicast messages. The implicit trees are well balanced with workload evenly spread across the network. We rigorously analyze the expected performance of multi-source capacity-aware multicasting, which was not thoroughly addressed in any previous work. We also perform extensive simulations to evaluate the proposed multicast systems.

CHAPTER 1
INTRODUCTION

1.1 Overlay Networks

Overlay networks have recently attracted a lot of attention and are now being considered in many application domains such as content distribution, conferencing, gaming, etc.

An overlay network is a computer network which is built on top of another network. Each node in an overlay network maintains pointers to a set of neighbor nodes. These pointers are used both to maintain the overlay and to implement application functionality, for example, to locate content stored by overlay nodes. Two neighbors in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet.

Manually configured static overlays are nothing new. A salient, modern feature of overlay networks is that they can be made to autonomically self-organize which provides great ease of deployment.

Resource sharing and application-level multicast are two of major applications in overlay networks, and we study the infrastructure support for these applications from following three aspects.

Query schemes for lookup services: The basic overlay applications are resource-sharing systems, such as grid computing, and file-sharing peer-to-peer networks. The core operation in these systems is an efficient lookup mechanism for locating resources. The fundamental challenge is to achieve faster response time, smaller network diameter, better resilience to network dynamics, and lower overhead. The overlays need to be constructed with good search properties, such as clustering the nodes with similar interests, and the query scheme needs to utilize the properties to support lookup efficiently.

Incentive schemes to encourage cooperation: In cooperative overlay networks, a node is allowed to consume resources from other nodes, and is also expected to share its resources with the community. Today's peer-to-peer networks suffer from the problem of free-riders, that consume resources in the network without contributing anything in return. Originally it was hoped that users would be altruistic, "from each according to his abilities, to each according to his needs." In practice, altruism breaks down as networks grow larger and include more diverse users. This leads to a "tragedy of the commons," where individual players' self interest causes the system to collapse.

Overlay multicast for group communication: Overlay multicast is becoming a promising alternative for group communications, because the global deployment of IP multicast has been slow due to the difficulties related to heterogeneity, scalability, manageability, and lack of a robust inter-domain multicast routing protocol. Even though overlay multicast can be implemented on top of overlay unicast, they have very different requirements, because in overlay unicast, low-capacity nodes only affect traffic passing through them and they create bottlenecks of limited impact. In overlay multicast, all traffic will pass all nodes in the group, and the multicast throughput is decided by the node with the smallest throughput, particularly in the case of reliable delivery. The strategy of assigning an equal number of children to each intermediate node is far from optimal. If the number of children is set too big, the low-capacity nodes will be overloaded, which slows down the entire session. If the number of children is set too small, the high-capacity nodes will be under-utilized. In such systems, the network heterogeneity, scalability, manageability must be well-addressed,

We focus on how to design overlay infrastructures to address these challenges. For resource sharing systems, we present a clustering algorithm and a labeling algorithm to cluster members within the same interest group, and propose a efficient query scheme to deliver a better tradeoff between communication overhead and response time. Moreover, we propose a new incentive scheme, which is particularly suitable to overlay networks without any centralized authority. Our incentive scheme is more resilient against various attacks, especially launched by a large number of colluding nodes.

With respect to overlay multicast, we propose two capacity-aware overlay systems that support distributed applications requiring multiple-source multicast with dynamic membership.

1.2 Related Work

The fundamental challenge of constructing an overlay network for resource-sharing systems such as peer-to-peer networks is to achieve faster response time, smaller network diameter, better resilience to network dynamics, and higher security. Structured P2P networks have been proposed by many researchers [1, 2, 3, 4, 5, 6, 7], in which distributed hash tables (DHT) are used to provide data location management in a strictly structured way. Whenever a node joins/leaves the overlay, a number of nodes need to update their routing tables to preserve desirable properties for fast lookup. While structured P2P networks can offer better performance in response time and communication overhead for query procedures, they suffer from the large overhead for overlay maintenance due to network dynamics. In addition, DHTs are inflexible in providing generic keyword searches because they have to hash the keys associated with certain objects [8].

Unstructured P2P networks such as Gnutella rely on a random process, in which nodes are interconnected in a random manner. The randomness offers high resilience to the network dynamics. Basic unstructured networks rely on flooding for users' queries, which is expensive in computation and communication overhead. Consequently, scalability has always been a major weakness for unstructured networks. Even with the use of super nodes in Morpheus and KaZaA, the traffic is still high, and even exceeds web traffic.

Searching through random walks are proposed in [8, 9, 10], in which incoming queries are forwarded to the neighbor that is chosen randomly. In random walks, there is typically no preference for a query to visit the most possible nodes maintaining the needed data, resulting in long response time.

Interest-based shortcut [11] exploits the locality of interests among different nodes. In this approach, a peer learns its shortcuts by flooding or passively observing its own traffic. A peer ranks its shortcuts in a list and locates content by sequentially asking all of the shortcuts on the list from the top, until content is found. The basic principle behind this approach is that a node tends to revisit accessed nodes again since it was interested in the data items from these nodes before. The concept of interest similarity is vague, and it is difficult to make a subtle, quantitative definition based on it. In addition, it may cause new problems.

Another challenge in resource sharing systems is the incentive schemes. Today's peer-to-peer networks suffer from the problem of free-riders, that consume resources in the network without contributing anything in return. Originally it was hoped that users would be altruistic, "from each according to his abilities, to each according to his needs." In practice, altruism breaks down as networks grow larger and include more diverse users. This leads to a "tragedy of the commons," where individual players' self interest causes the system to collapse.

To reduce free-riders, the systems have to incorporate incentive schemes to encourage cooperative behavior. Some recent works [12, 13, 14, 15, 16, 17] propose **reputation** based trust systems, in which each node is associated with a reputation established based on the feedbacks from others that it has made transactions with. The reputation information helps users to identify and avoid malicious nodes. An alternative is **virtual currency** schemes [18, 19, 20], in which each node is associated with a certain amount of money. Money is deducted from the consumers of a service, and transferred to the providers of the service after each transaction.

Both types of schemes rely on authentic measurement of service quality and unforgeable reputation/money information. Otherwise, selfish/malicious nodes may gain advantage based on false reports. For example, a consumer may falsely claim to have not received service in order to pay less or defame others. More seriously, malicious nodes may collude in cheating in order to manipulate their information. Several algorithms are proposed to address these problems. They either analyze statistical characteristics of the nodes' behavior patterns and other nodes' feedbacks [13, 21], or remove the underlying incentive for cheating [22]. In order to apply these algorithms, the nodes' history information must be managed by a central authority, which is not available in typical peer-to-peer networks.

Some other works [23, 24] find circular service patterns based on the history information shared among trusted nodes. Each node in a service circle has chance to be both a provider and a consumer. The communication overhead for discovering service circles is very high, which makes these schemes not scalable. In addition, nodes belonging to different interest groups have little chance to cooperate because service circles are unlike to form among them.

Besides the resource sharing systems, application-level multicast is another promising applications in overlay networks. Many research papers [25, 26, 27, 28] pointed out the disadvantages of implementing multicast at the IP level [29], and argued for an application-level overlay multicast service. More recent work [30, 31, 32, 33, 34, 35, 36] studied overlay multicast from different aspects.

To handle dynamic groups and ensure scalability, novel proposals were made to implement multicast on top of overlay networks. For example, Bayeux [37] and Borg [38] were implemented on top of Tapestry [3] and Pastry [4] respectively, and CAN-based Multicast [39] was implemented based on CAN [2]. El-Ansary et al. studied efficient broadcast in a Chord network, and their approach can be adapted for the purpose of multicast [40]. Castro et al. compares the performance of tree-based and flooding-based multicast in CAN-style versus Pastry-style overlay networks [41].

These systems assume each node has the same number of children. Host heterogeneity is not addressed. Even though overlay multicast can be implemented on top of overlay unicast, they have very different requirements. In overlay unicast, low-capacity nodes only affect traffic passing through them and they create bottlenecks of limited impact. In overlay multicast, all traffic will pass all nodes in the group, and the multicast throughput is decided by the node with the smallest throughput, particularly in the case of reliable delivery. The strategy of assigning an equal number of children to each intermediate node is far from optimal. If the number of children is set too big, the low-capacity nodes will be overloaded, which slows down the entire session. If the number of children is set too small, the high-capacity nodes will be under-utilized. To support efficient multicast, we should allow nodes in a P2P network to have different numbers of neighbors.

Shi et al. proved that constructing a minimum-diameter degree-limited spanning tree is NP-hard [42]. Note that the terms "degree" and "capacity" are interchangeable in the context of this dissertation. Centralized heuristic algorithms were proposed to balance multicast traffic among multicast service nodes (MSNs) and to maintain low end-to-end latency [42, 43]. The algorithms do not address the dynamic membership problem, such as MSN join/departure.

There has been a flourish of capacity-aware multicast systems, which excel in optimizing single-source multicast trees but are not suitable for multi-source applications such as distributed games, teleconferencing, and virtual classrooms, which are the target applications of our algorithms. Bullet [44] is designed to improve the throughput of data dissemination from one source to a group of receivers. An overlay tree rooted at the source must be established. Disjoint data objects are disseminated from the source via the tree to different receives. The receivers then communicate amongst themselves to retrieve the missing objects; these dynamic communication links, together with the tree, form a mesh, which offers better bandwidth than the tree alone. Overlay multicast network infrastructure (OMNI) [45] dynamically adapts its degree-constrained multicast tree to minimize the latencies to the entire client set. Riabov et al. proposed a centralized constant-factor approximation algorithm for the problem of constructing a single-source degree-constrained minimum-delay multicast tree [46]. Yamaguchi et al. described a distributed algorithm that maintains a degree-constrained delay-sensitive multicast tree for a dynamic group [47]. These algorithms are designed for a single source and not suitable when there are many potential sources (such as in distributed games). Building one tree for each possible source is too costly. Using a single tree for all sources is also problematic. First, a minimum-delay tree for one source may not be a minimum-delay tree for other sources. Second, the single-tree approach concentrates the traffic on the links of the tree and leaves the capacities of the majority nodes (leaves) unused, which affects the overall throughput in multi-source multicasting. Third, a single tree may be partitioned beyond repair for a dynamic group.

1.3 Contribution

There are three major contributions in this dissertation. First of all, we propose an efficient distributed hybrid query scheme for unstructured peer-to-peer networks [48]. Second, we propose an incentive scheme to encourage members to contribute to the community [49, 50]. Third, we propose two overlay infrastructures to support application-level multicast in heterogeneous environment [51, 52].

1.3.1 A Distributed Hybrid Query Scheme to Speed Up Queries in Unstructured Peer-to-Peer Networks

We propose a hybrid query scheme, which deliver a better tradeoff between communication overhead and response time.

- We define a metric, independent of any global information, to measure the interest similarity between nodes. Based on the metric, we propose a clustering algorithm to cluster nodes sharing similar interests with small overhead, and fast convergence.

- We propose a distributed labeling algorithm to explicitly capture the borders of clusters without any extra communication overhead.

- We propose a new query scheme, which is able to deliver a better tradeoff among response time, communication overhead, and the ability to locate more resources by mixing inter-cluster queries and intra-cluster queries.

1.3.2 A Distributed Incentive Scheme for Peer-to-Peer Networks

We propose a new incentive scheme to encourage end users to contribute to the systems, which is suitable to the network without any centralized authority, and is more resilient against various attacks, especially launched by a large collusion. Furthermore, we have designed a distributed authority infrastructure and a set of protocols to implement the scheme in peer-to-peer networks. More specifically, the scheme proposed has the following advantages over the conventional approaches:

We propose a new distributed incentive scheme, which combines reputation and virtual money. It is able to strictly limit the damage caused by malicious nodes and their colluding groups. The following features distinguish our scheme from others.

• The benefit that a node can get from the system is limited by its contribution to the system. The members in a colluding group cannot increase their total money or aggregate reputation by cooperation, regardless of the group size, and malicious nodes can only attack others at the cost of their own interest.

• We design a distributed authority infrastructure to manage the nodes' history information with low overhead and high security.

• We design a key sharing protocol and a contract verification protocol based on the threshold cryptography to implement the proposed distributed incentive scheme.

1.3.3 Capacity-Aware Multicast Algorithms on Heterogeneous Overlay Networks

We extend Chord, and Koorde to support application-level multicast, which has following properties.

• Capacity awareness: Member hosts may vary widely in their capacities in terms of network bandwidth, memory, and CPU power. Some are able to support a large number of direct children, but others support few.

• Any source multicast: The system should allow any member to send data to other members. A multicast tree that is optimal for one source may be bad for another source. On the other hand, one tree per member is too costly.

• Dynamic membership: Members may join and leave at any time. The system must be able to efficiently maintain the multicast trees for a dynamic group.

• Scalability: The system must be able to scale to a large Internet group. It should be fully distributed without a single point of failure.

CHAPTER 2
A HYBRID QUERY SCHEME TO SPEED UP QUERIES IN UNSTRUCTURED PEER-TO-PEER NETWORKS

2.1 Motivation

2.1.1 Problems in Prior Work

Small communication overhead and short response time are the two main concerns in designing efficient query schemes in peer to peer networks. Current approaches suffer various problems in achieving a better tradeoff between communication overhead and response time due to the blindness in searching procedures.

Flooding: Flooding [53, 54], is a popular query scheme to search a data item in fully unstructured P2P networks such as Gnutella. While flooding is simple and robust, its communication overhead, that is, the message number, increases exponentially with the hop number. In addition, most of these messages visit the nodes that have been searched in the same query, and they can be regarded as duplicate messages. Consequently, communication overhead and scalability are always the main problems in the flooding approach.

Random Walks: Random walks [8, 9, 10, 55] rely on query messages randomly selecting their next hops among neighbors to reduce the communication overhead. A query may have to go through many hops before it successfully locates the queried data item. Consequently, this approach takes a long time. If the networks are well-clustered (nodes with similar interests are densely connected), it is expected that the query latency can be reduced significantly. However, it is not true, because the chance of a random walk message escaping out of the original cluster increases exponentially with the ratio r of the inter-cluster edge number to the intra-cluster edge number, as shown in Figure 2-1.

In the case of a network with a small value of r (e.g., $r < 0.01$), if queried data items are in different clusters from the source node, a query message has to walk a long distance to be able to traverse the cluster border and locate the queried data items. In the case of a network with a large value of r (e.g., $r > 0.1$), query requests may escape out of the

Figure 2-1. Probability of random walks escaping out of the cluster decreases exponentially with the ratio of number of inter-cluster edges to the number of intra-cluster edges.

original cluster within a small number of hops, resulting in a long response time if the queried data is in the original cluster. These observations also are demonstrated by our simulations in Section 3.7. Consequently, random walks may suffer long response time regardless of the network having been well-clustered or not.

Interest-based shortcut: Interest-based shortcut [11] tries to avoid the blindness in random walks by favoring nodes sharing similar interests with the source, which can be regarded as a variation of markov random walks. Markov random walks may accelerate the query process to some extent in some cases. However, it causes new problems. Suppose nodes in an interest group have formed a cluster, and query messages can be artificially confined in this specific cluster. In the sense of nodes in the cluster share similar interests, any of them possibly maintains the queried data. The query procedure should shorten the covering time of the whole cluster instead of the hitting time of some specific nodes in it. However, due to the bias in selecting next hop in markov random walks, it tends to keep visiting some specific nodes, resulting in less distinct nodes being covered comparing to uniform random walks, as illustrated by Figure 2-2. Consequently, markov random walks work worse than uniform random walks if query messages can be confined in specific clusters.

Figure 2-2. Markov random walks discover less distinct number of nodes than uniform random walks do.

2.1.2 Motivation

Researchers [11, 56, 57] have found many peer-to-peer networks exhibit small-world topology, and most of queried data items are offered by the nodes, which share similar interest with the source node.

Intuitively, the nodes sharing similar interests with the source node should have higher priority to be searched than others. Practically, there are two challenges in designing such a query scheme. The first one is how to construct a small-world topology to cluster nodes sharing similar interests. By saying “similar interests”, we actually mean that two nodes are interested with a common set of data items. The number of common accessed data items can served as a metric to measure the interest similarity between two nodes. A clustering algorithm based on the metric can be easily designed to densely connect the nodes in the same interest group. Moreover, each node u can explicitly pick up a set of inter-cluster neighbors that have different interests from u, and a set of intra-cluster neighbors that share similar interests with u. Take Figure 2-3 as an example. The network consists of 5 clusters, and nodes in the same cluster fall into the same interest group. Note that there exists an interest group consisting of two clusters: 1, and 5.

Suppose the network has been well-clustered, and each node explicitly maintains a set of inter-cluster and intra-cluster neighbors. The second challenge is how to fast locate the clusters that share similar interests with the source node, and how to exhaustively search nodes in the found clusters, if the queried data items are in the source node's interest group. We introduce two types of queries: inter-cluster queries, and intra-cluster queries. The inter-cluster queries are for the purpose of discover the clusters that share similar interests with the source node, and they are issued by source node, carry the interest information, and only travel on inter-cluster neighbors. It can be expected that clusters sharing similar interests with the source node can be located quickly, because the number of cluster is much smaller comparing to the network size, and inter-cluster queries only travel among different clusters. The intra-cluster queries are spawned by inter-cluster queries when a cluster sharing similar interest with the source node is hit. An intra-cluster query thoroughly go through nodes in the found cluster, where it is spawned, by only traveling on intra-cluster neighbors. Note that inter-cluster queries and intra-cluster queries can be easily implemented if each node explicitly knows the types of its neighbors.

Occasionally, queried data may be out of the source node's interest group, and possibly maintained by a cluster(s) with different interests. This problem is addressed by blind search: inter-cluster messages randomly spawning intra-cluster messages when hitting clusters with different interests.

For example, in Figure 2-3, first inter-queries are initiated by a node in cluster 1, which travel among different clusters. By the interest information carried in the inter-queries, cluster 5 is found to share similar interests when it is hit, and an intra-cluster query is spawned, which then will exhaustively search the nodes in it. In addition, an intra-cluster query is spawned in cluster 2 by inter-cluster queries to support blind search.

Figure 2-3. A query scheme mixing inter-cluster queries and intra-cluster queries (the nodes in the grey clusters fall into the same interest group).

2.2 Constructing a Small-world Topology

2.2.1 Measuring the Interest Similarity between Two Nodes

Cluster is generally formed by connecting nodes with similar interests in a network. We start our discussion with the definition of interest similarity between two nodes in P2P networks.

If node u and node v share similar interests, then it is very likely that they have accessed same data items more or less previously. The size of the common subset of accessed data items can served as a metric to measure to what extent the interests of two nodes are similar.

Each node may offer hundreds of data items, and hence, there may exist a large number of data items even in a small network. As a result, only if u and v have visited a large number of data items respectively, they are able to show some degree of similarity. An alternative to evaluate the interest similarity is by the number of common accessed nodes, which may enable a clustering algorithm to converge faster than the former approach. The problem in this approach is that two nodes visiting a common node does not indicate they have similar interests, because a node may offer data items belonging to multiple interest groups. For instance, a user u may offer resources for two groups: a number of mp3 music files for one group, and a number of research literatures in P2P networks for the other group. It is possible that two nodes that have visited u may be interested in data items in different interest group. We have to address the discrepancy between the common set of accessed nodes and the common set of visited data items.

Suppose there are n nodes $N = \{1, 2, ..., n\}$ in the whole P2P network and a node i offers a number of data items to the community. It categorizes (maps) all of these data items into α_i different categories, denoted as $C^i = \{c^i_1, c^i_2, ..., c^i_{\alpha_i}\}$. Suppose a data item x in i is mapped to a category $c^i(x)$, where $c^i(x) \in C^i$. How to categorize the data items is determined by the node i independently. For instance, node i may classify music files as category 1, while another node may classify music files as category 2. On the other hand, node i may fall into multiple interest groups, denoted as $G^i = \{g^i_1, g^i_2, ..., g^i_{\beta_i}\}$.

For a node u, the access history with respect to each of its interest groups (e.g., g^u_1) can be specified by a set of data items x, denoted as $(i, c^i(x))$, where i represents the node offering the data item, and $c^i(x)$ is the category in C^i defined by i. If two nodes u and v share "similar interests" (e.g., $g^u_1 \approx g^v_2$), their histories for g^u_1 and g^v_2 tend to consist of a common set of $(i, c^i(x))$.

Note that in the above definitions, each node determines its interest groups and categories independently, indicating a node need not maintain any global information.

For easy explanation, we study a basic approach by assuming each user only falls into one interest group, and offers one category of data items. In this scenario, the access history can be represented by the accessed nodes alone. This approach can be easily extended to the multi-categories and multi-groups based on the definitions above.

One node u may access another node multiple times for different data items, and hence, the access history of node u can be represented by a vector $V^u = (v^u_1, .., v^u_n)$, [1] where v^u_x represents the number of times u has visited node x. To cancel out the number of queries a node has issued, the access vector V^u is normalized to the frequency vector

[1] The real size of the data structure maintaining V^u is much smaller than the network size n, and can be fixed to only record the nodes accessed most frequently by u

Figure 2-4. Interest similarity between nodes. The number of data items in 1, 2, and n are 50, 100, and 200 respectively. A) No common visited nodes (different interests). B) u and v have visited node 2 (100 data items) with 8 and 5 times respectively (a certain level of similar interests). C) u and v have visited node n (200 data items) with 8 and 5 times respectively (falls in-between)

F^u, in which the i-th element in F^u is denoted as f_i^u, computed by V^u as $f_i^u = \frac{v_i^u}{\sum_{j \in N} v_j^u}$, representing the frequency of the corresponding node i having been accessed.

Note that the value in F^u falls into the range $[0, 1]$. If u has never accessed node i, the corresponding element f_i^u is equal to 0. The summation of all elements is equal to 1.

Furthermore, if the number of data items in node i, denoted as d_i, is large, the chance that two nodes u and v have visited common data items in i may be small even if both of them have visited i multiple times. As an example, in B and C of Figure 2-4, u and v have visited one common node. But u and v in B have more chance of having visited common data items because the number of data items in node 2 is half of that in node n. To account for this issue, we introduce a weighted diagonal matrix W with (i, i)-th value $w_{i,i}$ equal to $\frac{1}{d_i}$. It represents the probability of both u and v visiting a common data items, if both of them visit i once.

Now we define the following metric to evaluate the interest similarity between two nodes (Take Figure 2-4 (B) as an example):

$$
\begin{aligned}
A_w^{u,v} &= F^{u^T} W F^v \\
&= \begin{pmatrix} .2 \\ .8 \\ 0 \\ . \\ . \\ 0 \end{pmatrix}^T
\begin{pmatrix} .02 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & . & 0 & 0 \\ 0 & 0 & 0 & 0 & . & 0 \\ 0 & 0 & 0 & 0 & 0 & .005 \end{pmatrix}
\begin{pmatrix} 0 \\ .5 \\ 0 \\ . \\ . \\ .5 \end{pmatrix} \\
&= 0.004
\end{aligned}
$$

Similarly, the interest similarity in Figure 2-4 (C) is 0.002, which means nodes in A show more interest similarity.

From the definition, we have

$$A_w^{u,v} = F^{u^T} W F^v = \sum_i f_i^u f_i^v \frac{1}{d_i} \tag{2–1}$$

If we view f_i^u and f_i^v as the probabilities of nodes u and v visiting node i, and $\frac{1}{d_i}$ as the probability of u and v visiting a common data items if both of them visit i, then the summation $A_w^{u,v}$ can be used to predict the probability that both u and v will visit a common data item in their future queries.

Note that if a node i has not been visited by both u and v, then $p_i^u = 0$ and/or $p_i^v = 0$, indicating that a node does not need to maintain any information of the nodes it has never visited. As we have discussed, the size of the vector V^u is fixed. Hence, both the storage for access history and the computation overhead for $A_w^{u,v}$ are constant.

Our definition is advantageous in manyfold. First, nodes u and v need not maintain any global information to compute $A_w^{u,v}$. Second, the frequency vector cancels out the effect of the number of queries that a node has issued, enabling a clustering algorithm based on this definition to converge fast with the average number of queries. Third, the definition prefers the nodes with good properties. For instance, if two nodes i, j maintaining same data items can be accessed by 1MB/s and 56Kb/s respectively. i obviously will be accessed more often, resulting in a larger value of p_i^u and p_i^v. Fourth, it reduces the impact of the possible discrepancy in the category definitions by nodes. For instance, if a category in node i is poorly defined, and consists of data items belonging to various interest groups, then the category will be seldom accessed comparing to its size, resulting in a small value of $f_i^u f_i^v \frac{1}{d_i}$.

2.2.2 Clustering Nodes with Similar Interests

Given the metric to evaluate the interest similarity between two nodes, we propose a light-weight clustering algorithm to connect nodes sharing similar interests.

In our strategy, each node i maintains a list L with limited size (e.g., 30) to record the nodes that possibly share same interests with itself. Each time a query message is processed, the similarity between the querying node itself and the node that owns the data items is computed. The newly obtained interest similarity, and the corresponding node's address are inserted into the list L. If the list is full, the stored neighbor with the lowest interest similarity is dropped.

By assuming that interests of nodes will not shift in a limited time frame, the nodes collected in L possibly fall into the same interest group as i, and will serve as candidates of its intra-cluster neighbors.

2.2.3 Bounding Clusters

Although a small-world topology can be formed along with queries by the above clustering algorithm, existing query schemes such as random walks can only benefit marginally from it as we discussed in Section 3.1.

To exploit the characteristics of the small world topology, our approach is to explicitly capture the clusters in the underlying topology by each node i maintaining a set of inter-cluster neighbors in other interest groups, and intra-cluster neighbors in its own interest group.

For inter-cluster neighbors, a node i can learn them easily. For example, i can issue a certain number of random walk messages only traveling on other nodes' inter-cluster neighbors, and choose the nodes hit by the messages as its inter-cluster neighbors. Note that the list L collects candidates of its intra-cluster neighbors, and should not overlap with the set of inter-cluster neighbors.

It is of the most importance to learn the intra-cluster neighbors, which can be selected from the nodes collected in the list L. The purpose of intra-cluster neighbors is to confine intra-cluster queries within a specific interest group. Two nodes falsely regarded as intra-cluster neighbors may create a dramatic impact because an intra-cluster query may traverse to another cluster with different interests. On the contrast, two nodes i and k that are falsely regarded as intra-cluster neighbors will only have limited impact, because i and k may be connected by other intermediate intra-cluster neighbors j. In addition, the chance of i and k falling into the same cluster tends to become larger along with query procedures if they are in the same interest group.

Based on this observation, we propose a labeling algorithm, which ensures that if a link (i, j) is labeled as an intra-cluster edge, then i and j are in the same interest group with high probability.

For a node i, we normalize the interest similarity of its neighbors j in L as follows.

$$p_{i,j} = \frac{A_w^{i,j}}{\sum_k A_w^{i,k}}$$

If a matrix P is organized such that its i, j-th element is $p_{i,j}$, then the rows in P sum to 1 as the matrix P is row stochastic. Intuitively, $p_{i,j}$ can be viewed as the transition probabilities for the markov random walk.

The transition probability $p_{i,j}$ can serve as a good metric to determine whether i and j are in the same interest group or not by introducing a threshold, denoted as T, as a lower bound. T can be set as a relatively larger value, because the false negative has limited impact as discussed. Suppose there are α neighbors in L that are possibly in the same interest group with i. T can be set as $\frac{1}{\alpha}$. Note that $p_{i,j}$ and T are computed by node i locally. The labeling algorithm does not involve any extra communication.

2.3 A Hybrid Query Scheme

2.3.1 Mixing Inter-Cluster Queries and Intra-Cluster Queries

By explicitly capturing the cluster structures in the underlying network, we can formally define following three types of query messages.

The first one is called **l-query message**, denoted as M_l, which is a special type of inter-cluster message only traveling on inter-cluster neighbors. The purpose of it is to quickly locate the clusters that may share similar interests with the source node and disperse intra-queries among different clusters. Messages in this type are issued by the source node, and walk among different clusters randomly. Moreover, if the queried data is in the source node's interest group, l-query messages should piggyback the source node's frequency vector, such that nodes hit by the messages can determine whether their clusters share similar interests with the source node.

The second one is called **s-query message**, denoted as M_s, which is a special type of intra-cluster message confining itself within a specific cluster by doing uniform random walks only on intra-cluster neighbors. s-query messages are only spawned/issued in the clusters that share similar interests with the source node. The purpose of it is to exhaustively search nodes that fall into the same interest group as the original node. Messages in this type are issued by the source node if the query is in the same interest group, and/or spawned by l-query messages, if clusters sharing similar interests with the source node are hit.

In order to reduce the number of duplicate s-query messages, each node having received the message should be able to estimate to what extent the cluster has been covered. If most of nodes have been visited, an s-query message has little chance to discover any new nodes by continuing to walk in the cluster, indicating the new received message should be discarded. Otherwise, the message should be forwarded.

Accurately estimating the covering time of a cluster is difficult and resource-consuming in a distributed system. Heuristically, if the message has been sequentially hitting a certain number, denoted as h, of nodes that have been visited by previous intra-cluster messages, most of nodes may have been covered, and hence, the message should be discarded. All messages in M_s need to maintain a counter to keep track on the sequential number of nodes having been visited by previous intra-cluster messages. The counter of each new spawned message in M_s is set to 0 or 1 depending on whether the starting node has been hit or not before. Note that the total (not sequential) number of nodes having been hit by previous intra-cluster queries is not a good metric for estimation, because it heavily depends on the cluster size.

The last one is called **b-query message**, denoted as M_b, which is also a special type of intra-cluster message similar to s-query message. The difference of b-query messages from s-query messages is that b-query messages may be spawned in clusters that have different interests from the source node. The purpose of it is to support blind search, because occasionally, the queried data may be out of the source node's interest group. The messages in this type are issued by the source node if the query is out of source's interest group, and/or possibly spawned by l-query messages, when clusters that have not been visited by intra-cluster query messages are hit. The chance that the queried data item in a cluster having different interests is very small. Once a b-query message hits a node that has been visited by intra-cluster messages before, the message is discarded to reduce the number of duplicated messages.

To control the communication overhead, the total number of concurrent query messages has to be limited. The source node needs to count the number of l-query, s-query and b-query messages, denoted as m_l, m_s and m_b respectively. Whenever an intra-cluster message, such as s-query message or b-query message, is spawned or discarded, the source node needs to be notified to update the corresponding counter. Only if the summation of m_l, m_s, and m_b is smaller than a certain number, denoted as m, a new b-query message can be spawned to support blind search. s-query messages can be spawned without restriction, and the total number of concurrent messages may be larger than m temporarily. Note that the counter m_l does not change in a query. In addition, all messages need to periodically check the status of the source node so that they can stop if the query has been successfully returned.

With these three types of messages, our query scheme is designed as follows.

Initialization: To initiate a query request, a node u issues a number m_l of l-query messages. If the queried data item falls in u's interest group, l-query messages carry the source's frequency vector, and a certain number m_s of s-query messages are issued to exhaustively search its own cluster. Otherwise, the message does not carry any interest information, and a b-query message is issued.

Receiving an l-query message: In the case of a node u receiving an l-query message, it calculates the interest similarity with the source node. If u shares similar interests (e.g., the similarity is larger than a small value), it spawns a new s-query message and update m_s maintained by the source node. Otherwise, a new b-query message is spawned, if the node has not been hit by other messages in M_s and M_b, and $m_l + m_s + m_b < m$. Finally, node u forwards the received message to a randomly selected inter-cluster neighbor.

Receiving an s-query message: In the case of a node u receiving an s-query message, if u has been hit by messages in M_s or M_b, it increases the counter in the message by 1. Otherwise, it resets the counter to 0. Next, if the counter is larger than the threshold h (e.g., 10), the node discards the message and notifies the source node to update the counter m_s. Otherwise, it forwards the message to a randomly selected intra-cluster neighbor.

Receiving a b-query message: In the case of a node u receiving a b-query message, if it has been hit by messages in M_s or M_b, the node discards the message and notifies the source node to update the counter m_b. Otherwise, it forwards the message to a randomly selected intra-cluster neighbor.

Our scheme can be considered to be stateful, in which if the same queries are reissued multiple times, less intra-cluster queries will be spawned in the clusters that have been well searched, and in contrast, more intra-cluster queries will be spawned in the less-searched clusters, resulting in stronger ability to discover more resources/replicas.

2.3.2 Reducing the Communication Overhead

By mixing inter-cluster and intra-cluster queries, it can be expected the system performance can be improved significantly. Because the access vector V^u of a node u can be fixed to a small size, the extra overhead will not increase largely (only l-query messages need to carry the frequency vector).

Moreover, l-query messages only travel among different clusters, and the number of clusters, especially in a well-clustered network, is much smaller comparing to the real network size. It can be expected that most of clusters can be covered by l-query messages within a small number of hops. l-query messages can remove the frequency vector from the payloads after a certain number of hops. In the meantime, a source node can specify one l-query message to keep the frequency vector for the case of that some clusters sharing similar interests with the source node have not been discovered after the specified number of hops.

2.4 Simulation

In this section, the performance of the proposed clustering algorithm and query scheme is studied by simulations. If not explicitly defined, the number of nodes is $10,000$, and the average group size is 150. Each node maintains 1,000 data items, the average number of queries issued by each node is 30, the threshold h is equal to 10, and the probability of a node incorrectly classifying its queries or data items is 0.1. Moreover, m and m_l are set to be 32 and 16 respectively.

Figure 2-5. Effect of average query number on the cluster size.

We compare our scheme to random walks, in which a source node issues 32 random walk messages in each query, correspondingly. We also have compared our scheme to flooding schemes. As expected, we observe the flooding schemes suffer from very large communication overhead.

In the figures, the legend “Uniform random walks (0)”/“Uniform random walks (1)” refers to the queried data items are out of/in the source node’s interest group in the uniform random walks query scheme, and similarly “Inter-intra (0)”/“Inter-intra (1)” refers to the queries are out of/in the source node’s interest group in the proposed scheme.

First, we study the effectiveness of the metric measuring interest similarity and the clustering algorithm. In Figure 2-5, it is observed that when the average query number is larger than 10, the algorithm reaches a stable state and almost all nodes in the same interest group form a single cluster. It indicates that our algorithm converge fast with the average number of queries, which is especially useful in P2P networks, where nodes tend to join/leave the system more frequently.

By Figure 2-6, it can be observed that the average number of nodes in a cluster is almost the same as group size, demonstrating that $A_w^{u;v}$ can effectively measure the nodes’ interest similarity.

Figure 2-6. Interest association is a good metric to estimate interest similarity

Figure 2-7. Percentage of returned query within a specific hop number.

Figure 2-8. Percentage of returned query within a specific message number.

Second we study the performance of our scheme with respect to query latency and communication overhead. In Figure 2-7, it is observed that if the queried data items fall into the original node's interest group, the number of hops needed for the majority of the queries is significantly reduced to about 20, while in the uniform random walks, it takes much longer time. Correspondingly, the number of messages is also much smaller in our scheme than that in random walks, as shown in Figure 2-8. The figures also show that if the queried data are out of the source node's interest group, the performance of our scheme is similar to uniform random walks. Note that a longer response time is acceptable since only a few queries will be out of source node's interest group in many P2P networks. In addition, these two figures also demonstrate that random walks for queries in the source node's interest group can only benefit marginally from the underlying clustered topology, that is, only a little larger percentage of them can be returned than those out of source node's interest group within the same number of hops). (messages).

We also have studied the performance of a network, in which each group consists of multiple different clusters, as shown in Figure 2-9 and Figure 2-10. The results show the similar trends, which keeps true with respect to all other metrics that will be studied later. Moreover, comparing Figure 2-7 with Figure 2-9, and Figure 2-8 with Figure 2-10, it can

Figure 2-9. Percentage of returned query within a specific hop number in a less clustered network.

Figure 2-10. Percentage of returned query within a specific message number in a less clustered network.

Figure 2-11. Number of distinct nodes discovered in the same group within a certain message range.

be observed that the performance of random walks in two different (well-clustered and poor-clustered) networks are similar, which further verifies our argument in Section 3.1.

As have been observed, when the queried data items are in the source nodes' interest group, our scheme works much better than random walks. The reason behind it is that our scheme can discover more distinct nodes in the source nodes' interest groups within the same number of messages or hops, as shown in Figure 2-11 and Figure 2-12. In the figures, it can be observed that within the first 1,000 messages or 30 hops, more than 120 nodes in the source node's interest group have been searched by query messages. Consequently, the majority of queried data falling into the node's interest group can be found with smaller overhead, and shorter latency. It also indicates our scheme has stronger ability to locate more replicas, since it can discover a much larger number of nodes sharing similar interests.

Occasionally, the queried data item may be maintained by nodes in other interest groups, or classified into wrong interest group by source node. In the former case, l-queries will not carry any interest information, but in the latter case, l-queries will carry wrong interest information. In both cases, the efficiency of our query scheme can be evaluated

Figure 2-12. Number of distinct nodes discovered in the same group within a specific hop number.

Figure 2-13. Percentage of messages discovering distinct nodes within a certain message range.

Figure 2-14. Total number of distinct nodes discovered within a specific hop number.

by the number of distinct nodes discovered by queries, including those out of the source node's interest group, within a certain number of messages and hops. Note that whether the queries are in or out of the original node's interest group makes no difference to random works. Figure 2-13 and Figure 2-14 show that in the first 1,000 messages, if the queries carrying interest information, less distinct nodes can be searched in our scheme. The reason is that s-query messages mistakenly exhaustively search the nodes in the clusters that share "similar" interests in the beginning, which has been demonstrated by our previous simulations. Consequently, the number of b-query messages is limited. Along with the increment of the number of messages/hops, our scheme works similar to the uniform random walks. It is because after most of nodes sharing similar interests are covered, more b-query messages will be spawned to search clusters with different interests, which are able to discover more distinct nodes. In addition, by the figures, if the queries carry no interest information, our scheme works similar to uniform random walks.

CHAPTER 3
MARCH: A DISTRIBUTED INCENTIVE SCHEME IN OVERLAY NETWORKS

3.1 Motivation

3.1.1 Limitation of Prior Work

Any node in a peer-to-peer network is both a service provider and a service consumer. It contributes to the system by working as a provider, and benefits from the system as a consumer. A transaction is the process of a provider offering a service to a consumer, such as supplying a video file. The purpose of an incentive scheme is to encourage the nodes to take the role of providers. Neither reputation systems [12, 13, 14, 15, 16, 17] nor virtual currency systems [18, 19] can effectively prevent malicious nodes, especially those in collusion, from manipulating their history information by using false service reports. Specifically, the existing schemes have the following problems.

Reputation inflation: In the reputation schemes, malicious nodes can work together to inflate each other's reputation or to defame innocent nodes, by which colluding nodes protect themselves from the complaints by innocent nodes as these complaints may be treated as noise by the systems.

Money depletion: In the virtual currency schemes, malicious nodes may launch attacks to deplete other nodes' money and paralyze the whole system. Without authentic reputation information, innocent nodes are not able to proactively select benign partners and avoid malicious ones.

Frequent complainer: In many incentive schemes, nodes will be punished if they complain frequently, which prevents malicious nodes from constantly defaming others at no cost. It also discourages innocent nodes from reporting frequent malicious acts because otherwise they would become frequent complainers.

Punishment scale: In most existing schemes, the scale of punishment is related to the service history of the transaction participants. Consequently, an innocent node may be subject to negative discrimination attacks [12] launched by nodes with excellent history.

3.1.2 Motivation

Punishing malicious nodes and limiting the damage caused by a colluding group are indispensable requirements of an incentive scheme that is able to deter bad behavior. There are two major kinds of bad behavior. First, a provider may deceive a consumer by providing less-than-promised service. Second, a consumer may defame a provider by falsely claiming the service is poor.

Consider how these problems are dealt with in real life. Before a transaction happens, the provider would want to know if the consumer has enough money to pay for the service, and the consumer would want to know the reputation of the provider. With such information, they can control the risk and decide whether to carry out the transaction or not. After the transaction, if the provider deceives, it will be sued by the consumer. Consequently, the malicious provider will build up bad reputation, which prevents it from deceiving more consumers. Now consider a consumer intentionally defames a provider. It does so only after it can show the evidence of a transaction, which requires it to pay money first. Consequently, defaming comes with a cost. The ability of the malicious consumer to defame others is limited by the amount of money it has.

Inspired by the observation above, we propose a new incentive scheme: **MARCH**, which is a combination of Money And Reputation sCHemes.

The basic idea behind the scheme is simple: each node is associated with two parameters: money and reputation. The providers earn money (and also reputation) by serving others. The consumers pay money for the service. If a consumer does not think the received service worth the money it has paid, it reports to an authority, specifying the amount of money it believes it has overpaid. If the authority can determine who is lying, the liar is punished. Otherwise, the authority freezes the money claimed to have been overpaid. The money will not be available to the provider and will not be returned to the consumer either, which eliminates any reason for the consumer to lie. If the provider is guilty, the consumer has the revenge and the provider's reputation suffers. If the provider is innocent, the consumer does it at a cost because after all it has paid the price of the transaction. In addition, the falsely-penalized provider will not serve it any more. The technical challenges are (1) establishing a distributed authority for managing the money and reputation, (2) designing the protocol of transaction that ensures authentic exchange of money/reputation information and allows the unsatisfied consumers to sue the providers, (3) analyzing the properties of such a system, and (4) evaluating the system.

3.2 System Model

The nodes in a P2P network fall in three categories: honest, selfish, and malicious. Honest nodes follow the protocol exactly, and they both provide and receive services. Selfish nodes will break the protocol only if they can benefit. Malicious nodes are willing to compromise the system by breaking the protocol even when they benefit nothing and may be punished. Selfish/malicious nodes may form colluding groups. There may exist a significant number of selfish nodes, but the malicious nodes are likely to account for a relatively small percentage of the whole network. At a certain time, all self/malicious nodes that break the protocol are called dishonest nodes. A node is said to be rejected from the system if it has too little money and too poor reputation such that no honest providers/consumers will perform transaction with it. We study the incentive scheme in the context of DHT-based P2P networks [1, 2, 4]. We assume the routing protocol is robust, ensuring the reliable delivery of messages in the network [58]. We also assume the networks have the following properties.

• Random, non-selectable identifier: A node can not select its identifier, which should be arbitrarily assigned by the system. This requirement is essential to defending the Sybil attack [59]. One common approach is to hash a node's IP address to derive a random identifier for the node [1].

• Public/private key pair: Each node A in the network has a public/private key pair, denoted as P_A and S_A respectively. A trusted third party is needed to issue public-key certificates. The trusted third party is used off-line once per node for certificate issuance, and it is not involved in any transaction.

3.3 Authority Infrastructure

3.3.1 Delegation

Who will keep track of the money/reputation information in a P2P network? In the absence of a central authority for this task, we design a distributed authority infrastructure. Each node A is assigned a **delegation**, denoted as D_A, which consists of k nodes picked pseudo-randomly. For example, we can apply k hash functions, i.e., $\{h_1, h_2, ..., h_k\}$, on the identifier of node A to derive the identifiers of nodes in D_A. If a derived identifier does not belong to any node currently in the network, the "closest" node is selected. For example, in [1], it will be the node clockwise after the derived identifier on the ring. The j-th element in D_A is denoted as $D_A(j)$.

D_A keeps track of A's money/reputation. Any anomaly in the information stored at the delegation members may indicate an attempt to forge data. The information is legitimate only if the majority of the delegation members agree on it. As long as the majority of the delegation members are honest, the information about node A cannot be forged. Such a delegation is said to be trustworthy. On the other hand, if at least half of the members are dishonest, then the delegation is untrustworthy.

The delegation members are appointed pseudo-randomly by the system. A node cannot select its delegation members, but can easily determine who are the members in its or any other node's delegation. To compromise a delegation, the malicious/selfish nodes from a colluding group must constitute the majority of the delegation. Unless the colluding group is very large, the probability for this to happen is small because the identifiers of the colluding nodes are randomly assigned by the system and the identifiers of the delegation are also randomly assigned. Let m be the size of a colluding group and n be the total number of nodes in the system. The probability for t out of k nodes to be in the colluding group is

$$P(t, k, \frac{m}{n}) = \begin{pmatrix} k \\ t \end{pmatrix} (\frac{m}{n})^t (1 - \frac{m}{n})^{k-t}$$

Figure 3-1. Trustworthy probability for a delegation and 5-pair delegation set with $m^* = 3,000$ are 99.975% and 99.815% respectively. Even if a delegation/k-pair delegation set is not trustworthy, it may not be compromised because very unlikely a single colluding group can control the majority of them.

where $P(t, k, \frac{m}{n})$ denotes the probability of t successes in k trials in a Binomial distribution with the probability of success in any trial being $\frac{m}{n}$. Let m^* be the total number of distinct nodes in all colluding groups, also including all malicious nodes. The probability of a delegation being trustworthy is at least $\sum_{t=0}^{\lfloor \frac{k}{2} \rfloor} P(t, k, \frac{m^*}{n})$. We plot the trustworthy probability with respect to m^* when $k = 5$ in Figure 3-1 (the upper curve). In order to control the overhead, we shall keep the value of k small.

3.3.2 k-pair Trustworthy Set

A transaction involves two delegations, one for the provider and the other for the consumer. They have to cooperate in maintaining the money and reputation information, and avoiding any fraud. To facilitate the cooperation, we introduce a new structure, called **k-pair delegation set**, consisting of k pairs of delegation members. Suppose node A is the provider and node B is the consumer. The ith pair is $(D_A(i),\, D_B(i))$, $\forall i \in [1..k]$, and the whole set is

$$\{(D_A(1), D_B(1)), (D_A(2), D_B(2)), ..., (D_A(k), D_B(k))\}$$

If both $D_A(i)$ and $D_B(i)$ are honest, the pair $(D_A(i), D_B(i))$ is trustworthy. If the majority of the k pairs are trustworthy, the whole set is trustworthy. It can be easily verified that the probability for the whole set to be trustworthy is

$$\sum_{t=0}^{\lfloor \frac{k}{2} \rfloor} P(t, k, 2\frac{m^*}{n} - (\frac{m^*}{n})^2)$$

We plot the trustworthy probability for the whole set with respect to m^* in Figure 3-1 (the lower curve).

3.4 MARCH: A Distributed Incentive Scheme

3.4.1 Money and Reputation

With the distributed authority designed in the previous section, the following information about a node A is maintained by a delegation of k nodes.

Total money (TM_A): It is the total amount of money paid by others to node A minus the total amount of money paid to others by A in all previous transactions. The universal refilled money (Section 3.6.2) will also be added to this variable.

Overpaid money (OM_A): It is the total amount of money overpaid by consumers. A consumer pays money to node A before a service. If the service contract is not fulfilled by the transaction, the consumer may file a complaint, specifying the amount of money that it has overpaid. This amount cannot be greater than what the consumer has paid.

When a new node joins the network, its total money and overpaid money are initialized to zero. From TM_A and OM_A, we define the following two quantities.

Available money (m_A): It is the amount of money that node A can use to buy services from others.

$$m_A = TM_A - OM_A \tag{3–1}$$

Reputation (r_A): It evaluates the quality of service (with respect to the service contracts) that node A has provided.

$$r_A = \begin{cases} \frac{TM_A - OM_A}{TM_A} & \text{if } TM_A \neq 0 \\ 1 & \text{if } TM_A = 0 \end{cases} \tag{3–2}$$

For example, if $TM_A = 500$ and $OM_A = 10$, then A's available money is 490, i.e., m_A=490, and its reputation is 0.98, i.e., $r_A = 0.98$.

To track every node's available money and reputation, we propose a set of protocols. Consider a transaction, in which Alice (A) is the provider and Bob (B) is the consumer. The transaction consists of five sequential phases.

- Phase 1: Contract Negotiation. Alice and Bob negotiate a service contract.
- Phase 2: Contract Verification. Through the help of their delegations, Alice and Bob verify the authenticity of the information claimed in the contract.
- Phase 3: Money Transfer. The amount of money specified in the contract is transferred from Bob's account in D_B to Alice's account in D_A.
- Phase 4: Contract Execution. Alice offers the service to Bob based on the contract specification.
- Phase 5: Prosecution. After the service, Bob provides feedback reflecting the quality of service offered by Alice.

3.4.2 Phase 1: Contract Negotiation

Suppose Bob has received a list of providers through the lookup routine of the P2P network. Each provider specifies its reputation and its price for the service. Bob wants to minimize his risk when deciding which service provider he is going to use.

Let L_A be the price specified by Alice and G_B be the fair price estimated by Bob himself. According to the definition, r_A can roughly be used as a lower bound on the probability of Alice being honest. Intuitively, the probability for Bob to receive the service is at least r_A, and the probability for Bob to waste its money L_A is at most $(1 - r_A)$. We define the benefit for Bob to have a transaction with Alice as $G_B \times r_A - L_A \times (1 - r_A)$. We further normalize it as

$$R = r_A - \frac{L_A}{G_B}(1 - r_A) \tag{3–3}$$

To avoid excessive risk, Bob takes Alice as a potential provider if R is greater than a threshold value T. The use of threshold helps the system reject dishonest providers with poor reputation. Among all potential providers, Bob picks the one with the highest normalized benefit.

Both the value of L and the value of r_A are given by the provider A. If L is set too high, R will be small and the provider runs the risk of not being picked by Bob. Providers with poor reputation can improve their R values by setting their prices low. In this way, they can recover their reputation by selling services at lower prices. If Alice lies about its r_A, she will be caught in the next phase and be punished.

Now suppose Bob chooses Alice as the best service provider. They have to negotiate a service contract, denoted as c, in the following format.

$$< A, B, S, Q, L, Seq_A, Seq_B, r_A, m_B >$$

where A, B, S, Q, and L specify the provider, the consumer, the service type, the service quality, and the service price respectively. Seq_A and Seq_B are the contract sequence numbers of Alice and Bob, respectively. After the transaction, Alice and Bob each increase their sequence numbers by 1. The values of r_A and m_B in the contract will be verified by the delegations in the next phase.

As an example, if $S = Storage$, $Q = 200G$, and $L = 5$, the contract means that Alice offers storage with size 200G to Bob, and as return, the amount of money Bob must pay is 5.

3.4.3 Phase 2: Contract Verification

After negotiating a contract, Alice and Bob should exchange an authenticatable contract proof, so that Alice is able to activate the money transfer procedure and Bob is granted the prosecution rights. In addition, the information in the contract such as r_A and m_B should be verified by the delegations of Alice and Bob.

We use the notation $[x]_y$ for the digital signature signed on message x with key y and $\{x\}_y$ for the cipher text of message x encrypted with key y. After Phase two, if the contract is verified by the delegations, Alice should have the following contract proof

$$c_A = [c]_{S_B}$$

c_A should not be produced by Bob, who may lie about m_B. Instead, Alice must receive c_A from Bob's delegation after the members confirm the value of m_B. Bob has k delegation members. Each of them will produce a "piece" of c_A and send it to Alice, who will combine the "pieces" into a valid contract proof. Similarly, Bob must receive the following contract proof from Alice's delegation

$$c_B = [c]_{S_A}$$

The contract proofs will be used by Alice for money transfer and by Bob for prosecution.

It is important to ensure that either both Alice and Bob, or none of them, receive the contract proofs. Otherwise, dishonest nodes may take advantage of it. It can be shown that ensuring both or neither one receives her/his contract proof is impossible without using a third party (the delegation of Alice or Bob in this case).

Key Sharing Protocol

A k-member delegation is not a centralized third party. One possible approach for producing a contract proof by a delegation is to use threshold cryptography [60]. A (k, t) threshold cryptography scheme allows k members to share the responsibility of performing a cryptographic operation, so that any subgroup of t members can perform this operation successfully, whereas any subgroup of less than t members can not. For digital signature, k shares of the private key are distributed to the k members. Each member generates a partial signature by using its share of the key. After a combiner receives at least t partial signatures, it is able to compute the signature, which is verifiable by the public key. An important property is that less than t compromised members cannot produce a verifiable signature on a false message.

In our case, the problem is to produce c_B (or c_A) by the k-member delegation of Alice (or Bob). We employ a $(k, \lfloor \frac{k}{2} \rfloor + 1)$ threshold cryptography scheme to produce the contract proof. Alice distributes shares $S_A(i)$ of her private key S_A to her delegation members $D_A(i)$, which will produce partial signatures $[c]_{S_A(i)}$ and forward them to Bob for combination. As long as the delegation of Alice is trustworthy, Bob will receive enough correct partial signatures to compute a verifiable contract proof, while the false partial signatures generated by the compromised delegation members will not yield any verifiable proof.

When applying threshold cryptography, we have to defend against dishonest nodes, which may intentionally distribute incorrect secret shares. The incorrect partial signatures cannot yield a valid signatures. We propose a protocol for distributing the key shares. Take Alice as an example. The protocol guarantees that either all delegation members receive the correct shares, or they all detect that Alice is dishonest.

Step 1: Alice sends a key share $S_A(i)$ to each delegation member $D_A(i)$, encrypted by the member's public key $P_{D_A(i)}$. The messages are shown below.

MSG1 $Alice \rightarrow D_A(i)$: $[\{S_A(i)\}_{P_{D_A(i)}}]_{S_A}, \forall D_A(i) \in D_A$

Step 2: After all members receive their key shares, they negotiate a common random number s (possibly by multi-party Diffie-Hellman exchange with authentication). Each member sends the number s as a challenge to Alice, signed by the member's private key and then encrypted by Alice's public key.

MSG2 $D_A(i) \rightarrow Alice$: $\{[s]_{S_{D_A(i)}}\}_{P_A}, \quad \forall D_A(i) \in D_A$

Step 3: Alice signs s with $S_A(i)$ and then with S_A before sending it back to $D_A(i)$.

MSG3 $Alice \rightarrow D_A(i)$: $[[s]_{S_A(i)}]_{S_A}, \quad \forall D_A(i) \in D_A$

Step 4: After authentication, if the received $[s]_{S_A(i)}$ value matches the locally computed one, $D_A(i)$ forwards the message to all other members in D_A. [1]

MSG4 $\quad D_A(i) \rightarrow D_A(j)$: $\quad [[s]_{S_A(i)}]_{S_A}, \quad \forall D_A(j) \in D_A$

Otherwise, $D_A(i)$ files a certified complaint to other members.

MSG5 $\quad D_A(i) \rightarrow D_A(j)$: $\quad$ [“$S_A(i)$ is invalid”]$_{S_{D_A(i)}}$,

$\forall D_A(j) \in D_A$

Step 5: $D_A(i)$ needs to collect $[s]_{S_A(j)}$, $\forall D_A(j) \in D_A$, which are the partial signatures on s. If it receives $MSG4$ $[[s]_{S_A(j)}]_{S_A}$ from $D_A(j)$, the value of $[s]_{S_A(j)}$ is in the message. If it receives $MSG5$ from $D_A(j)$, there are two possibilities: either Alice or $D_A(j)$ is dishonest. To resolve this situation, $D_A(i)$ forwards the certified complaint to Alice. If Alice challenges the complaint, she must disclose the correct value of $S_A(j)$ to $D_A(i)$ in the following message (then $D_A(j)$ can learn $S_A(j)$ from $D_A(i)$).

MSG6 $\quad Alice \rightarrow D_A(i)$: $\quad [\{S_A(j)\}_{P_{D_A(i)}}]_{S_A}$

Learning $S_A(j)$ from this message, $D_A(i)$ can compute $[s]_{S_A(j)}$. After $D_A(i)$ has all k partial signatures on s, it can determine that Alice is honest if any $(\lfloor \frac{k}{2} \rfloor + 1)$ partial signatures produce the same signature $[s]_{S_A}$, which can be verified by Alice's public key. Otherwise, Alice must be dishonest.

Since the value of k is typically set small (e.g. 5) and the key distribution is performed once per node, the overhead of the above protocol is not significant.

Theorem 3.1. *The key sharing protocol ensures that all delegation members will either obtain the correct shares of Alice's private key or detect Alice's fraud.*

Proof. First of all, any node cannot deny the messages it has sent to others or falsely declare it has received some messages from others, because all messages in the protocol are signed by the corresponding nodes with their private keys.

[1] Note that $D_A(i)$ knows s and learns $S_A(i)$ from $MSG1$.

Consider the first case that Alice is honest. All delegation members can obtain the correct shares in Step 1. In the meantime, only if a complaint is signed by a delegation member, Alice will disclose the corresponding share to challenge the complaint. If Alice is honest, only dishonest members may issue certified complaints. The total number of distinct shares exposed by Alice is no larger than the number of dishonest members.

Next consider the second case that Alice is dishonest. Alice may try to deceive the delegation in two possible ways. One way is that Alice does not send shares to some delegation members $D_A(i)$, which can be easily detected by $D_A(i)$ when it receives $MSG3$ from Alice or $MSG4$ from other delegation members. Subsequently, $D_A(i)$ will file a certified complaint ($MSG6$). If Alice discloses the correct share ($MSG7$) to challenge the complaint, $D_A(i)$ can obtain its share from other members; otherwise, honest members are certain that Alice is dishonest, and will punish her.

The other possible way for Alice to deceive is to distributes incorrect shares to some members $D_A(i)$ in $MSG1$. There are three possible outcomes when $MSG3$ is processed by $D_A(i)$. (1) The partial signature in $MSG3$ matches the locally computed one. Subsequently, $MSG3$ is forwarded to all other members by $D_A(i)$. Then all honest members can detect Alice's fraud in Step 5 because, in addition to $[s]_{S_A(i)}$, there are $\frac{\lfloor k \rfloor}{2}$ other partial signatures that cannot be used to compute the signature $[s]_{S_A}$. (2) The partial signature in $MSG3$ does not match the locally computed one. $D_A(i)$ will detect Alice's fraud in Step 4 because of the inconsistency between $MSG1$ and $MSG3$. It will forward two inconsistent messages from Alice to all other members in $MSG5$. Consequently, all members learn the inconsistency and punish Alice. (3) Alice does not send $MSG3$ to $D_A(i)$ at all. This can be handled in a way similar to the previous case that $D_A(i)$ does not receive $MSG1$ from Alice. □

Contract Verification Protocol

Both Alice and Bob must register the contract with their delegations so that the money transfer and the optional prosecution can be performed through the delegations at

Figure 3-2. A) Protocols for contract verification and exchange. B) money transfer. C)Prosecution.

later times. The delegations must verify the information claimed by Alice and Bob in the contract and generate the contract proofs that Alice and Bob need in order to continue their transaction. We design a contract verification protocol to implement the above requirements. The protocol consists of four steps, illuminated in Figure 3-2 (A), and the number of messages is $O(k)$ for normal cases.

A procedure call is denoted as $x.\mathbf{y}(z)$, which means to invoke procedure y at node x with parameter(s) z. If x is a remote node, a signed message carrying z must be sent to x first.

Step 1: Alice sends the contract c and a digital signature c' to the delegation D_A for validation. c' may be a signature of the contract concatenating the identifier of the receiver, i.e., $c' = [c|D_A(i)]_{S_A}$. Bob does the same thing.

```
Alice.**SendContract**(Contract $c$, Signature $c'$)
1.  **for** $i = 1$ to $k$ **do**
2.      $D_A(i)$.**ComputePartialProof**($c$, $c'$)
```

Step 2: Then the delegation member $D_A(i)$ verifies the reputation claimed by Alice in the contract (denoted as $c.r_A$) and computes a partial signature (denoted as ps_i) on the contract with its key share established by the previous protocol.

```
$D_A(i)$.**ComputePartialProof**(Contract $c$, Signature $c'$)
1. **if** $r_A \geq c.r_A$ **then**
2.     $ContractList.add(c, c')$
3.     $ps_i = [c]_{S_A(i)}$
4.     $D_B(i)$.**DeliverPartialProof**$(c, ps_i)$
5. **else** punish(A)
```

Line 1 verifies whether $c.r_A$ is over-claimed or not. Line 2 saves the contract for later use in Step 3. The signature c' will be used in a procedure called detect(). Line 3 produces a partial signature on the contract by using $S_A(i)$. Line 4 sends the partial signature to the ith member of D_B. If $c.r_A$ is over-claimed, Alice will be punished at Line 5.

The delegation members in D_B execute a similar procedure except that the condition in Line 1 should be $m_B \geq c.m_B$.

Step 3: When $D_B(i)$ receives the contract c and the partial signature sp_i from $D_A(i)$, it executes the following procedure.

```
$D_B(i)$.**DeliverPartialProof**(Contract $c$, PartialSignature $ps_i$)
1. wait for a timeout period
2. **if** $c$ is found in $ContractList$ **then**
3.     $Bob$.**ProcessPartialProof**$(ps_i)$
4. **else**
5.     detect()
```

Line 1 waits for a timeout period to ensure that the contract from Bob has arrived. Line 2 checks if the received contract c is also in the local $ContractList$. If that is true, Bob has announced the exact same contract as Alice does, and $D_B(i)$ forwards the partial signature ps_i to Bob. Otherwise, there are two possibilities: (1) Bob does not have the same contract as Alice has, or (2) Bob does have the same contract but its contract has not arrived at $D_B(i)$ yet. To distinguish these two cases, Line 4 waits for a timeout period before checking again if c is now in $ContractList$. If not, $D_B(i)$ believes that Alice and

Bob do not have the same contract, and the detect() procedure is executed to detect the special case of a malicious Bob forging the contract. Once Bob is detected to be dishonest, the delegation can regard him as a liar, and omit the detection procedure in the future suspicious transactions, indicating the detection procedure needs to be invoked only once for each dishonest node.

In the following, we will present the design details of the detect() procedure, and then provide the correctness proof. The delegation member $D_B(i)$, which has received two different contracts from Bob and $D_A(i)$, must handle two possible cases. One case is that $D_B(i)$ has received a contract with the sequence number $c.Seq_B$ from Bob, and the other is that $D_B(i)$ has never received such a contract from Bob. In the former case, $D_B(i)$ stops the verification procedure immediately. Then it tries to detect whether Bob is lying or not by sending Bob's signature c' to all other delegation members in D_B. If a member finds that c' is different from Bob's signature that it receives directly from Bob, it sends its version of Bob's signature to all other members. Otherwise, the member discards the signature from $D_B(i)$. In the latter case, $D_B(i)$ sends a special request, denoted REQ, with the sequence number $c.Seq_B$ to all other members in D_B. If a member has already received the contract from Bob with the specified sequence number, it sends the corresponding signature c' to all other members after receiving REQ. Otherwise, the member discards the request REQ. In both cases, any member that has received two different versions of Bob's signature c' punishes Bob and refuses to participate in the rest of transaction for the contract. In addition, for the latter case, $D_B(i)$ refuses to proceed with the verification if no replies are received from other members, or punishes Bob but still continues the verification procedure using the the contract retrieved from other members if all of the signatures received are the same.

We show that, having received conflicted contracts, if a delegation member simply stops the verification procedure without invoking the detect() routine, Bob will be able to break the protocol. Suppose k is equal to 3, $D_B(1)$ is a friend of Bob, and both $D_B(2)$

and $D_B(3)$ are honest. Bob can break the protocol by sending $D_B(2)$ a correct contact while sending $D_B(3)$ a forged contract (for instance, with lower price $c.L$) or not sending the contract to $D_B(3)$ at all. Because $D_B(1)$ is Bob's friend, it may forward the partial signature $ps_A(1)$ from $D_A(1)$ to Bob, but not send the partial signature $ps_B(1)$ to $D_A(1)$. Then, Bob can collect two partial signatures $ps_A(1)$ and $ps_A(2)$ because $D_B(2)$ cannot detect Bob's fraud and will forward the partial signature $ps_A(2)$ to Bob, while Alice can only get one signature $ps_B(2)$ from $D_A(2)$. Bob can compute the contract proof signed by Alice, while Alice cannot compute the proof signed by Bob. In our protocol, this problem is addressed by $D_B(3)$ invoking the detect() routine after receiving the contract from $D_A(3)$. The detect() routine guarantees that either $D_B(2)$ detects Bob's fraud or $D_B(1)$ forwards the partial signature of the contract retrieved from $D_B(2)$.

Step 4: After Alice (Bob) receives t or more correct partial signatures, she (he) can compute the contract proof c_A (c_B), which can be verified by using Bob's (Alice's) public key.

Theorem 3.2. *If k-pair delegation set of Alice and Bob is trustworthy, the contract verification protocol ensures that both Alice and Bob will receive the correct proofs, or neither one can receive a valid contract proof and the transaction is aborted.*

Proof. First, we prove that neither Alice nor Bob can deceive the authority. This is a symmetric protocol, so without losing generality we only consider Bob. Below we analyze the four possible ways that Bob may use to deceive the authority.

1. Bob over-claims its available money m_B. In this case, all honest members in D_B can detect Bob's fraud in Step 1, and punishes Bob. In the meantime, these members will neither forward the partial signature $[c]_{S_B(i)}$ to $D_A(i)$ nor deliver $[c]_{S_A(i)}$ to Bob, and consequently the transaction will be aborted.
2. Bob modifies the contract specification, for example, by lowering the transaction price $c.L$ in order to pay less for the transaction. He sends the same modified

contract to D_B. In Step 3, all members in D_B learn that the contract presented by Bob is different from that presented by Alice, and they will invoke the detect() procedure. In this case, all honest delegation members will stop contract verification immediately, but they will not punish Bob, because there is only one contract signature from Bob and either Alice or Bob may be lying.

3. Bob sends different modified contracts to the delegation members. Multiple delegation members will detect that the contracts from Bob and Alice are different, and they will invoke the detect() routine. At the end of the detection procedure, all members will learn that there are different contract signatures coming from Bob. Consequently, they will all punish Bob and abort the transaction.
4. Bob does not send the contracts to some (or all) delegation members. In this case, a member $D_B(i)$ that does not received the contract from Bob will send the request REQ to all other members. If no other member receives the contract from Bob, $D_B(i)$ will receive no reply back. It will refuse to continue the verification process, but will not punish Bob because either Alice or Bob can be lying. Similarly, all other members will also stop the verification. Now if some other members have received the contracts from Bob, $D_B(i)$ will receive Bob's signatures in the replies from them, and it will continue the verification process using the contracts retrieved from other members. No one stops the verification process.

In summary, we can see that all members in the trustworthy set will take the same action (continuing or stoping the contract verification process) in all four possible cases.

Next, we prove that dishonest members cannot deceive honest members in the trustworthy sets to stop the verification process if both Alice and Bob are honest. As we have discussed above, only in two cases, an honest delegation member, $D_A(i)$ or $D_B(i)$, will stop the contract verification. One case is that the contract signatures (both Alice's and Bob's) received by the member in the detect() routine are not identical. The other case is that the member receives different contracts from Alice and Bob in Step 3. The former case happens only when Alice/Bob sign and distribute different contracts to delegation members dishonestly, while the latter case happens when both $D_A(i)$ and $D_B(i)$ are untrustworthy or when either Alice or Bob is dishonest. If both Alice and Bob are honest, dishonest members cannot interrupt the verification process at the members in the trustworthy sets.

By Theorem 3.1 and the discussions above, if Alice and Bob are honest, both of them are able to collect no less than $\frac{\lfloor k \rfloor + 1}{2}$ correct partial signatures, and compute the valid contract proofs. Otherwise, if either Alice or Bob is dishonest, the transaction is aborted. $\square$

3.4.4 Phases 3 and 4: Money Transfer and Contract Execution

Before providing the service, Alice requests its delegation to transfer money, which is illuminated in Figure 3-2 (B). Upon receiving a money transfer request from Alice, the delegation member $D_A(i)$ invokes the following procedure.

```
$D_A(i)$.**TransferMoneyProvider**(Contracts $c$, ContractProof $c_A$)
1.  **if** valid($c$, $c_A$) and $D_B(i)$.**TransferMoneyConsumer**($c, c_A$)
2.      $TM_A = TM_A + c.L$
3.  **else** verify()
```

In Line 1, both $D_A(i)$ and $D_B(i)$ need to validate the contract by using Bob's public key, which can be queried from Bob if it is not locally available. After validation, $D_A(i)$ increases Alice's earned money in Line 2.

Note that $D_B(i)$ may be malicious. If $D_A(i)$ cannot get a positive answer from $D_B(i)$, it must verify the validity of the contract further (Line 3), which can be designed as follows. $D_A(i)$ asks other members in D_A. If the majority of D_A have received a positive answer from D_B, the contract is considered to be valid ($D_B(i)$ is malicious). Otherwise, the contract is considered to be invalid and Alice is punished.

When $D_B(i)$ receives a money transfer request from $D_A(i)$, it performs the following operations.

$D_B(i)$.**TransferMoneyConsumer**(Contract c, ContractProof c_A)

```
1. if $valid(c, c_A)$ then
2.     if $m_B > c.L$ then
3.         $TM_B = TM_B - c.L$
4.         return true;
5.     else
6.         punish(B)
7.         return false;
8. else return false;
```

First, if the contract is valid (Line 1) and Bob has enough money to pay the service (Line 2), then Bob's spent money is increased and a positive answer is returned to $D_A(i)$ (Line 3 and 4). Second, it is possible that the contract is valid but Bob does not have enough money. This happens when Alice and Bob are colluding nodes and Alice gets the contract proof c_A directly from Bob instead through her delegation. In such a case, Bob is punished and a negative answer is returned (Line 6 and 7). Third, if the contract is invalid, a negative answer is returned (Line 8).

$D_A(i)$ and $D_B(i)$, $\forall i \in [1..k]$, perform money transfer at most once for each contract. They keep track of the sequence numbers (Seq_A and Seq_B) of the last contract for which the money has been transferred. All new contracts have larger sequence numbers.

3.4.5 Phase 5: Prosecution

After Bob receives the service from Alice, if the quality of service specified in the contract is not met, Bob may issue a prosecution request to Alice's delegation, as illustrated in Figure 3-2 (C). The request specifies the amount of money f that Bob thinks he has overpaid.

Upon receiving a prosecution request from Bob, if D_A cannot evaluate the service quality, it punishes both Alice and Bob by freezing the money overpaid by Bob. The procedure is given as follows.

```
$D_A(i)$.**Prosecution**(Contract $c$, ContractProof $c_B$, Overpaid $f$)
1.  if $valid(c, c_B)$ and $f \le c.L$ then
2.      $OM_A = OM_A + f$
3.      notify($A$)
```

First $D_A(i)$ validates the prosecution request by checking if the contract proof is authentic (Line 1). If the contract is valid, it increases Alice's overpaid money by f (Line 2). Finally, it notifies Alice so that Alice is able to determine whether to sell service to Bob in the future.

3.5 System Properties and Defense against Various Attacks

3.5.1 System Properties

We study the properties of MARCH, which solves or alleviates the problems in the previous approaches.

First, according to the money transfer procedures in Section 3.4.4, transactions among members in the same colluding group cannot increase the total amount of available money of the group. We have the following property, which indicates that the malicious nodes cannot benefit by cooperation.

Property 1. *Regardless of its size, a colluding group cannot increase its members' money or reputation by cooperation without decreasing other members' money and/or reputation.*

Second, unlike some other schemes [12, 13, 15, 16], MARCH does not maintain the history of any consumer's complaints, and does not punish frequent complainers. We have the following property.

Property 2. *If a consumer is deceived, it is not restricted by the system in any way from seeking prosecution against the malicious providers.*

Third, the overpaid money is not returned to the complaining consumer, which eliminates any reason for the consumer to lie if the consumer is not malicious. If the consumer is malicious and intends to defame the providers, it has to pay the price for the transactions before committing any harm, which serves as an automatic punishment. Consequently, its ability of defaming is limited by the money it has, which cannot be increased artificially by collusion, according to Property 1.

In addition, by Property 2, a deceived consumer can seek revenge with no restriction, which means a malicious provider cannot benefit from its action. We have the following properties.

Property 3. *A malicious provider cannot benefit by deceiving the consumers, and a malicious consumer will be automatically punished for defaming the providers.*

Property 4. *The maximum amount of loss for an innocent provider or consumer in a transaction with a malicious node is limited by the price specified in the contract.*

Property 3 removes financial incentives to cheat. A provider can make money only by serving others; a consumer will not be refunded for cheating. Property 4 makes sure that an innocent node will not be subject to negative discrimination attacks [12], in which nodes with excellent reputation can severely damage other nodes.

In summary, the malicious nodes cannot increase their power (in terms of available money) by cooperation, and they can only attack others at the cost of their own interests, i.e., money and/or reputation. Consequently, the total damage caused by the malicious nodes is strictly limited. They will eventually be rejected from the system due to poor reputation or be enforced to serve others for better reputation in order to stay in the system.

3.5.2 Defending Against Various Attacks

In the following, we consider four different types of attacks launched by a colluding group [12].

Unfairly high ratings: The members of a colluding group cooperate to artificially inflate each other's reputation by false reports, so that they can attack innocent nodes more effectively. In MARCH, a colluding group can inflate the reputation of some members only by moving the available money from other members to them. According to Property 1, the total money in the group cannot be inflated through cooperation. Although some members' reputation can be made better, other members' reputation will become worse, making them ineffective in attacks.

Unfairly low ratings: Providers collude with consumers to "bad-mouth" other providers that they want to drive out of the market. Because MARCH requires all consumers to pay money for their transactions before they can defame the providers, the malicious consumers lose their money (and reputation) for "bad-mouthing", which in turn makes it harder for them to stay in the system.

Negative discrimination: A provider only discriminates a few specific consumers by offering services with much lowered quality than what the contract specifies. It hopes to earn some "extra" money without damaging its reputation since it serves most consumer honestly. In MARCH, a provider cannot make such "extra" money because of the prosecution mechanism and Properties 2-3.

Positive discrimination: A provider gives an exceptionally good service to a few consumers with high reputation and an average service to the rest consumers. The strategy will work in an incentive scheme where a consumer's ability of affecting a provider's reputation is highly related to the consumer's own reputation, and vice versa. MARCH does not have this problem. The provider's reputation changes after a transaction is determined by how much money it receives for the service, not by the reputation of the consumer.

3.6 Discussions

In this section, we discuss other important issues on implementing MARCH.

3.6.1 Rewarding Delegation Members

The system should offer incentive for the delegation members to perform their tasks. A simple approach is for the provider and the consumer of a transaction to reward their delegation members with a certain amount of money, which should be less than the price of the transaction. After the transaction, the provider A signs an incentive payment certificate and sends the certificate to every delegation member $D_A(i)$, which reduces TM_A by a certain amount and then forwards the certificate to its delegation members, where the certificate is authenticated and the money is deposited. The consumer pays its delegation members in a similar way. If a delegation member in D_A refuses to serve, node A can increase k to bring new members into D_A.c

3.6.2 Money Refilling

Because the overpaid money will be frozen forever, the total amount of available money in the whole system may decrease over the time. As a result, the system may enter into deflation and lack sufficient money for the providers and the consumers to engage in transactions. This problem can be addressed by money refilling. The delegation members of a node A will replenish the total money TM_A of the node at a slow, steady rate. In this way, a minimal amount of service is provided to all consumers, even the free-riders, at all time, which we believe is reasonable. For additional service, a consumer has to contribute to the P2P network by also serving as a provider.

3.6.3 System Dynamics and Overhead

In a P2P network, nodes may join/leave the network at any time. When a node X leaves the network, its DHT table will be taken over by the closest neighbor X'. In MARCH, suppose X is a delegation member of A. After X leaves the network, X' will become a new member in A's delegation. In order to deal with abrupt departure, X' should cache the information kept at X, or it can learn the information from other delegation members after X leaves.

For a specific DHT network, there are better ways of selecting the delegation members than the approach in Section 3.3.1. Take Chord [1] as an example. We can select a subset of the $\log n$ neighbors of node A as the delegation D_A. In this way, the maintenance of the delegation is free as Chord already maintains the neighbor set.

The communication overhead of a transaction (excluding the actual service) consists of $O(k)$ control messages, which are sent from the provider (consumer) via k pairs of

Figure 3-3. Trustworthiness of delegation

Figure 3-4. Trustworthiness of k-pair delegation set

delegation members to the consumer (provider) throughput direct TCP connections. This overhead is quite small comparing to the typical services such as downloading video files of many gigabytes or sharing storage for months. More importantly, the overhead does not increase with the network size, which makes MARCH a scalable solution, comparing with other schemes [23, 24] whose overhead increases with the network size.

3.7 Simulation

In our simulations, the dishonest nodes fall into three categories with equal probability.

Category one: These nodes never offer services to others after receiving money, and always defame the providers after receiving services.

Category two: When these nodes find that they may be rejected from the system, they behave honestly. Otherwise, they behave in the same way as the nodes in category one.

Category three: When these nodes find that they may be rejected from the system, they behave honestly. Otherwise, they cheat their transaction partners with a probability taken from $[0.5, 1]$ uniformly at random.

If not explicitly specified otherwise, the system parameters are set as follows. The number of nodes is 100,000 and k is 5. The average number of dishonest nodes is 1,000. Initially, the total money for a node is 500, and the overpaid money is 0. The service price G estimated by the consumers is 10. The threshold T is 0.9. To satisfy the threshold requirement, the maximum selling price for a provider is denoted as max (max is the maximum value of L that keeps R above the threshold, calculated based on Eq. (4–2). If max is negative, then the node can no longer be a provider. If a dishonest node in Category two or three finds that its max value may become negative after additional malicious acts, it will behave honestly). The actual selling price is a random number taken uniformly from $(0, max]$. If a node can neither be a provider (due to poor reputation), nor be a consumer (due to little money), it is said to be rejected from the system.

If one participant in a transaction tries to deceive the other one, the transaction is called a failed transaction. We define "failed transaction ratio" as the number of failed transactions divided by the total number of transactions, and "overpaid money ratio" as the total amount of overpaid money divided by the total amount of money paid in the transactions. These metrics are used to assess the overall damage caused by dishonest nodes.

Figure 3-5. Most of malicious nodes are rejected within the first 50 transactions.

3.7.1 Effectiveness of Authority

In the first set of simulations, we study the trustworthiness of the delegations and the k-pair delegation sets. Figure 3-3 shows the number of untrustworthy delegations with respect to the number of dishonest nodes for $k = 3$, 5, and 7. Recall that a delegation is untrustworthy if at least half of its members are dishonest. Out of 100,000 delegations, only a few number of them are untrustworthy. For $k = 5$, the number of nodes with untrustworthy delegations is just 23 even when there are 3,000 dishonest nodes. Figure 3-4 shows the probability for an arbitrary k-pair delegation set to be untrustworthy (Section 3.3.2). The 5-pair delegation set is trustworthy with a probability larger than 99.8% even when there are 3,000 dishonest nodes. Note that when a delegation is untrustworthy, the dishonest members may not belong to the same colluding group. Without cooperation, the damage they can cause will be smaller.

3.7.2 Effectiveness of MARCH

The second set of simulations study the effectiveness of our incentive scheme. Figure 3-5 presents how the number of rejected nodes changes with the average number of transactions performed per node, which can be used as the logical time as the simulation progresses. Recall that the default number of dishonest nodes is 1,000. The figure shows that most dishonest nodes are rejected from the system within 50 transactions per node.

Figure 3-6. Failed transaction ratio and the overpaid money ratio drop quickly to small percentages within the first 100 transactions.

Because of money refilling, some rejected nodes will recover after enough money is refilled, but they will be rejected again after performing malicious transactions. No honest nodes are rejected from the system during the simulation.

Figure 3-6 shows that the failed transaction ratio drops quickly from 1.4% to 0.3% within the first 100 transactions per node, and the overpaid money ratio drops from 1.4% to 0.2% in the same period. As the time progresses, these ratios become even more insignificant. Note that the overpaid money ratio is smaller than the failed transaction ratio. This is because the dishonest providers have to lower their prices in order to compete with honest providers, which in turn lowers their ability to cause significant damage. Ironically, if a dishonest node with poor reputation wants to stay in the system, not only does it have to behave honestly to gain reputation, but also it has to do so with lower price in order to get consumers, which "repairs" the damage it does to the system previously.

Next, we study how the number of dishonest nodes affects the system performance. Figure 3-7 shows the overpaid money ratio after 250 transactions per node. We find that the ratio increases linearly with the number of dishonest nodes. Even when there are 3,000

Figure 3-7. Overpaid money ratio (measured after 250 transactions) increases linearly with the number of dishonest nodes.

Figure 3-8. Number of rejected dishonest nodes (measured after 250 transactions) increases linearly to the number of dishonest nodes.

Figure 3-9. Overpaid money ratio with respect to the threshold

Figure 3-10. Number of rejected nodes with respect to the threshold

dishonest nodes, the overpaid money ratio remains very small, just 0.15%. Figure 3-8 shows that the more the number of dishonest nodes, the more they are rejected.

Last, we study the impact of the threshold on the system performance. The threshold is used by a consumer to select the potential providers (Section 3.4.2). Figure 3-9 shows that the overpaid money ratio decreases linearly with the threshold value, which means the system performs better with a larger threshold. Figure 3-10 shows that the number of rejected dishonest nodes is largely insensitive to the threshold value. When the threshold is too low, some honest nodes may be rejected by the system because a smaller threshold allows the dishonest nodes to do more damage on the honest nodes, which may even cause some honest nodes to be rejected from the system due to defamed reputation. The numbers in the above two figures are measured after 250 transactions per node.

CHAPTER 4
CAPACITY-AWARE MULTICAST ALGORITHMS ON HETEROGENEOUS OVERLAY NETWORKS

4.1 Motivation

Multicast is an important network function for group communication among a distributed, dynamic set of heterogeneous nodes. The global deployment of IP multicast has been slow due to the difficulties related to heterogeneity, scalability, manageability, and lack of a robust inter-domain multicast routing protocol. Application-level multicast becomes a promising alternative.

Even though overlay multicast can be implemented on top of overlay unicast, they have very different requirements. In overlay unicast, low-capacity nodes only affect traffic passing through them and they create bottlenecks of limited impact. In overlay multicast, all traffic will pass all nodes in the group, and the multicast throughput is decided by the node with the smallest throughput, particularly in the case of reliable delivery. The strategy of assigning an equal number of children to each intermediate node is far from optimal. If the number of children is set too big, the low-capacity nodes will be overloaded, which slows down the entire session. If the number of children is set too small, the high-capacity nodes will be under-utilized.

There has been a flourish of capacity-aware multicast systems, which excel in optimizing single-source multicast trees but are not suitable for multi-source applications such as distributed games, teleconferencing, and virtual classrooms. Especially, they are insufficient in supporting applications that require any-source multicast with varied host capacities and dynamic membership.

To support efficient multicast, we should allow nodes in a P2P network to have different numbers of neighbors. We proposes two overlay multicast systems that support any-source multicast with varied host capacities and dynamic membership. We model the capacity as the maximum number of direct children to which a node is willing to forward multicast messages. We extend Chord [1] and Koorde [61] to be capacity-aware, and they are called CAM-Chord and CAM-Koorde, respectively.[1] A delicate CAM-Chord or CAM-Koorde overlay network is established for each multicast group. We then embed implicit degree-varying multicast trees on top of CAM-Chord or CAM-Koorde and develop multicast routines that automatically follow the implicit multicast trees to disseminate multicast messages. Dynamic membership management and scalability are inherited features from Chord or Koorde. Capacity-aware multiple-source mlticast are added features. Our analysis on CAM multicasting sheds light on the expected performance bounds with respect to the statistical distribution of host heterogeneity.

4.2 System Overview

Consider a multicast group G of n nodes. Each node $x \in G$ has a capacity c_x, specifying the maximum number of direct child nodes to which x is willing to forward the received multicast messages. The value of c_x should be made roughly proportional to the upload bandwidth of node x. Intuitively, x is able to support more direct children in a multicast tree when it has more upload bandwidth. In a heterogeneous environment, the capacities of different nodes may vary in a wide range. Our goal is to construct a resilient capacity-aware multicast service, which meets the capacity constraints of all nodes, allows frequent membership changes, and delivers multicast messages from any source to the group members via a dynamic, balanced multicast tree.

Our basic idea is to build the multicast service on top of a capacity-aware structured P2P network. We focus on extending Chord [1] and Koorde [61] for this purpose. The resulting systems are called CAM-Chord and CAM-Koorde, respectively. The principles and techniques developed should be easily applied to other P2P networks as well.

A CAM-Chord or CAM-Koorde overlay network is established for each multicast group. All member nodes (i.e., hosts of the multicast group) are randomly mapped by a hash function (such as SHA-1) onto an identifier ring $[0, N-1]$, where the next

[1] CAM stands for Capacity-Aware Multicast.

identifier after $N-1$ is zero. $N(=2^b)$ should be large enough such that the probability of mapping two nodes to the same identifier is negligible. Given an identifier $x \in [0, N-1]$, we define successor(x) as the node clockwise after x on the ring, and predecessor(x) as the node clockwise before x on the ring. $\widehat{x}$ refers to the node whose identifier is x; if there is not such a node, then it refers to successor(x). Node $\widehat{x}$ is said to be responsible for identifier x. With a little abuse of notation, x, $\widehat{x}$, successor(x), and predecessor(x) may represent a node or the identifier that the node is mapped to, depending on the appropriate context where the notations appear. Given two arbitrary identifiers x and y, $(x, y]$ is an identifier segment that starts from $(x+1)$, moves clockwise, and ends at y. The size of $(x, y]$ is denoted as $(y-x)$. Note that $(y-x)$ is always positive. It is the number of identifiers in the segment of $(x, y]$. The distance between x and y is $|x-y| = |y-x| = \min\{(y-x), (x-y)\}$, where $(y-x)$ is the size of segment $(y, x]$ and $(x-y)$ is the size of segment $(x, y]$. $(y, x]$ and $(x, y]$ form the entire identifier ring.

Before we discuss the CAMs, we briefly review Chord and Koorde. Each node x in Chord has $O(\log_2 n)$ neighbors, which are $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, ..., of the ring away from x, respectively. When receiving a lookup request for identifier k, a node forwards the request to the neighbor closest to k. This greedy algorithm takes $O(\log_2 n)$ hops with high probability to find $\widehat{k}$, the node responsible for k. Each node in Koorde has m neighbors. A node's identifier x is represented as a base-m number. Its neighbors are derived by shifting one digit (with value 0..m-1) into x from the right side and discard x's leftmost bit to maintain the same number of digits. When x receives a lookup request for k, the routing path of the request represents a transformation from x to k by shifting one digit of k at a time into x from the right until the request reaches the node responsible for k. Because k has $O(\log_m n)$ digits, it takes $O(\log_m n)$ hops with high probability to resolve a lookup request. Readers are referred to the original papers for more details.

Our first system is CAM-Chord, which is essentially a base-c_x (instead of base-2) Chord with c_x variable for different nodes. The number of neighbors of a node x is

$O(c_x \frac{\log n}{\log c_x})$, which is related to the node's capacity. Hence, different nodes may have different numbers of neighbors. The distances between x and its neighbors on the identifier ring are $\frac{1}{c_x}$, $\frac{2}{c_x}$, ..., $\frac{c_x-1}{c_x}$, $\frac{1}{c_x^2}$, $\frac{2}{c_x^2}$, ..., $\frac{c_x-1}{c_x^2}$, ... of the ring, respectively. Apparently CAM-Chord becomes Chord if $c_x = 2$, for all node x. We will design a greedy lookup routine for CAM-Chord and a multicast routine that essentially embeds implicit, capacity-constrained, balanced multicast trees in CAM-Chord. The multicast messages are disseminated via these implicit trees. It is a challenging problem to analyze the performance of CAM-Chord. The original analysis of Chord cannot be applied here because c_x is variable. We will provide a new set of analysis on the expected performance of the lookup routine and the multicast routine.

The second system is CAM-Koorde, which differs from Koorde in both variable number of neighbors and how the neighbors are calculated. This difference is critical in constructing balanced multicast trees. Each node x has c_x neighbors. The neighbor identifiers are derived by shifting x to the right for a variable number l of bits and then replacing the left-most l bits of x with a certain value. In comparison, Koorde shifts x one digit (base m) to the left and replaces the right-most digit. This subtle difference makes sure that CAM-Koorde spreads neighbors of a node evenly on the identifier ring while neighbors in Koorde tend to cluster together. We will design a lookup routine and a multicast routine that essentially performs broadcast. Remarkably, we show that this broadcast-based routine achieves roughly balanced multicast trees with the expected number of hops to a receiver being $O(\log n / E(\log c_x))$.

CAM-Chord maintains a larger number of neighbors than CAM-Koorde (by a factor of $O(\frac{\log n}{\log c_x})$), which means larger maintenance overhead. On the other hand, CAM-Chord is more robust and flexible because it offers backup paths in its topology [62]. The two systems achieve their best performances under different conditions. Our simulations show that CAM-Chord is a better choice when the node capacities are small and CAM-Koorde is better when the node capacities are large.

4.3 CAM-Chord Approach

CAM-Chord is an extension of Chord. It takes the capacity of each individual node into consideration. We first describe CAM-Chord as a regular P2P network that supports a lookup routine, which is to find $\widehat{k}$ for a given identifier k. We then present our multicast algorithm on top of CAM-Chord.

When a node joins or leaves the overlay topology, the lookup routine is needed to maintain the topology as it is defined. CAM-Chord is not designed for data sharing among peers as most other P2P networks (e.g., Chord [1]) do. There are NO data items associated with the identifier space. Each multicast group forms its own CAM-Chord network, whose sole purpose is to provide an infrastructure for dynamic capacity-aware multicasting.

4.3.1 Neighbors

For a node x in Chord, its neighbor identifiers are $(x + 2^i) \bmod N$, $\forall i \in [1..\log_2 N]$, which are $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, ..., of the ring away from x. CAM-Chord is a base-c_x Chord with variable c_x for different nodes. Let c_x^i mean $(c_x)^i$. The neighbor identifiers are $(x + j \times c_x^i) \bmod N$, denoted as $x_{i,j}$, $\forall j \in [1..c_x - 1]$, $\forall i \in [0..\frac{\log N}{\log c_x} - 1]$. i and j are called the level and the sequence number of $x_{i,j}$. Let $x_{0,0} = x$. The actual neighbors are $\widehat{x_{i,j}}$, which are the nodes responsible for $x_{i,j}$. Note that $\widehat{x_{0,1}}$ is always successor(x).

See an illustration in Figure 4-1 with $c_x = 3$. The level-one neighbors in CAM-Chord divide the whole ring into c_x segments of similar size. The level-two neighbors further divides a close segement $(x, x + \frac{1}{c_x}N]$ into c_x sub-segments of similar size. And so on. Consider an arbitrary identifier k. Let

$$i = \lfloor \frac{\log (k - x)}{\log c_x} \rfloor \tag{4–1}$$

$$j = \lfloor \frac{k - x}{c_x^i} \rfloor \tag{4–2}$$

Figure 4-1. Chord v.s. CAM-Chord neighbors ($c_x = 3$)

It can be easily verified that $x_{i,j}$ is the neighbor identifier of x that is counter-clockwise closest to k, which means $\widehat{x_{i,j}}$ is the neighbor node of x that is counter-clockwise closest to node $\widehat{k}$.[2] We call i the level and j the sequence number of k with respect to x.

4.3.2 Lookup Routine

CAM-Chord requires a lookup routine to assist member join/departure during a dynamic multicast session. This routine returns the address of node $\widehat{k}$ responsible for a given identifier k. $x.foo()$ denotes a procedure call to be executed at x. It is a local (or remote) procedure call if x is the local (or a remote) node. The set of identifiers that x is responsible for is $(\text{predecessor}(x),\ x]$. The set of identifiers that successor(x) is responsible for is $(x,\ \text{successor}(x)]$.

```
$x$.LOOKUP($k$)
1.  **if** $k \in (x,\ successor(x)]$ **then**
2.      **return** the address of $successor(x)$
3.  **else**
4.      $i \leftarrow \lfloor \frac{\log(k-x)}{\log c_x} \rfloor$
5.      $j \leftarrow \lfloor \frac{k-x}{c_x^i} \rfloor$
6.      **if** $k \in (x,\ \widehat{x_{i,j}}]$ **then**
```

[2] It is possible that $\widehat{x_{i,j}} = \widehat{k}$.

```
7.            return  the address of $\widehat{x_{i,j}}$
8.        else
              /* forward request to $x_{i,j}$ */
9.            return $\widehat{x_{i,j}}$.LOOKUP($k$)
```

First the LOOKUP routine checks if k is between x and successor(x). If so, LOOKUP returns the address of successor(x). Otherwise, it calculates the level i and the sequence number j of k. If k falls between x and $\widehat{x_{i,j}}$, which means $\widehat{x_{i,j}}$ is responsible for the identifier k, LOOKUP returns the address of $\widehat{x_{i,j}}$. On the other hand, if $\widehat{x_{i,j}}$ precedes k, then x forwards the lookup request to $\widehat{x_{i,j}}$. Because $\widehat{x_{i,j}}$ is x's closest neighbor preceding k, CAM-Chord makes a greedy progress to move the request closer to k.

4.3.3 Topology Maintenance

Because CAM-Chord is an extension of Chord, we use the same Chord protocols to handle member join/departure and to maintain the correct set of neighbors at each node. The difference is that our LOOKUP routine replaces the Chord LOOKUP routine. The details of the protocols can be found in [1].

The join operation of Chord can be optimized because two consecutive nodes on the ring are likely to have similar neighbors. When a new node joins, it first performs a lookup to find its successor and retrieves its successor's neighbors (called fingers in Chord). It then checks those neighbors to make correction if necessary. In a base-c Chord, the join complexity without the optimization is $O(c\frac{\log^2 n}{\log^2 c})$ for a constant c. The optimization reduces the complexity to $O(c\frac{\log n}{\log c})$.

CAM-Chord can be regarded as a base-c Chord where c is a random variable following the node-capacity distribution. It cannot always perform the above optimization because consecutive nodes may have different capacities, which make their neighbor sets different. When this happens, a new node x has to perform $O(c_x\frac{\log n}{\log c_x})$ lookups to find all its neighbors. The lookup complexity is $O(-\frac{\log n}{\log E(\frac{\log c}{c})})$ by Theorem 1 (to be proved). The join complexity is $O(c_x\frac{\log^2 n}{-\log c_x \log E(\frac{\log c}{c})})$, which would be reduced to $O(c\frac{\log^2 n}{\log^2 c})$ if the capacities of all nodes had the same value, i.e., c was a constant. This overhead is too high for a traditional P2P file-sharing application such as FastTrack, because the observations in [63] showed that over 20% of the connections last 1 minute or less and 60% of the IP addresses keep active for no more than 10 minutes each time after they join the system. But CAM-Chord is not designed for file-sharing applications. One appropriate CAM-Chord application is teleconferencing, which has far less participants than FastTrack and less dynamic member changes. We do not expect the majority of participants to keep joining and departing during a conference call. Another application is distributed games, where a user is more likely to play for a hour than for one minute.

CAM-Chord makes a tradeoff between capacity awareness and maintenance overhead, which makes it unsuitable for highly dynamic multicast groups. For them, CAM-Koorde is a better choice because a node only has c_x neighbors. Our future research will attempt to develop new techniques to overcome this limitation of CAM-Chord.

4.3.4 Multicast Routine

On top of the CAM-Chord overlay, we want to implicitly embed a dynamic, roughly balanced multicast tree for each source node. Each intermediate node in the tree should not have more children than its capacity. It should be emphasized that no explicit tree is built. Given a multicast message, a node x executes a MULTICAST routine, sending the message to c_x selected neighbors, which in turn execute the MULTICAST routine to further propagate the message. The execution of the MULTICAST routine at different nodes makes sure that the message follows a capacity-aware multicast tree to reach every member.

Let msg be a multicast message, k be an identifier, and x be a node executing the MULTICAST routine. The goal of x.MULTICAST(msg, k) is to deliver msg to all nodes whose identifiers belong to $(x, k]$. The routine is implemented as follows: x chooses c_x neighbors that split $(x, k]$ into c_x subsegments as even as possible. Each subsegment begins from a chosen neighbor and ends at the next clockwise chosen neighbor. x forwards the multicast message to each chosen neighbor, together with the subsegment assigned to this neighbor. When a neighbor receives the message and its subsegment, it forwards the message using the same method. The above process repeats until the size of the subsegment is reduced to one. The distributed computation of MULTICAST recursively divides $(x, k]$ into non-overlapping subsegments and hence no node will receive the multicast message more than once.

```
$x$.MULTICAST(msg, $k$)
1.  **if** $k = x$ **then**
2.      **return**
3.  **else**
4.      $i \leftarrow \lfloor \frac{\log k-x}{\log c_x} \rfloor$
5.      $j \leftarrow \lfloor \frac{k-x}{c_x^i} \rfloor$
        /*select children from level-$i$ neighbors preceding $k$*/
6.      $k' \leftarrow k$
7.      **for** $m = j$ **down to 1**
8.          $\widehat{x_{i,m}}$.MULTICAST$(msg, k')$
9.          $k' \leftarrow x_{i,m} - 1$
        /* select $(c_x - j - 1)$ children from level-$(i-1)$ neighbors */
10.     $l \leftarrow c_x$
11.     **for** $m = c_x - j - 1$ **down to** 1
12.         $l \leftarrow l - \frac{c_x}{c_x - j}$     /* for even separation */
13.         $\widehat{x_{i-1,\lceil l \rceil}}$.MULTICAST$(msg, k')$
14.         $k' \leftarrow x_{i-1,\lceil l \rceil} - 1$
        /* select $x$'s successor */
15.     $\widehat{x_{0,1}}$.MULTICAST$(msg, k')$
```

To split $(x, k]$ evenly, x first calculates the level i and the sequence number j of k with respect to x (Line 4-5). Then neighbors $\widehat{x_{i,m}}$ ($\forall m \in [1..j]$) at the ith level preceding k

are selected as children of x in the multicast tree (Line 6-9). We also select x's successor, which is $\widehat{x_{0,1}}$ (Line 15). Since $j+1$ may be less than c_x, in order to fully use x's capacity, $c_x - 1 - j$ neighbors at the $(i-1)$th level are chosen; Line 10-14 ensures that the selection is evenly spread at the $(i-1)$th level. Because the algorithm selects neighbors that divide $(x, k]$ as even as possible, it constructs a multicast tree that is roughly balanced. At Line 9, we optimize the code by using $k \leftarrow x_{i,m} - 1$ instead of $k \leftarrow \widehat{x_{i,m}} - 1$. That is because there is no node in $(x_{i,m},\ \widehat{x_{i,m}})$ by the definition of $\widehat{x_{i,m}}$.

4.3.5 Analysis

Assume $c_x \geq 2$ for every node x. We analyze the performance of the LOOKUP routine and the multicast routine of CAM-Chord. Suppose a node x receives a lookup request for identifier k and it forwards the request to a neighbor node $\widehat{x_{i,j}}$ that is closest to k. We call $\frac{k-\widehat{x_{i,j}}}{k-x}$ the distance reduction ratio, which measures how much closer the request is from k after one-hop routing. The following lemma establishes an upper bound on the distance reduction ratio with respect to c_x, which is a random variable of certain distribution.

Lemma 4.1. *Suppose a node x forwards a lookup request for identifier k to a neighbor $\widehat{x_{i,j}}$. If $\widehat{x_{i,j}} \in (x, k]$, then*

$$E\left(\frac{k-\widehat{x_{i,j}}}{k-x}\right) < E\left(\frac{\ln c_x}{c_x - 1}\right)$$

Proof. Based on the algorithm of the LOOKUP routine, i must be the height of k with respect to x. By (4–1) and (4–2), k can be written as

$$k = x + jc_x^i + l$$

where $j \in [1..c_x - 1]$ is the sequence number of k with respect to x and $l \in [0..c_x^i)$.

$$k - x = jc_x^i + l \tag{4–3}$$

By definition (Section 4.3.1), $x_{i,j} = x+jc_x^i$. Because x, $x_{i,j}$, $\widehat{x_{i,j}}$, and k are in clockwise order on the identifier ring, we have

$$k - \widehat{x_{i,j}} \leq k - x_{i,j} = l \tag{4–4}$$

By (4–3) and (4–4), we have

$$\frac{k-\widehat{x_{i,j}}}{k-x} \leq \frac{l}{jc_x^i + l} < \frac{c_x^i}{jc_x^i + l}$$

We now derive the expected distance reduction ratio. $E(\frac{k-\widehat{x_{i,j}}}{k-x})$ depends on three random variables, j, l, and c_x. Because the location of k is arbitrary with respect to x, we can consider j and l as independent random variables with uniform distributions on their respective value ranges.

$$\begin{aligned} E(\frac{k'-\widehat{x_{i,m}}}{k-x}) &= E(\frac{1}{c_x-1}\sum_{j=1}^{c_x-1}\frac{1}{c_x^i}\int_0^{c_x^i}\frac{k'-\widehat{x_{i,j}}}{k-x}dl) \\ &= E(\frac{1}{c_x-1}\sum_{j=1}^{c_x-1}\frac{1}{c_x^i}\int_0^{c_x^i}\frac{c_i^x}{jc_x^i+l}dl) \\ &= E(\frac{1}{c_x-1}(\sum_{j=1}^{c_x-1}(\ln(j+1)-\ln j))) = E(\frac{\ln c_x}{c_x-1}) \end{aligned}$$

□

Theorem 4.2. *Let c_x, for all nodes x, be independent random variables of certain distribution. The expected length of a lookup path in CAM-Chord is $O(-\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x})})$.*

Proof. Suppose $(x_1, x_2, ..., x_m)$ is a prefix of a lookup path for identifier k, where x_1 is the node that initiates the lookup, and x_i, $i \in [1..m]$, and k are in clockwise order on the identifier ring. Because the nodes are randomly mapped to the identifier ring by a hash function, the distance reduction ratio after each hop is independent of those after other hops. Consequently $\frac{k-x_i}{k-x_{i-1}}, i \in [2..m]$, are independent random variables.

$$\begin{aligned}
E(k-x_m) &= E(\frac{k-x_m}{k-x_{m-1}} \cdot \frac{k-x_{m-1}}{k-x_{m-2}} \cdot \ldots \cdot \frac{k-x_2}{k-x_1} \cdot (k-x_1)) \\
&= E(\frac{k-x_m}{k-x_{m-1}}) \cdot E(\frac{k-x_{m-1}}{k-x_{m-2}}) \cdot \ldots \cdot E(\frac{k-x_2}{k-x_1}) \cdot E(k-x_1) \\
&< (E(\frac{\ln c_x}{c_x - 1}))^{m-1} \cdot N
\end{aligned} \tag{4–5}$$

where c_x is a random variable with the same distribution as $c_{x_i}, i \in [1..m]$. Next we derive the value of m that ensures $E(k-x_m) < \frac{N}{n}$, which is the average distance between two adjacent nodes on the identifier ring. The following is a sufficient condition to achieve $E(k-x_m) < \frac{N}{n}$.

$$\begin{aligned}
(E(\frac{\ln c_x}{c_x - 1}))^{m-1} \cdot N &= \frac{N}{n} \\
m &= -\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x - 1})}
\end{aligned}$$

If $E(k-x_m) < \frac{N}{n}$, the expected number of additional routing hops from x_m to k is $O(1)$. $O(m) = O(-\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x})})$ gives the expected length of the lookup path. □

It is natural that the expected length of a lookup path in CAM-Chord depends on the probability distribution of c_x, which affects the topological structure of the overlay network. For a given distribution, an upper bound of the expected path length can be derived from Theorem 4.2. The following theorem gives an example.

Theorem 4.3. *Suppose the node capacity c_x follows a uniform distribution with $E(c_x) = c$. The expected length of a lookup path in CAM-Chord is $O(\frac{\ln n}{\ln c})$.*

Proof. Suppose the range of c_x is $[t_1..t_2]$ with $E(c_x) = c$. We perform Big-O reduction as follows.

$$\begin{aligned}
E(\frac{\ln c_x}{c_x}) =& \Theta(\sum_{c_x=t_1}^{t_2} \frac{\ln c_x}{c_x} \times \frac{1}{t_2 - t_1 + 1}) \\
=& \Theta(\int_{t_1}^{t_2} \frac{\ln c_x}{c_x} dc_x \times \frac{1}{t_2 - t_1 + 1}) \\
=& \Theta((\frac{1}{2}\ln^2 t_2 - \frac{1}{2}\ln^2 t_1) \times \frac{1}{c}) \\
=& \Theta(\frac{\ln^2 c}{c}) \qquad \text{because } t_2 \le 2c \text{ and } t_1 \le c
\end{aligned}$$

Therefore,

$$\ln E(\frac{\ln c_x}{c_x}) = \Theta(\ln \frac{\ln^2 c}{c}) = \Theta(-\ln c)$$

By Theorem 4.2, $O(-\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x})}) = O(\frac{\ln n}{\ln c})$. □

Other distributions of c_x may be analyzed similarly.

Next we analyze the performance of the MULTICAST routine in CAM-Chord. Suppose x executes x.MULTICAST(msg, k), which is responsible for delivering msg to all nodes in the identifier segment (x, k). Specifically, x forwards msg to some neighbor nodes $\widehat{x_{i,m}}$ by remotely invoking $\widehat{x_{i,m}}$.MULTICAST(msg, k'), which is responsible for delivering msg to a smaller subsegment $(\widehat{x_{i,m}}, k')$, where $\widehat{x_{i,m}}, k' \in (x, k)$. It is a typical divide-and-conquer strategy. We call $\frac{k' - \widehat{x_{i,m}}}{k - x}$ the segment reduction ratio, which measures the degree of reduction in problem size after one-hop multicast routing. The following lemma establishes an upper bound on the segment reduction ratio with respect to c_x, which is a random variable of certain distribution.

Lemma 4.4. *Suppose a node x forwards a multicast message to a neighbor $\widehat{x_{i,m}}$, i.e., x.MULTICAST(msg, k) calls $\widehat{x_{i,m}}$.MULTICAST(msg, k'). It must be true that*

$$E(\frac{k' - \widehat{x_{i,m}}}{k - x}) < E(\frac{\ln c_x}{c_x - 1})$$

Proof. Based on the algorithm of the MULTICAST routine, the execution of x.MULTICAST(msg, k) will divide its responsible segment (x, k) into c_x subsegments, and $\widehat{x_{i,m}}$ is responsible for delivering msg to all nodes in one subsegment $(\widehat{x_{i,m}}, k')$. The largest subsegment is created by Lines 6-9. When Line 7 is executed for $m \in [j-1..1]$, $k' = x_{i,m+1} - 1$. Therefore,

$$\begin{aligned} k' - \widehat{x_{i,m}} &< x_{i,m+1} - x_{i,m} \\ &= x + (m+1)c_x^i - x - mc_x^i \\ &= c_x^i \end{aligned} \tag{4–6}$$

By Line 4, i is the level of k with respect to x. By (4–1) and (4–2), k can be written as

$$k = x + jc_x^i + l$$

where $j \in [1..c_x - 1]$ is the sequence number of k with respect to x and $l \in [0..c_x^i)$.

$$k - x = jc_x^i + l \tag{4–7}$$

By (4–6) and (4–7), we have

$$\frac{k' - \widehat{x_{i,m}}}{k - x} < \frac{c_i^x}{jc_x^i + l}$$

We now derive the expected segment reduction ratio. $E(\frac{k'-\widehat{x_{i,m}}}{k-x})$ depends on three random variables, j, l, and c_x. Because the location of k is arbitrary with respect to x, we can consider j and l as independent random variables with uniform distributions on their respective value ranges.

$$\begin{aligned} E(\frac{k' - \widehat{x_{i,m}}}{k - x}) &= E(\frac{1}{c_x - 1}\sum_{j=1}^{c_x-1}\frac{1}{c_x^i}\int_0^{c_x^i}\frac{k' - \widehat{x_{i,j}}}{k - x}dl) \\ &= E(\frac{1}{c_x - 1}\sum_{j=1}^{c_x-1}\frac{\ln c_x}{c_x^i}\int_0^{c_x^i}\frac{c_i^x}{jc_x^i + l}dl) \\ &= E(\frac{1}{c_x - 1}(\sum_{j=1}^{c_x-1}(\ln(j+1) - \ln j))) \\ &= E(\frac{\ln c_x}{c_x - 1}) \end{aligned} \tag{4–8}$$

□

A multicast path is defined as the path that the MULTICAST routine takes to deliver a multicast message from a source node to a destination node. The proofs of the following two theorems are very similar to those of Theorems 4.2 and 4.3, due to the similarity between Lemma 4.4 and Lemma 4.1, on which the theorems are based. To avoid excessive repetition and to conserve space, we omit the proofs for Theorems 4.5 and 4.6.

Theorem 4.5. *Let c_x, for all nodes x, be independent random variables of certain distribution. The expected length of a multicast path in CAM-Chord is $O(-\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x})})$.*

Theorem 4.6. *Suppose the node capacity c_x follows a uniform distribution and $E(c_x) = c$. The expected length of a multicast path in CAM-Chord is $O(\frac{\ln n}{\ln c})$.*

4.4 CAM-Koorde Approach

This section proposes CAM-Koorde. For any node x in CAM-Koorde, its number of neighbors is exactly equal to its capacity c_x. The maintenance overhead of CAM-Koorde is smaller than that of CAM-Chord due to a smaller number of neighbors.

Like Koorde, CAM-Koorde embeds the Bruijn graph in the identifier ring. On the other hand, it has two major differences from Koorde, which are critical to our capacity-aware multicast service.

• The first difference is about neighbor selection. The neighbor identifiers of a node x in Koorde are derived by shifting x one digit (base m) to the left and then replacing the last digit with 0 through $m - 1$. The neighbor identifiers differ only at the last digit. Consequently they are clustered and often refer to the same physical node. For the purpose of multicast, we want the neighbors to spread evenly on the identifier ring. The neighbor identifiers of a node x in CAM-Koorde are derived by shifting x one or more bits to the right and then replacing the high-order bits with 0 through certain number. The neighbor identifiers differ at the high-order bits, and they are evenly distributed on the identifier ring.

• The second difference is about the number of neighbors. Koorde requires every node to have the same number of neighbors. CAM-Koorde allows nodes to have different numbers of neighbors.

4.4.1 Neighbors

Let $N = 2^b$. In CAM-Koorde, x has c_x neighbors, which are categorized into three groups. All computations are assumed to be modulo N.

Figure 4-2. CAM-Koorde topology with identifier space [0..63]

• The basic group has four neighbors. Two are x's predecessor and successor. The other two are the nodes responsible for identifiers $(x/2)$ and $2^{b-1} + (x/2)$, respectively.

• After the basic group, there are $c_x - 4$ remaining neighbors. Let $s = \lfloor \log(c_x - 4) \rfloor$. If $s > 1$, we shall shift x by s bits to the right to derive the neighbor identifiers.[3] If $s > 1$, then let $t = 2^s$; otherwise let $t = 0$. The neighbors in the second group are the nodes responsible for identifiers $(i \times 2^{b-s} + x/2^s)$, $\forall\ i \in [0..t-1]$.

• After the basic and second groups, there are $t' = c_x - 4 - t$ remaining neighbors. Let $s' = s + 1$. The neighbors in the third group are the nodes responsible for identifiers $(i \times 2^{b-s'} + x/2^{s'})$,$\forall i \in [0..t'-1]$.

It is required that $c_x \geq 4$. The basic group is mandatory. The optional second and third groups pick up the remaining neighbors.

An example is given in Figure 4-2, showing the neighbors of node 36 (100100) whose capacity is 10. The binary representation of the node identifier is given in the parentheses. The basic group is

$$\{35\ (100011),\ 37\ (100101),\ 18\ (010010),\ 50\ (110010)\}$$

The second group is

$$\{9\ (001001),\ 25\ (011001), 41\ (101001), 57\ (111001)\}$$

[3] If $s = 1$, it means to shift one bit. The basic group already does that.

The third group is

$$\{4\ (000100), 12\ (001100)\}$$

4.4.2 Lookup Routine

Definition 1. *Given two b-bit identifiers x and k, if an l-bit prefix of x matches an l-bit suffix of k, we say x and k share l ps-common bits. $x = k$ if the two share b ps-common bits.*

Similar to CAM-Chord, a lookup routine is needed in CAM-Koorde for member join/departure. First consider an N-node network with every identifier having a corresponding node. Given an identifier k, suppose node x wants to query for the address of node k. The lookup routine forwards the lookup request along a chain of neighbors whose identifiers share progressively more ps-common bits with k. Starting from x, we identify a neighbor that has the longest prefix matching the suffix of k. More specifically, if the third group is not empty and a third-group neighbor can be derived by selecting the $(\lfloor \log(c_x - 4) \rfloor + 1)$ bits of k that precedes the current ps-common bits and shifting them from the left into x, then the lookup request is forwarded to this neighbor. Otherwise, if the second group is not empty and a second-group neighbor can be derived by selecting the $\lfloor \log(c_x - 4) \rfloor$ bits of k that precedes the current ps-common bits and shifting them from the left into x, then the lookup request is forwarded to this neighbor. Otherwise, we forward the request to a first-group neighbor that increases the number of ps-common bits by one. To determine each subsequent node on the forwarding path, a similar process repeats by shifting more bits of k into the identifier of the next hop receiver. After at most b hops, the request can reach node k.

Now suppose $n \ll N$, which is normally the case. We still calculate the chain of neighbor identifiers in the above way, which essentially transforms identifier x to identifier k in a series of steps, each step adding one or more bits from k. Once the next neighbor identifier y on the chain is calculated, the request is forwarded to $\widehat{y}$, which in turn calculates its neighbor identifier that should be the next on the forwarding path and then forwards the request.

The pseudo code of the LOOKUP routine is shown below. It uses the high-order bits of the node identifier to match the low-order bits of k, which is different from Koorde's routine and is critical for our multicast routine, to be discussed shortly.

```
$x$.LOOKUP($k$)
1.  **if** $k \in (\text{predecessor}(x), x]$ **then**
2.      **return** the address of $x$
3.  **if** $k \in (x, \text{successor}(x)]$ **then**
4.      **return** the address of successor($x$)
5.  let $m_1$ be the number of ps-common bits shared by $x$ and $k$
6.  find the neighbor $y$ that shares the largest number $m_2$ of ps-common bits with $k$
7.  **if** $m_1 \leq m_2$ **then**
8.      **return** $y$.LOOKUP($k$)
9.  **else**
10.     **if** $|k - \text{predecessor}(x)| < |k - \text{successor}(x)|$ **then**
11.         **return** predecessor($x$).LOOKUP($k$)
12.     **else**
13.         **return** successor($x$).LOOKUP($k$)
```

Koorde uses Chord's protocols with a new LOOKUP routine for node join/departure, so does CAM-Koorde.

4.4.3 Multicast Routine

When a node receives a multicast message, it forwards the message to all neighbors except those that have received or are receiving the message. Because neighbor connections are bidirectional, it is easy for a node to perform the checking through a short control packet. The overhead is negligible when the message is large, such as a video file. Note that a node does not have to wait for the entire message to arrive before forwarding it to neighbors. The forwarding is done on per packet basis, but the checking is performed only for the first packet of a message, which carries the message header. The pseudo code of the MULTICAST routine is shown below.

```
$x$.MULTICAST($msg$)
1.  **for** each neighbor $y$ **do**
2.      **if** $y$ has not received or is not receiving $msg$ **then**
3.          $y$.MULTICAST($msg$)
```

4.4.4 Analysis

Lemma 4.7. *Let c_x, for all nodes x, be independent random variables of certain distribution. The expected length of the shortest path between two nodes in CAM-Koorde is $O(\log n/E(\log c_x))$.*

Proof. Consider two arbitrary nodes x_0 and y. Let y' be an $O(\log n)$-bit prefix of y. We show there exists a path of length $O(\log n/E(\log c_x))$ that reaches a node $\widehat{x_m}$ with y' also as its prefix.

We construct a physical path (x_0, $\widehat{x_1}$, $\widehat{x_2}$, ..., $\widehat{x_{m-1}}$, $\widehat{x_m}$) as follows. Node x_0 has a basic- or second-group neighbor $\widehat{x_1}$, where x_1 is derived by shifting $\max\{1, \lfloor\log(c_{x_0} - 4)\rfloor\}$ low-order bits of y' into x_0 from the left.[4] The bits of y' that have been used in shifting are called used bits. Similarly, $\widehat{x_1}$ has a second-group neighbor $\widehat{x_2}$, where x_2 is derived by shifting $\max\{1, \lfloor\log(c_{\widehat{x_1}} - 4)\rfloor\}$ low-order unused bits of y' into $\widehat{x_1}$ ... Finally, $\widehat{x_{m-1}}$ has a second-group neighbor $\widehat{x_m}$, where x_m is derived by shifting the remaining $\max\{1, \lfloor\log(c_{\widehat{x_{m-1}}} - 4)\rfloor\}$ low-order unused bits of y' into $\widehat{x_{m-1}}$. The length of path (x_0, $\widehat{x_1}$, $\widehat{x_2}$, ..., $\widehat{x_{m-1}}$) is m. The total number of used bits of y' is $\sum_{i=0}^{m-1} \max\{1, \lfloor\log(c_{\widehat{x_i}} - 4)\rfloor\}$,

[4] If $c_x < 6$, we can pick x_1 from the basic group, which shifts one bit of y' into x_0; if $c_x \geq 6$, we can pick x_1 from the second group, which shifts $\lfloor\log(c_{x_0} - 4)\rfloor$ bits of y' into x_0.

which is $O(\log n)$. Let c_x be a random variable of the same distribution as $c_{\widehat{x_i}}$, $\forall i \in [0..m-1]$.

$$\sum_{i=0}^{m-1} \max\{1, \lfloor \log(c_{\widehat{x_i}} - 4) \rfloor\} = O(\log n)$$

$$\sum_{i=0}^{m-1} \log c_{\widehat{x_i}} = O(\log n)$$

$$E(\sum_{i=0}^{m-1} \log c_{\widehat{x_i}}) = O(\log n)$$

$$mE(\log c_x) = O(\log n)$$

$$m = O(\log n / E(\log c_x))$$

Next we construct an identifier list (x_0, x'_1, x'_2, ..., x'_{m-1}, x'_m) as follows. x'_1 is derived by shifting $\max\{1, \lfloor \log(c_{x_0} - 4) \rfloor\}$ low-order bits of y' into x_0 from the left. x'_2 is derived by shifting $\max\{1, \lfloor \log(c_{\widehat{x_1}} - 4) \rfloor\}$ low-order unused bits of y' into x'_1 ... Finally, where x'_m is derived by shifting the remaining $\max\{1, \lfloor \log(c_{\widehat{x_{m-1}}} - 4) \rfloor\}$ low-order unused bits of y' into x'_{m-1}.

y' is an $O(\log n)$-bit prefix of both x'_m and y. The distance between x'_m and y on the identifier ring, $|x'_m - y|$, is $O(\frac{N}{n})$. Note that $\frac{N}{n}$ is the average distance between any two adjacent nodes on the ring. If we can show that $E(|\widehat{x_m} - x'_m|) < \frac{N}{n}$, then $E(|\widehat{x_m} - y|) \leq E(|\widehat{x_m} - x'_m| + |x'_m - y|) = E(|\widehat{x_m} - x'_m|) + E(|x'_m - y|) = O(\frac{N}{n})$, which means that the expected number of hops from $\widehat{x_m}$ to y is $O(1)$.

$\forall i \in [1..m]$, define a random variable $\Delta_i = |\widehat{x_i} - x_i|$. Because x_i can be anywhere in (predecessor($\widehat{x_i}$), $\widehat{x_i}$), we have

$$E(\Delta_i) = \frac{1}{2} E(|\text{predecessor}(\widehat{x_i}), \widehat{x_i}|) = \frac{N}{2n} \qquad (4\text{–}9)$$

where $\frac{N}{n}$ is the expected distance between two adjacent nodes on the identifier ring.

x_i and x'_i are derived from $\widehat{x_{i-1}}$ and x'_{i-1}, respectively, by shifting the same $\max\{1, \lfloor\log(c_{\widehat{x_{i-1}}} - 4)\rfloor\}$ bits of y' in from the left. Therefore,

$$|x_i - x'_i| = \frac{|\widehat{x_{i-1}} - x'_{i-1}|}{2^{\max\{1, \lfloor\log(c_{\widehat{x_{i-1}}} - 4)\rfloor\}}}$$

It follows that, $\forall i \in [i..m]$,

$$\begin{aligned} |\widehat{x_i} - x'_i| &\le |\widehat{x_i} - x_i| + |x_i - x'_i| \\ &= \Delta_i + \frac{|\widehat{x_{i-1}} - x'_{i-1}|}{2^{\max\{1, \lfloor\log(c_{\widehat{x_{i-1}}} - 4)\rfloor\}}} \end{aligned}$$

By induction, we have

$$|\widehat{x_m} - x'_m| \le \sum_{i=1}^{m} \Delta_i \prod_{j=i}^{m-1} \frac{1}{2^{\max\{1, \lfloor\log(c_{\widehat{x_j}} - 4)\rfloor\}}}$$

Because $c_x \ge 4$ for any node x in CAM-Chord, $\max\{1, \lfloor\log(c_{\widehat{x_j}} - 4)\rfloor\} \ge 1$. Hence,

$$\begin{aligned} |\widehat{x_m} - x'_m| &\le \sum_{i=1}^{m} \Delta_i (\frac{1}{2})^{m-i} \\ E(|\widehat{x_m} - x'_m|) &\le \sum_{i=1}^{m} E(\Delta_i)(\frac{1}{2})^{m-i} \\ &= \frac{N}{2n} \sum_{i=1}^{m} (\frac{1}{2})^{m-i} \\ &= \frac{N}{2n} \sum_{i=0}^{m-1} (\frac{1}{2})^{i} < \frac{N}{n} \end{aligned}$$

Consequently, the expected number of hops from $\widehat{x_m}$ to x'_m and then to y is $O(1)$. The expected length of the entire path from x_0 to y is $O(m) = O(\log n / E(\log c_x))$. □

Theorem 4.8. *Let c_x, for all nodes x, be independent random variables of certain distribution. The expected length of a multicast path in CAM-Koorde from a source node to a member node is $O(\log n / E(\log c_x))$.*

Proof. According to the MULTICAST routine, a multicast packet is delivered in CAM-Koorde by broadcast, which follows the shortest paths to the member nodes.

The expected length of a multicast path from a source node to a member node is $O(\log n / E(\log c_x))$ by Lemma 4.7. $\square$

Theorem 4.9. *Suppose the node capacity c_x follows a uniform distribution and $E(c_x) = c$. The expected length of a multicast path in CAM-Koorde from a source node to a member node is $O(\log n / \log c)$.*

Proof. Suppose the range of c_x is $[t_1..t_2]$ with $E(c_x) = c$. We perform Big-O reduction as follows.

$$\begin{aligned}
E(\log c_x) =& \frac{1}{t_2 - t_1 + 1} \sum_{c_x = t_1}^{t_2} \log c_x \\
=& \Theta\Big(\frac{1}{t_2 - t_1 + 1} \int_{t_1}^{t_2} \log c_x dc_x\Big) \\
=& \Theta\Big(\frac{1}{t_2 - t_1 + 1} ((t_2 \log t_2 - t_2) - (t_1 \log t_1 - t_1))\Big) \\
=& \Theta(\log c) \qquad \text{because } \; t_2 \leq 2c \text{ and } t_1 \leq c
\end{aligned}$$

By Theorem 4.8, $O(\log n / E(\log c_x)) = O(\frac{\log n}{\log c})$. $\square$

4.5 Discussions

4.5.1 Group Members with Very Small Upload Bandwidth

A node x with very small upload bandwidth should only be a leaf in the multicast trees unless itself is the data source. In order to make sure that the MULTICAST routine does not select x as an intermediate node in any multicast tree, x must not be in the CAM-Chord (or CAM-Koorde) overlay network. Instead, it joins as an external member. x asks a node y known to be in CAM-Chord (or CAM-Koorde) to perform LOOKUP(k) for an arbitrary identifier k, which returns a random node z in the overlay network. x then attempts to join z as an external member. If z cannot support x, z forwards x to successor(z). If z admits x as an external member, z will forward the received multicast messages to x and x will multicast its messages via z. If z leaves the group, x must rejoin via another node in CAM-Chord (or CAM-Koorde).

4.5.2 Proximity and Geography

The overlay connections between neighbors may have very different delays. Two neighbors may be separated by transcontinental links, or they may be on the same LAN. There exist some approaches to cope with geography, for example, Proximity Neighbor Selection and Geographic Layout. With Proximity Neighbor Selection, nodes are given some freedom in choosing neighbors based on other criteria (i.e. latencies) in addition to the arithemic relations between their identifiers. With Geographic Layout, node identifiers are chosen in a geographically informed manner. The main idea is to make geographically closeby nodes form clusters in the overlay. Readers are referred to [64, 62] for details.

Extension to the existing P2P networks, CAMs can naturally inherit most of those features without much additional effort. For example, instead of choosing the ith neighbor to be $(x + 2^i)$, a proximity optimization of Chord allows the ith neighbor to be any node within the range of $[(a + 2^i),\ (a + 2^{i+1})$, which does not affect the complexities [62]. This optimization can also be applied to CAM-Chord (which is an extention of Chord) without affecting the complexities. A node x can choose any node whose identifier belongs to the segment $[x + j \times c_x^i, x + (j + 1) \times c_x^i)$ as the neighbor $x_{i,j}$. Given this freedom, some heuristics such as smallest delay first, may be used to choose neighbors to promote proximity-clustering. Specifically, a node x can progressively scan the nodes in the allowed segment for neighbor $x_{i,j}$, for example, by following the successor link to probe the next node in the segment after every 100k data bits sent by x, which trivializes the probing overhead. x always use the nearest node it has found recently as its neighbor.

4.6 Simulation

Throughput and latency are two major performance metrics for a multicast application. We evaluate the performance of CAMs from these two aspects. We simulate multicast algorithms on top of CAM-Chord, Chord, CAM-Koorde, and Koorde, respectively. The identifier space is $[0, 2^{19})$. If not specified otherwise, the default number of nodes in an overlay network is $100,000$, and the node capacities are taken from [4..10] with uniform probability. The upload bandwidths of the nodes are randomly distributed in a default range of $[400, 1000]$ kbps.

It should be noted that the value ranges may go far beyond the default ones in specific simulations. In our simulations, $c_x = \lfloor B_x/p \rfloor$, where B_x is the node's upload bandwidth and p is a system parameter of CAMs, specifying the desired bandwidth per link in the multicast tree. By varying the value of p, we can construct CAMs with different average node capacities, which also mean different average numbers of children per non-leaf node and consequently different tree depths (latency). If the average node capacity c is not the default value of 7, the node capacities are taken uniformly from $[4, 2c - 4]$.

4.6.1 Throughput

We compare the sustainable throughput of multicast systems based on CAM-Chord, Chord, CAM-Koorde, and Koorde. Throughput is defined as the rate at which data can be continuously delivered from a source node to all other nodes. Due to limited buffer space at each node, the sustainable multicast throughput is decided by the link with the least allocated bandwidth in the multicast tree. CAM-Chord and CAM-Koorde produce much larger throughput because a node's capacity c_x (which is its number of children in the multicast tree) is adjustable based on the node's upload bandwidth. The primary advantage of CAMs over the Chord/Koorde is their ability to adapt the overlay topology according to host heterogeneity.

Figure 4-3 compares the throughput of CAM-Chord, Chord, CAM-Koorde, and Koorde with respect to the average number of children per non-leaf node in the multicast tree. CAMs perform much better. Their throughput improvement over Chord and Koorde is 70-80% when the nodes' upload bandwidths are chosen from the rather narrow default range of $[400, 1000]$ kbps. The larger the upload-bandwidth range, the more the throughput improvement, as demonstrated by Figure 4-4. In general, let $[a, b]$ be the range of upload bandwidth. The upper bound b of the range is shown on the horizontal axis

Figure 4-3. Multicast throughput with respect to average number of children per non-leaf node

Figure 4-4. Throughput improvement ratio with respect to upload bandwidth range

Figure 4-5. Multicast throughput with respect to size of the multicast group

Figure 4-6. Throughput vs. average path length

of Figure 4-4, while the lower bound a is fixed at 400 kbps. The figure shows that the throughput improvement by CAMs increases when the upload-bandwidth range is larger, representing a greater degree of host heterogeneity. The simulation data also indicate that the throughput ratio of CAM-Chord (CAM-Koorde) over Chord (Koorde) is roughly proportional to $\frac{a+b}{2a}$.

Figure 4-5 shows the multicast throughput with respect to the size of the multicast group. According to the simulation results, the throughput is largely insensitive to the group size.

4.6.2 Throughput vs. Latency

We measure multicast latency by the average length of multicast paths. Latency is determined by both the number of hops and the hop delay. In CAM-Chord and CAM-Koorde, the overlay links are randomly formed among the nodes due to the use of DHT. The latency of an overlay path is statistically proportional to the number of hops. That's why we used the number of hops to characterize the latency performance. In fact, the measurement in terms of number of hops carries information beyond latency. It is also an indication of how many times a message has to be relayed, which is a resource consumption issue. With the proximity neighbor selection in Section 4.5.2, the latency is no longer proportional to the number of hops. We add a simulation in Section 4.6.4 to study this case, where the actual delay is measured.

Both throughput and latency are functions of average node capacity. With a larger average node capacity (achieved by a smaller value of p), the throughput decreases due to more children per non-leaf node and the latency also decreases due to smaller tree depth. There exists a tradeoff between throughput and latency, which is depicted by Figure 4-6. Higher throughput can be achieved at the cost of longer routing paths. Given the same upload bandwidth distribution, the system's performance can be tuned by adjusting p. The figure also shows that, for relatively small throughput (less than 46kbps in the figure) — namely, for large node capacities — CAM-Koorde slightly outperforms CAM-Chord; for relatively large throughput (greater than 46kbps in the figure) — namely, for small node capacities — CAM-Chord outperforms CAM-Koorde, which will be further explained in Section 4.6.4.

4.6.3 Path Length Distribution

Figure 4-7 and Figure 4-8 present the statistical distribution of multicast path lengths, i.e., the number of nodes that can be reached by a multicast tree in certain number of hops. Each curve represents a simulation with node capacities chosen from a different range. When the capacity range increases, the distribution curve moves to

Figure 4-7. Path length distribution in CAM-Chord. Legend “[$x..y$]” means the node capacities are uniformly distributed in the range of [$x..y$].

Figure 4-8. Path length distribution in CAM-Koorde. Legend “[$x..y$]” means the node capacities are uniformly distributed in the range of [$x..y$].

Figure 4-9. Average path length with respect to average node capacity

the left of the plot due to shorter multicast paths. The improvement in the distribution can be measured by how much the curve is shifted to the left. At the beginning, a small increase in the capacity range causes significant improvement in the distribution. After the capacity range reaches a certain level ($[4, 10]$ in our simulations), the improvement slows down drastically.

Each curve has a single peak, and the right side of the peak quickly decreases to zero. It means that the vast majority of nodes are reached within a small range of path lengths. We didn't observe any multicast path whose length was significantly larger than the average path length.

4.6.4 Average Path Length

Figure 4-9 shows the average path length with respect to the average node capacity. We also plot an artificial line, $1.5\frac{\ln n}{\ln c}$ with $n = 10^5$, which upper-bounds the average path lengths of CAM-Chord and CAM-Koorde, verifying Theorem 4.6 and Theorem 4.9.

From the figure, when the average node capacity is less than 10, the average path length of CAM-Chord is smaller than that of CAM-Koorde; when the average node capacity is greater than 12, the average path length of CAM-Koorde is smaller than CAM-Chord. A smaller average path length means more balanced multicast trees. For small node capacities, CAM-Chord multicast trees are more balanced than CAM-Koorde

Figure 4-10. Proximity optimization

multicast trees, and vice versa. The reasons are explained as follows. On one hand, a non-leaf CAM-Koorde node x may have less children than c_x because some neighbors may have already received the multicast message from a different path. This tends to make the depth of a CAM-Koorde multicast tree larger than that of a CAM-Chord tree. On the other hand, a CAM-Chord node x may split $(x, k]$ into uneven subsegments (i.e., subtrees) with a ratio up to c_x (Lines 6-15 in Section 4.3.4). This ratio of unevenness becomes small when the node capacities are small. Combining these two factors, CAM-Chord creates better balanced trees for small node capacities; CAM-Koorde creates better balanced trees for large node capacities.

Next we use CAM-Chord as an exmple (Section 4.5.2) to study the impact of proximity optimization. In [65], Mukherjee found that the end-to-end packet delay on the Internet can be modeled by a shifted Gamma distribution, which is a long-tail distribution. The shape parameter varies from approximately 1.0 during low loads to 6.0 during high loads on the backbone. We set the shape parameter to be 5.0 and the average packet delay to be 50 ms. Figure 4-10 compares the average latency of delivering a multicast message from a source to a receiver in CAM-Chord with or without the proximity optimization. The simulation is performed for different average node capacities,

Figure 4-11. Throughput vs. latency

and the impact of proximity optimization is significant. In most cases, it reduces the latency more than by half.

4.6.5 Impact of Dynamic Capacity Variation

In a real environment, the upload bandwidth of a node may fluctuate. If we always use the same implicit multicast trees, then the dynamic variation of node capacities will cause variation in average throughput but not in average latency. CAMs can also be easily modified to ensure throughput but allow latency variation. If a node's capacity decreases, it simply forwards messages to a smaller number of neighbors, which automatically reshapes the implicit tree. If a node's capacity increases for a long time, the node can take advantage of the improved capacity by increasing the number of neighbors.

We define the capacity ratio as the actual capacity of a node x at the real time divided by the claimed capacity c_x that is used to build the topology of CAMs. We define the latency ratio as the actual delay at a given capacity ratio divided by the "benchmark" delay when the capacity ratio is 100%, namely, no dynamic capacity variation. Apparently, the latency ratio is a function of the capacity ratio.

Figure 4-11 shows the relation between the latency ratio and the capacity ratio. When the capacity ratio is smaller, which means the nodes cannot support as many children as they have claimed, the nodes will forward the received messages to a less number of neighbors such that each child will still receive a sufficient amount of bandwidth. But the height of the implicit multicast tree will be larger, which means a larger latency ratio. As show in the figure, when the average capacity becomes 50% of the claimed one, the average latency is increased by 47% and 42% for CAM-Chord and CAM-Koorde, respectively. It shows that the dynamic capacity variation will cause the latency variation (instead of throughput variation).

CHAPTER 5
SUMMARY

In the first part of this dissertation, a hybrid query scheme for unstructured P2P networks that mixing inter-cluster queries and intra-cluster queries is proposed. It achieves a better tradeoff among response time and network diameter of the P2P network. A totally distributed clustering algorithm and a labeling algorithm are also presented. The algorithm gives P2P networks better resilience to network dynamics. It can potentially be used in other scenarios when centralized clustering is not possible. The performance of the algorithms is demonstrated by simulations.

In the second part, we have proposed a new incentive scheme suitable to the network without any centralized authority: MARCH, in which the contribution of a node is evaluated by the quantity and quality of service the node has provided. We also have designed a distributed authority infrastructure and a set of protocols to implement the scheme with very small overhead. MARCH is more resilient than conventional approaches, in which selfish/malicious nodes cannot benefit by breaking the protocols, and are punished automatically. A collusion, regardless of its size, cannot increase its damage. Consequently, malicious nodes will be finally rejected from the system or enforced to contribute to the system in order to keep their available money and reputation above a certain level.

In the third part, we have proposed two overlay multicast services, called CAM-Chord and CAM-Koorde, which are capacity-aware extensions of Chord and Koorde with multicast routines that follow implicit, well-balanced trees to disseminate multicast messages. One attractive property is that the number of multicast children of a node is bounded by its capacity, which may vary widely among the nodes. It prevents the multicast throughput from degrading due to the overload of low-capacity nodes. With each source node having a separate, implicit multicast tree, the overall traffic is well balanced across the network.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking (TON)*, 2003.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," *Proc. of ACM SIGCOMM '01*, August 2001.

[3] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Global-scale Overlay for Rapid Service Deployment," *IEEE J-SAC*, January 2004.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proc. of Middleware '01*, November 2001.

[5] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, 1999.

[6] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," *Proc. of ACM PODC '02*, pp. 183–192, 2002.

[7] A. Kumar, S. Merugu, J. Xu, and X. Yu, "Ulysses: A Robust, Low-Diameter, Low-Latency Peer-to-Peer Network," *Proc. of IEEE ICNP '03*, Nov. 2003.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," *Proc. of ACM SIGCOMM '03*, pp. 407–418, 2003.

[9] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. of IEEE INFOCOM '04*, Mar. 2004.

[10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. of ICS '02*, pp. 84–95, Sep. 2002.

[11] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. of IEEE INFOCOM '03*, Mar. 2003.

[12] C. Dellarocas, "Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior," *EC '00: Proc. of the 2nd ACM conference on Electronic commerce*, 2000.

[13] M. Srivatsa, L. Xiong, and L. Liu, "TrustGuard: Countering Vulnerabilities in Reputation Management for Decentralized Networks," *Proc. of WWW'05*, May 2005.

[14] P. Dewan and P. Dasgupta, "Securing Reputation Data in Peer-to-Peer Networks," *Proc. of Parallel and Distributed Computing and Systems*, 2004.

[15] Y. Wang and J. Vassileva, “Bayesian Network Trust Model in Peer-to-Peer Networks,” *Proc. of AP2PC ’03*, 2003.

[16] M. Venkatraman, B. Yu, and M. P. Singh, “Trust and Reputation Management in a Small-World Network,” *ICMAS ’00: Proc. of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, 2000.

[17] S. Marti and H. Garcia-Molina, “Identity Crisis: Anonymity vs. Reputation in P2P Systems,” *Third IEEE International Conference on Peer-to-Peer Computing*, 2003.

[18] M. Jakobsson, J. Hubaux, and L. Buttyan, “A Micropayment Scheme Encouraging Collaboration in Multi-hop Cellular Networks,” *Proc. of Financial Crypto ’03*, 2003.

[19] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “KARMA: A Secure Economic Framework for P2P Resource Sharing,” *Proc. of the Workshop on the Economics of Peer-to-Peer Systems*, June 2003.

[20] Y. Zhang, W. Lou, and Y. Fang, “SIP: A Secure Incentive Protocol against Selfishness in Mobile Ad hoc Networks,” *Proc of WCNC ’04*, March 2004.

[21] J. C. L. T.B. Ma, Sam C.M. Lee and D. K. Yau, “A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks,” *Proc. of ACM SIGMETRICS/PERFORMANCE*, June 2004.

[22] L. Buttyn, “Removing the Financial Incentive to Cheat in Micropayment Schemes,” *IEE Electronics Letters*, pp. 132–133, January 2002.

[23] S. Lee, R. Sherwood, and B. Bhattacharjee, “Cooperative Peer Groups in NICE,” *Proc. of IEEE INFOCOM ’03*, Apr 2003.

[24] M. Feldman, K. Lai, I. Stoica, and J. Chuang, “Robust Incentive Techniques for Peer-to-Peer Networks,” *ACM Electronic Commerce*, 2004.

[25] G. Banavar, M. Chandra, B. Nagarajaro, R. Strom, and C. Sturman, “An Efficient Multicast Protocol for Content-based Publish-Subscribe System,” *Proc. of ICDCS ’98*, May 1998.

[26] Y. H. Chu, S. Rao, and H. Zhang, “A Case for End System Multicast,” *Proc. of SIGMETRICS ’00*, June 2000.

[27] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O’Toole, “Overcast: Reliable Multicasting with an Overlay Network,” *Proc. of OSDI ’00*, Oct 2000.

[28] C. K. S.Banerjee, B.Bhattacharjee, “Scalable Application Layer Multicast,” *Proc. of ACM SIGCOMM ’02*, August 2002.

[29] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, “An Architecture for Wide-Area Multicast Routing,” *Proc. of ACM SIGCOMM ’94*, pp. 126–135, August 1994.

[30] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, “Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture,” *Proc. of ACM SIGCOMM '01*, Auguest 2001.

[31] B. Zhang, S. Jamin, and L. Zhang, “Host Multicast: A Framework for Delivering Multicast to End Users,” *Proc. of IEEE INFOCOM '02*, June 2002.

[32] G.-I. Kwon and J. W. Byers, “ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections,” *Proc. of IEEE INFOCOM '04*, March 2004.

[33] P. M. Zhi Li, “Impact of Topology On Overlay Routing Service,” *Proc. of IEEE INFOCOM '04*, March 2004.

[34] Y. Shavitt and T. Tankel, “On the Curvature of the Internet and Its Usage for Overlay Construction and Distance Estimation,” *Proc. of IEEE INFOCOM '04*, March 2004.

[35] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, and S. Sahu, “Scalability of Reliable Group Communication Using Overlays,” *Proc. of IEEE INFOCOM '04*, March 2004.

[36] V. Pappas, B. Zhang, A. Terzis, and L. Zhang, “Fault-Tolerant Data Delivery for Multicast Overlay Networks,” *Proc. of ICDCS '04*, March 2004.

[37] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, “Bayeux: An Architecture for Scalable and Fault-tolerant WideArea Data Dissemination,” *Proc. of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '01)*, June 2001.

[38] R. Zhang and Y. C. Hu, “Borg: a Hybrid Protocol for Scalable Application-level Multicast in Peer-to-Peer Networks,” *Proc. of NOSSDAV '03*, 2003.

[39] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Application-Level Multicast Using Content-Addressable Networks,” *Proc. of NGC '01*, 2001.

[40] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, “Efficient Broadcast in Structured P2P Networks,” *Proc. of IPTPS '03*, Feb 2003.

[41] M. Castro, M. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, “An Evaluation of Scalable Application-Level Multicast Built Using Peer-To-Peer Overlays,” *Proc. of IEEE INFOCOM '03*, April 2003.

[42] S. Shi, J. Turner, and M. Waldvogel, “Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks,” *Proc. of NOSSDAV '01*, June 2001.

[43] S. Shi and J. Turner, “Routing in Overlay Multicast Networks,” *Proc. of IEEE INFOCOM '02*, June 2002.

[44] J. A. Dejan Kosti, Adolfo Rodriguez and A. Vahdat, “Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh,” *Proc. of SOSP '03*, October 2003.

[45] S. Banerjee, C. Kommareddy, B. B. K. Kar, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," *Proc. of IEEE INFOCOM '03*, March 2003.

[46] A. Riabov and L. Z. Zhen Liu, "Overlay Multicast Trees of Minimal Delay," *Proc. of ICDCS '04*, March 2004.

[47] H. Yamaguchi, A. Hiromori, T. Higashino, and K. Taniguchi, "An Autonomous and Decentralized Protocol for Delay Sensitive Overlay Multicast Tree," *Proc. of ICDCS '04*, March 2004.

[48] Z. Zhang, Y. Tang, S. Chen, and Y. Jian, "A Hybrid Query Scheme to Speed Up Queries in Unstructured Peer-to-Peer Networks," *Advances in Multimeida, Special Issue on Towards the Next Generation Peer-to-Peer Services*, vol. 2007, 2007.

[49] Z. Zhang, S. Chen, and M. Yoon, "MARCH: A Distributed Incentive Scheme for Peer-to-Peer Networks," *Proc. of IEEE INFOCOM*, pp. 1955–1963, 2007.

[50] Z. Zhang, S. Chen, Z. Mo, and M. Yoon, "An Efficient Incentive Scheme with a Distributed Authority Infrastructure in Peer-to-Peer Networks," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, December 2012.

[51] Z. Zhang, S. Chen, Y. Ling, and R. Chow, "Resilient Capacity-Aware Multicast Based on Overlay Networks," *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 565–574, 2005.

[52] ——, "Capacity-Aware Multicast Algorithms on Heterogeneous Overlay Networks," *IEEE Transactions on Parallel and Distributed Systems, Special Issue on Algorithm Design and Scheduling Techniques (Realistic Platform Models)*, vol. 17, no. 2, pp. 135–147, 2006.

[53] N. Chang and M. Liu, "Revisiting the TTL-based controlled Flooding Search: Optimality and Randomization," *Proc. of ACM MobiCom '04*, 2004.

[54] ——, "Controlled Flooding Search in a Large Network," *Proc. of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2005.

[55] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," *Proc. of IEEE INFOCOM '05.*, 2005.

[56] A. Iamnitchi, M. Ripeanu, and I. Foster, "Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations," *Proc. of IPTPS '02: 1st International Workshop on Peer-to-Peer Systems*, 2002.

[57] ——, "Small-world Filesharing Communities," *Proc. of IEEE INFOCOM '04.*, 2004.

[58] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach, "Secure Routing for Structured Peer-to-Peer Overlay Networks," *Proc. of OSDI '02*, 2002.

[59] J. R. Douceur, “The Sybil Attack,” *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pp. 251–260, 2002.

[60] Y. G. Desmedt and Y. Frankel, “Threshold Cryptosystems,” *CRYPTO ’89: Proceedings on Advances in cryptology*, pp. 307–315, 1989.

[61] M. Kaashoek and D. Karger, “Koorde: A Simple Degree-optimal Distributed Hash Table,” *Proc. of 2nd IPTPS, Berkeley, CA*, Feb 2003.

[62] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The Impact of DHT Routing Geometry on Resilience and Proximity,” *Proc. of ACM SIGCOMM ’03*, August 2003.

[63] S. Sen and J. Wong, “Analyzing Peer-to-Peer Traffic across Large Networks,” *Proc. of Second Annual ACM Internet Measurement Workshop*, November 2002.

[64] S. Ratnasamy, “Routing Algorithms for DHTs: Some Open Questions,” *Proc. of 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[65] A. Mukherjee, “On the Dynamics and Significance of Low Frequency Components of Internet Load,” *Internetworking: Research and Experience*, vol. 5, no. 4, pp. 163–205, 1994.

BIOGRAPHICAL SKETCH

Zhan Zhang received his M.S. degree in computer science from Fudan University of China in 2003, and received his doctorate in Computer and Information Science and Engineering from University of Florida in 2007. His current research fields include overlay networks, network security and sensor networks. His email address is zzhan@cise.ufl.edu.