Resilient Capacity-Aware Multicast Based on Overlay Networks

Zhan Zhang† Shigang Chen† Yibei Ling‡ Randy Chow †
† Dept. of Computer & Information Science & Engineering, University of Florida
‡ Applied Research Laboratories, Telcordia Technologies

Abstract

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. Many overlay multicast systems have been proposed in recent years. However, they are insufficient in supporting applications that require large-scale any-source multicast with highly varied host capacities and highly dynamic membership. In this paper, we propose two capacity-aware multicast systems that focus on host heterogeneity, dynamic membership, scalability, and any source multicast. 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 also perform extensive simulations to evaluate the proposed multicast systems.

1 Introduction

Multicast is an important function for efficient information exchange among a distributed, dynamic cluster of heterogeneous nodes. IP Multicast [1, 2] was proposed as an extension of the Internet datagram service to support group-oriented communication or multi-point packet delivery at the network level. With IP Multicast, the data flow from a source node is delivered efficiently to all interested receivers through a distribution tree. The global deployment of IP multicast has been slow due to the difficulties related to heterogeneity, scalability, manageability, lack of a robust inter-domain multicast routing protocol, and so forth.

In recent years, many research papers (e.g., [3, 4, 5, 6, 7, 8, 9]) pointed out the disadvantages of implementing multicast at the IP level, and argued for an application-level overlay multicast service. More recent work (e.g., [10, 11, 12, 13, 14, 15, 16]) studied overlay multicast from different aspects.

There are four important design issues to be addressed in an overlay multicast system.

• *host heterogeneity:* Member hosts may vary widely in their capacities in terms of CPU power, memory, or network bandwidth. Some may be able to support a large number of direct children, but others may only support few.

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

• *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.

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

None of the existing systems meet the above four requirements.

To handle dynamic groups and ensure scalability, novel proposals were made to implement multicast on top of P2P overlay networks. For example, Bayeux [17] and Borg [18] were implemented on top of Tapestry [19] and Pastry [20] respectively, and CAN-based Multicast [21] was implemented based on CAN [22]. The scheme of overlay multicasting can be extended to other structured peer-to-peer networks. El-Ansary et al. studied efficient broadcast in a Chord network, and their approach can be adapted for the purpose of multicast [23]. Castro et al. compares the performance of tree-based and flooding-based multicast in CAN-style versus Pastry-style overlay networks [24].

Host heterogeneity is not addressed in the above systems that assume each node have the same number of children. While overlay multicast can be implemented on top of overlay unicast, they also have very different requirements. In overlay unicast, low-capacity nodes only affect traffic passing through them and therefore they create bottlenecks of limited impact. In overlay multicast, all traffic will pass all nodes in the group. The multicast throughput is decided by the node of 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 high, the low-capacity nodes will be overloaded, which slows the entire session down. If the number of children is set too low, 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 [25]. Centralized heuristic algorithms were proposed to balance multicast traffic among multicast service nodes (MSNs) and to maintain low end-to-end latency [25, 26]. The algorithms do not address the dynamic membership problem, i.e., MSN join/departure. Overlay Multicast Network Infrastructure (OMNI) [27] dynamically adapts its multicast tree to minimize the latencies to the entire client set. It is designed for a single source and therefore not suitable for interactive multicast applications such as distributed games. Riabov et al. proposed a constant-factor approximation algorithm for the problem of constructing a single-source degree-constrained minimum-delay multicast tree [28]. It is not a distributed algorithm, which is the focus of this paper. Yamaguchi et al. described a distributed algorithm that maintains a degree-constrained multicast tree for a dynamic group [29]. The single-tree approach leaves the capacities of the majority nodes (leafs) unused, which affects the overall throughput in multi-source multicasting. In addition, a single tree may be partitioned beyond repair for a highly dynamic group.

Given the above limitations, the existing multicast algorithms are not sufficient in supporting distributed applications that require large-scale any-source multicast with highly varied host capacities and highly dynamic membership. This paper proposes two capacity-aware multicast systems that satisfy all four requirements discussed previously. We model the capacity as the maximum number of direct children that a node is willing to forward multicast messages. We extend Chord [30] and Koorde [31] to be capacity-aware and they are called CAM-Chord and CAM-Koorde, respectively.[1] A dedicated 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. Host heterogeneity and capacity-aware multicast are added features.

The rest of the paper is organized as follows. Section 2 gives an overview of our proposed systems. Section 3 and Section 4 describe CAM-Chord and CAM-Koorde in details, respectively. Section 5 discusses some related issues. Section 6 presents the the simulation results. Section 7 draws the conclusion.

[1] CAM stands for Capacity-Aware Multicast.

2 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 heterogenous 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. No prior work possesses all these features.

Our basic idea is to build a multicast service on top of a *capacity-aware* stuctured P2P network. We focus on extending Chord [30] and Koorde [31] for such a service. The resulting systems are called CAM-Chord and CAM-Koorde, respectively. The principles and techniques developed in this paper 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 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) 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, the notations, 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 $|x-y| = |y-x| = \min\{(y-x), (x-y)\}$.

Before we overview the proposed CAMs, we briefly review Chord and Koorde. Each node x in Chord has $O(\log_2 n)$ neighbors, which are responsible for identifiers $(x+2^i) \bmod N, \forall i \in [1..\log_2 N]$. It takes $O(\log_2 n)$ hops with high probability to find a node responsible for any given identifier. Koorde forms a de Brujin graph amongst

the nodes. With k neighbors per node, it finds a node responsible for any given identifers in $O(\log_k n)$ hops with high probability. Readers are referred to the original papers for more details.

Our first system is CAM-Chord. Different from Chord, the number of neighbors of a node in CAM-Chord is related to the node's capacity. Not all nodes have the same number of neighbors. In particular, each node x maintains $O(c_x \frac{\log n}{\log c_x})$ neighbors that are responsible for the following identifiers, $(x + j \times c_x^i) \bmod N$, $\forall j \in [1..c_x - 1]$, $\forall i \in [0..\frac{\log N}{\log c_x} - 1]$.[2] Node x selects c_x neighbors as its direct children in the multicast tree and each of the c_x neighbors handles a subtree of similar size. The robustness of the system comes from the maintenance protocol of Chord.

The second system is CAM-Koorde, which differs from Koorde in both the number of neighbors and how the neighbors are calculated. The difference is critical in constructing balanced multicast trees. Again, nodes have different numbers of neighbors if their capacities are different. Each node x has c_x neighbors, and the neighbor identifiers are derived by shifting x one or more bits to the right and then replacing the highest bits with a certain number. In comparison, Koorde shifts x one digit (base k) to the left and replaces the lowest 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.

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 [32]. The two systems achieve their best performances under different conditions.

• If node capacities are small, CAM-Koorde is not resilient against frequent membership changes. The overlay network may even be partitioned as nodes depart. CAM-Chord is a better choice in such an environment because of denser connectivity. In addition, our simulations in Section 6 show that CAM-Chord has much shorter delivery paths than CAM-Koorde when node capacities are small.

• If node capacities are large, the maintenance overhead of CAM-Chord is significant. Meanwhile the robustness of CAM-Koorde is improved with an increased number of neighbors. Furthermore, when the node capacities reach certain level, the average path lengths of CAM-Chord and CAM-Koorde become more or less the same, as demonstrated by our simulations. Hence, CAM-Koorde becomes a better choice.

[2]There is a disparity between the number of neighbors, $c_x \frac{\log n}{\log c_x}$, and the number of identifiers, $c_x \frac{\log N}{\log c_x}$. That is because some neighbors will be responsible for more than one of those identifiers, similar to the same disparity in the original Chord.

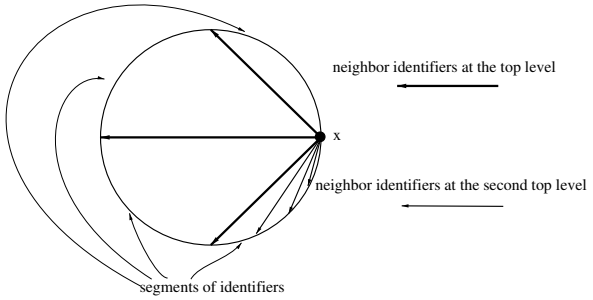

Figure 1. Neighbor identifiers at the top two levels

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 the lookup routine, and then present our multicast algorithm on top of CAM-Chord.

The members of a multicast group use the lookup routine to join and leave the overlay topology. CAM-Chord is not designed for data sharing among peers as most other P2P networks (e.g., Chord [30]) do. There are NO data items associated with the identifier space. Each multicast group forms its own CAM-Chord network. The CAM-Chord overlay provides the foundation for *dynamic capacity-aware* multicasting.

3.1 Neighbors

Instead of maintaining $O(\log n)$ neighbors as Chord does, each node x in CAM-Chord maintains $O(c_x \frac{\log n}{\log c_x})$ neighbors, which are nodes responsible for the identifiers $(x + j \times c_x^i) \bmod N$, 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 $x_{i,j}$. The top two levels of neighbor identifiers are illustrated in Figure 1. There are $(c_x - 1)$ neighbor identifiers at each level. The higher the level is, the wider the neighbor identifiers are separated. Let $x_{0,0} = x$.

Figure 2 illustrates an 8-node example of the CAM-Chord network. The neighbors of node x are shown by arrows. The name space is $[0..31]$, and the capacity of x is 3. $x_{0,1} = x + 1$, $x_{0,2} = x + 2$, and $x_{1,1} = x + 3$. Node $x+4$ are responsible for these identifiers. It means that $\widehat{x_{0,1}}$, $\widehat{x_{0,2}}$, and $\widehat{x_{1,1}}$ refer to the same node $x + 4$. Similarly, $\widehat{x_{1,2}}$ refers to node $x + 8$, $\widehat{x_{2,1}}$ refers to node $x + 13$, $\widehat{x_{2,2}}$ refers to node $x + 18$, and $\widehat{x_{3,1}}$ refers to node $x + 29$.

Consider an arbitrary identifier k. Let

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

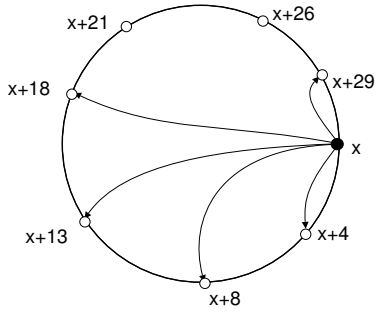

Figure 2. Neighbors of x. $N = [0..31]$. $c_x = 3$.

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

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 $\widehat{k}$.[3] We call i the *level* and j the sequence number of k with respect to x.

3.2 Lookup Routine

CAM-Chord requires a lookup routine in order to support group management for 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 (predecessor(x), x]. The set of identifiers that successor(x) is responsible for is (x, successor(x)]. An identifier region (x, y] starts from x, moves clockwise, and ends at y.

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**
7. **return** the address of $\widehat{x_{i,j}}$
8. **else**
/* forward request to next hop $x_{i,j}$ */
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 contacts $\widehat{x_{i,j}}$ to handle the request, which makes a greedy progress to move the request closer to k because $\widehat{x_{i,j}}$ is x's closest neighbor preceding k.

As an example, consider the CAM-Chord ring in Figure 2. Suppose node x wants to find the address of the node that is responsible for identifier $x + 25$. The level and the sequence number of identifier $x + 25$ are both 2 with respect to x. Recall that $\widehat{x_{2,2}}$ refers to node $x + 18$ and it precedes $x + 25$. By Line 9, the request is forwarded to the node $(x + 18)$. Node $x + 18$ repeats the above process by executing $(x + 18)$.LOOKUP(k). Suppose the capacity of node $(x + 18)$ is also 3. The level and the sequence number of identifier $x + 25$ are 1 and 2 with respect to $x + 18$. Because $\widehat{(x+18)}_{1,2}$ refers to the node $x + 26$ and $x + 25 \in (x + 18,\ x + 26]$, by Lines 6-7, the address of node $x + 26$ is returned.

Due to space limitation, we omit the proof for all theorems.

Theorem 1 *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 $O(-\frac{\ln n}{\ln E(\frac{\ln c_x}{c_x})})$.*

Theorem 2 *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 $O(\frac{\log n}{\log c})$.*

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 only difference is that our LOOKUP routine replaces the Chord LOOKUP routine and the rest of the Chord protocols stay the same. The details of the protocols can be found in [30].

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. The outdegree of each intermediate node in a tree does not exceed its capacity. It should be emphasized that no explicit tree is built. Given a multicast message, the source x executes a MULTICAST routine, which sends the message to c_x selected neighbors. Upon receipt of a multicast message, a node will also executes the same MULTICAST routine and the message are propagated to

[3] It is possible that $\widehat{x_{i,j}} = \widehat{k}$ if there is not a node between $x_{i,j}$ and k on the ring.

more nodes. The collective 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 and k indicate an identifier region $(x, k]$. The goal of x.MULTICAST(msg, k) is to deliver msg to all nodes in $(x, k]$. The basic idea is described as follows: x chooses c_x child neighbors that split $(x, k]$ into c_x subregions as even as possible, as illustrated in Figure 1. Each subregion begins from one chosen neighbor and ends with the next clockwise chosen neighbor. x sends each chosen neighbor a control message, which carries the message as well as the subregion assigned to this neighbor. After the neighbor receives the message, it forwards the message using the same method recursively until the size of the subregion is reduced to zero. The process divides $(x, k]$ into non-overlapping subregions and hence no node will receive the message twice. The key is how to select the child neighbors.

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 $\widehat{x_{i,m}}$.

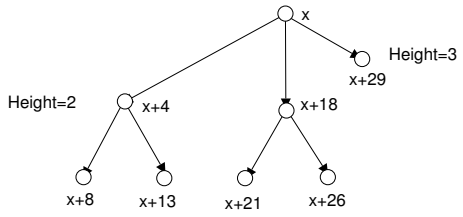

Figure 3. Multicast Spanning Tree

Consider the CAM-Chord topology in Figure 2. According to the algorithm of MULTICAST, the implicit multicast tree rooted at x is shown in Figure 3. A source node x initiates the delivery of a multicast message msg by calling x.MULTICAST$(x - 1, msg)$, whose distributed, recursive execution makes sure that every member node will receive one and only one copy of the message. By Line 4-5, the level and the sequence number of $x - 1$ is 3 and 1 respectively. By Line 6-9, x forwards msg to $\widehat{x_{3,1}}$ (node $x + 29$) with assigned segment $(x + 29,\ x + 31]$. By Line 10-14, x forwards msg to $\widehat{x_{2,2}}$ (node $x + 18$) with assigned segment $(x + 18,\ x + 26]$. By Line 15, x forwards msg to $\widehat{x_{0,1}}$ (node $x + 4$) with assigned segment$(x + 4,\ x + 17]$. After $(x + 18)$ receives the message, the execution of $(x + 18)$.MULTICAST$(x + 26,\ msg)$ will forward the message to nodes $(x + 21)$ and $(x + 26)$. Similarly, $(x + 4)$.MULTICAST$(x+17,\ msg)$ will forward the message to nodes $(x + 8)$ and $(x + 13)$.

Because each multicast group forms its own overlay network, multicast is implemented as broadcast. The broadcast in Chord has been studied in [23]. Given a message, the source node forwards it to all $M = O(\log n)$ neighbors, and each neighbor x is responsible for delivering the message to the identifier segment (x, y), where y is the next neighbor after x. More specifically, x forwards the message only to its neighbors in (x, y), and these neighbors are then responsible for delivering the message to smaller segments in a similar way. This process repeats until all nodes receive a copy of the message. Because the nodal degree and the tree depth are both $O(\log n)$, the multicast tree is not optimal. Consider the first hop from the root to its neighbors. Because the size of (x, y) ranges from $N/2$ to $N/2^M$, the depths of the root's subtrees range from $O(\log n)$ to $O(1)$. Therefore, the whole multicast tree is not balanced. Moreover, any subtree is not balanced due to the same reason that the spacing between Chord neighbors varies exponentially. At the hth level of the multicast tree, the number of children per node ranges from 1 to $(M - h)$ [23], independent of the node's capacity.

Our multicast algorithm is similar to the algorithm of [23] but has differences to make the multicast trees better

balanced and capacity-aware. First of all, in a CAM-Chord multicast tree, the number of children for an internal node is always equal to the node's capacity as long as the node is not at the bottom levels of the tree where there are not enough child nodes to fill up the capacities of parent nodes. Second, if the capacities of all nodes are the same, our algorithm still performs differently than [23]. It balances the tree by making sure that the number of children per node does not vary from subtree to subtree or from level to level as long as the node is not at the bottom levels of the tree. The tree depth is $O(\frac{\log n}{\log c})$ if the nodes have the same capacity c. This property is due to Lines 6-14, which choose the children of a node such that they are as evenly spaced as possible. The cost for capacity awareness and better balanced trees is that each node has more neighbors than Chord.

Theorem 3 *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 *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 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 numbr 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 k) to the left and then replacing the last digit with 0 through k. 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 therefore 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.

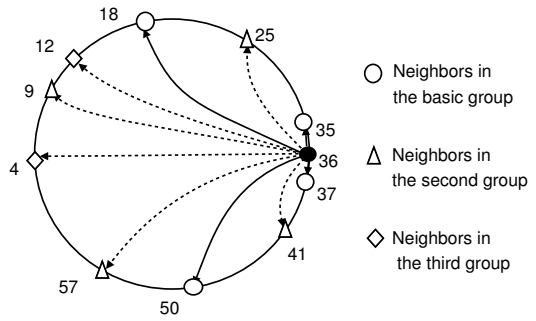

Figure 4. CAM-Koorde topology with identifier space [0..63]

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.

• 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.[4] 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, 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

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

The third group is

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

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

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)

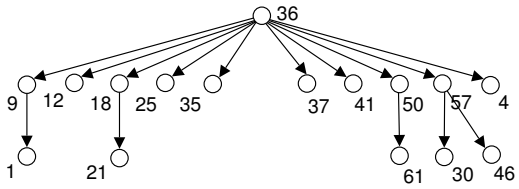

Figure 5. Multicast Spanning Tree

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

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, e.g., 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)

We give an example based on the CAM-Koorde topology in Figure 4. For simplicity, assume the node capacities are all 10. An implicit multicast tree is shown in Figure 5, which is rooted at node 36. When the source node 36 initiates the delivery of a multicast message msg by calling MULTICAST(msg), it forwards the message to all its neighbors, nodes 9,12,18,25,35,37,41,50,57, and 4. The neighbors then forward the received message to their neighbors that have not received the messages, forming an implicit multicast tree.

Theorem 5 *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 $O(\log n / E(\log c_x))$.

Theorem 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-Koorde from a source node to a member node is* $O(\log n / \log c)$.

5 Discussion

5.1 Tree building versus flooding

To accomplish application-level multicast based on structured P2P networks, there are two general approaches: *tree building* and *flooding*, categorized by [24]. In the approach of tree building, nodes from different multicast groups participate in a single overlay network, and each group forms a multicast tree on top of the overlay network by using reverse path forwarding. In the approach of flooding, each group forms its own overlay network, which transforms multicast to broadcast within the overlay. We call the single overlay network in the tree-building approach as *global overlay*, in order to distinguish it from the *per-group ovelays* in the flooding approach.

Castro et al. performed extensive simulations to evaluate the two approaches by using both Pastry and CAN [24]. The paper concluded that building per-group overlays incurs significantly more overhead than building per-group trees on a global overlay. Therefore, the tree-building approach is a better choice in general.

On the other hand, there also exist several common situations where the flooding approach is preferred. First, as pointed out in [24], due to administrative reasons, it may be undesired for external nodes to relay the multicast traffic. Furthermore, the tree-building approach needs to maintain the multicast tree not only when a group member changes but also when a non-group member that is in the tree leaves the overlay.

Second, the end users may not always stay in the global overlay in anticipation for possible future multicast communication. The observations in [33] showed that over 20% of the connections last 1 minute or less and 60% of the IP addresses keep active in the FastTrack P2P system for no more than 10 minutes each time after they join the system. If the common pattern of end users is to launch an overlay multicast software (possibly via a browser) before joining a multicast group and quit the software when leaving the group, then the overhead saving by a global overlay may be jeopardized because ad-hoc users have to join the global overlay anyway before joining multicast trees and the global overlay has a much larger size than the per-group overlays.

Third, the flooding approach distributes forwarding workload more evenly over all group members. In the tree-building approach, there is one multicast tree per group. The multicast messages from group members travel to the root first and then disseminate to all other nodes. An internal node in the tree forwards every message, while a leaf node never forwards a message. Let k be the average number of children and M be the total size of all messages. If $k > 2$, then the majority of nodes will be leaves. The average forwarding load of an internal node is $O(kM)$; the forwarding load of a leaf node is zero. In the flooding approach, there is one implicit tree per node. Each node serves as an internal node in some implicit trees and a leaf in other trees. Every node in the group receives M traffic. Suppose there are n nodes. The total volume of forwarding load is therefore nM, and this load is spread among all nodes. If the traffic sources are well distributed, most nodes are likely to have comparable forwarding load, which is $O(nM/n) = O(M)$.

The proposed CAMs fall in the flooding category. We are currently investigating the capacity-aware multicast problem following the tree-building approach. An observation is that the multicast tree is constrained by the node capacities but the global overlay is not. How to match the disparity raises some interesting design issues.

5.2 Latency 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 [34, 32] for details.

Extension to the existing P2P networks, CAMs can naturally inherit most of those features without much additional effort. Take CAM-Chord as an example. Although the set of neighbors is fixed in our description, nodes actually can have some freedom in choosing their neighbors. 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 $x_{i,j}$.[5] Given this freedom, some heuristics (e.g., least delay first) may be used to choose neighbors to promote geograhpic clustering.

[5]The lookup and multicast routines need to be modified superficially.

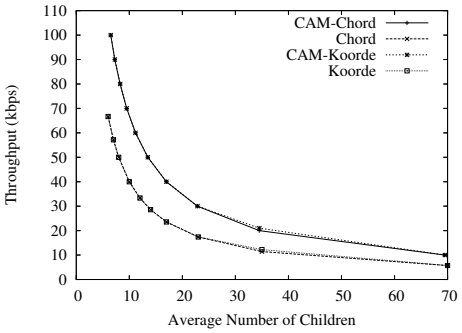

Figure 6. Multicast throughput with respect to average number of children per non-leaf node

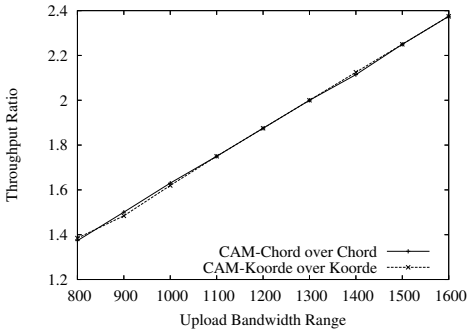

Figure 7. Throughput improvement ratio with respect to upload bandwidth range

6 Simulation

Throughput and latency are the 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 size of a multicast group (and thus the size of the overlay network) is 100,000, and the node capacities are taken from $[4..10]$ with uniform probability. The upload bandwidth of nodes are randomly distributed in a default range of [400,1000] kbps. In our simulation, $c_x = \lfloor B_x/p \rfloor$, 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 capacity, which means different average number of children per non-leaf node and consequently different tree depth (latency).

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 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 the 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 6 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 a rather narrow range of [400, 1000] kbps. In general, let [a, b] be the range of upload bandwidth. The upper bound b of the range is shown on the horizontal axis of Figure 7, 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 $\frac{a+b}{2a}$ (which can be regarded as a measurement for the degree of bandwidth heterogeneity).

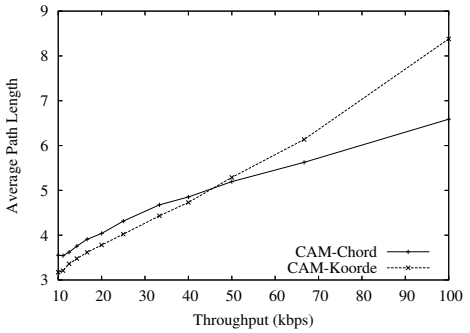

Figure 8. Throughput vs. average path length

6.2 Throughput vs. Latency

We measure multicast latency by the average length of multicast paths. 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

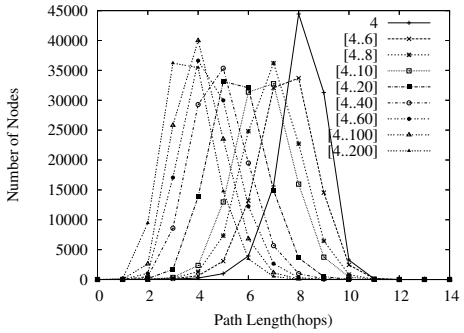

Figure 9. Path length distribution in CAM-Chord. Legend "[x..y]" means the node capacities are uniformly distributed in the range of [x..y].

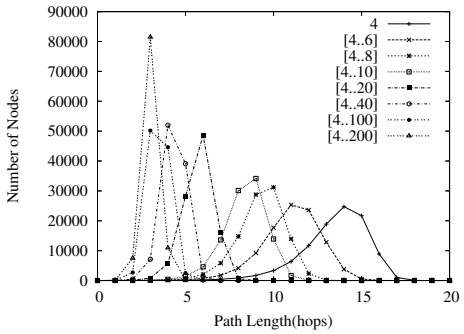

Figure 10. Path length distribution in CAM-Koorde. Legend "[x..y]" means the node capacities are uniformly distributed in the range of [x..y].

decreases due to more children per non-leaf node and the latency also decreases due to smaller tree depth. Therefore, there exists a tradeoff between throughput and latency, which is depicted by Figure 8. Higher throughput can be achieved at the cost of larger latency. 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 6.4.

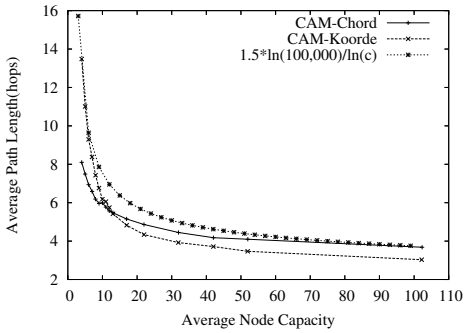

Figure 11. Average path length with respect to average node capacity

6.3 Path Length Distribution

Figure 9 and Figure 10 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 increase, the distribution curve moves to 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.

6.4 Average Path Length

Figure 11 shows the average path length with respect to the average node capacity. We also plot an artificial line, $\frac{1.5 \log n}{logc}$ with $n = 10^5$, which upper-bounds the average path lengths of CAM-Chord and CAM-Koorde, verifying Theorem 4 and Theorem 6. 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. Therefore, for small node capacities, CAM-Chord multicast trees are more balanced than CAM-Koorde multicast trees, and vice versa. The reasons are explained as follows. On the 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 3.2). 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.

7 Conclusion

This paper 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 ensures that the multicast throughput is not degraded by overloaded low-capacity nodes. With each source node having a separate, implicit multicast tree, the overall traffic is well balanced across the network.

Based on the simulation results, CAM-Chord and CAM-Koorde achieve their best performances under different conditions, depending on membership change frequency and node capacities. CAM-Chord works better with relatively small frequency of membership change and small node capacities, while CAM-Koorde works better with relatively large frequency of membership change and large node capacities.

References

[1] S. Dering, "Multicast Routing in a Datagram Internetwork ," *Phd thesis,Stanford University*, Dec. 1991.

[2] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An architecture for wide-area multicast routing," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 126–135, 1994.

[3] Francis, "Yoid: Extending the internet Multicast Architecture," *Technical report*, 2000.

[4] Y. Chawathe, S. McCanne, and E. Brewer, "An architecture for internet content distribution as an infrastructure service," *Unpublished work*, 2000.

[5] Y. H. Chu, S. Rao, and H. Zhang, "A case for end system multicast," *Proc. of SIGMETRICS'00*, June 2000.

[6] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, "Overcast: Reliable Multicasting with an overlay network," *Proc. of OSDI'00*, 2000.

[7] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application level Multicast Infrastructure," *In 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[8] 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*, 1998.

[9] C. K. S.Banerjee, B.Bhattacharjee, "Scalable Application Layer Multicast," *Proc. of ACM SIGCOMM'02*, 2002.

[10] 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.

[11] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," *Proc. of IEEE INFOCOM'02*, June 2002.

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

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

[14] 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.

[15] 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.

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

[17] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant WideArea Data Dissemination," *Proc. of NOSSDAV'01*, June 2001.

[18] 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.

[19] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," *Technical Report*, no. UCB/CSD-01-1141, Apr 2001.

[20] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, 2001.

[21] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-Level Multicast Using Content-Addressable Networks," *Lecture Notes in Computer Science*, vol. 2233, 2001.

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

[23] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient Broadcast in Structured P2P Networks," *In IPTPS'03*, Feb 2003.

[24] 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 INFOCOM03*, April 2003.

[25] S. Shi, J. Turner, and M. Waldvogel, "Dimensioning server access bandwidth and multicast routing in overlay networks," *Proc. of NOSSDAV'01*, June 2001.

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

[27] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," *Proc. of IEEE INFOCOM'03*, March 2003.

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

[29] 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.

[30] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications ," *Proc. of ACM SIGCOMM'01*, August 2001.

[31] M. Kaashoek and D. Karger, "Koorde: A simple degree-optimal distributed hash table," *Proc. of 2nd IPTPS, Berkeley, CA*, Feb 2003.

[32] 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.

[33] S. Sen and J. Wong, "Analyzing peer-to-peer traffic across large networks," *Second Annual ACM Internet Measurement Workshop*, November 2002.

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