Z-Ring: Fast Prefix Routing via a Low Maintenance Membership Protocol

Qiao Lian[†] , Wei Chen[†] , Zheng Zhang[†] , Shaomei Wu[§], and Ben Y. Zhao[§], *Member, IEEE*
[†] *Microsoft Research Asia, Beijing, China*
[§]*Computer Science Department, U. C. Santa Barbara, CA, U.S.*
{qiaol, weic, zzhang}@microsoft.com, smwu@mails.tsinghua.edu.cn, ravenben@cs.ucsb.edu

Abstract

In this paper, we introduce Z-Ring, a fast prefix routing protocol for peer-to-peer overlay networks. Z-Ring incorporates cost-efficient membership protocol to achieve fast routing with small maintenance cost. Z-Ring achieves routing in logGN steps, where N is the network size and G is the size of a group that can be maintained by a membership protocol with low cost. With G=4096, it translates to one-hop routing for intranet environments (N<4096), two-hop routing for mid-scale internet applications (N<16 million), and three-hop routing for ultra-large internet applications (N<64 billion). Z-Ring maintains good routing success rate under churn and low maintenance cost even at large network size. Its modularized use of the membership protocol also makes it adaptive to dynamic and wide-range network size changes.

***Index Terms*—Peer-to-peer, DHT, routing, group membership**

1. Introduction

Recent research has shown that structured peer-to-peer overlay networks such as Pastry [13], Chord [15], and Tapestry [17] provide scalable and resilient abstractions to large-scale network applications. They support routing to endpoints or nodes inside a network requiring only logarithmic routing state at each node. In a network of size *N*, they route a message to any destination within *logN* overlay hops, with each node storing *O(logN)* outgoing links to neighbors.

While the logarithmic number of hops scales well with network size, the latency they incur can be substantial in practice. For example, a message still goes through several hops inside a corporate network, where *N*<10000. On Internet-scale applications (*N>1 million*), a message might take more than 10 hops. Each overlay hop potentially increases the relay delay compared to IP routing. In addition, each overlay hop requires that the message traverse all the way up and then back down the network stack. Queuing and processing delays can add to end to end routing delay. Finally, overlay nodes are generally edge nodes, with each overlay hop routing across slower and more congested edge links.

Clearly, reducing the number of overlay hops for peer-to-peer (P2P) protocols can dramatically improve application performance. We can leverage the routing state to routing hops tradeoff and increase the number of neighbor links maintained per peer. The challenge is to do so while keeping maintenance costs manageable. Existing proposals ([7][8]) use extremely large membership tables to achieve one-hop routing for networks with 1000's of peers, and two-hop routing with millions of peers. However, they employ a fixed hierarchical network structure, reducing the protocol's ability to adapt to network changes and scale beyond initial size estimates.

We achieve both goals of low routing hops and low maintenance costs in an adaptive system by integrating P2P routing with efficient membership maintenance algorithms. We start with Pastry, an existing prefix routing protocol, and expand its prefix routing base *b* from 16 to 4096. Using *b*=4096, we can achieve one-hop routing with 4096 nodes and two-hop routing across 16 million peers. While traditional protocols require a node to periodically probe its links to each of its neighbors and thus make the maintenance of routing entries with *b*=4096 infeasible, we leverage cost-efficient membership protocols ([4][6]) to maintain routing entries and detect link or node failures. Each node belongs to a few self-contained routing groups, and forwards routing messages through members of these groups. To maintain the routing entries, we use an efficient membership protocol in which each node only probes a small number of other members, and status changes detected are propagated quickly to all members in the same groups using an efficient and scalable broadcast mechanism. For a group of size *G*, this reduces maintenance cost per peer from *O*(*G*) to *O*(1).

This paper makes three contributions. First, we introduce the concept of using membership protocols to minimize P2P route maintenance. Second, we introduce Z-Ring, a protocol that utilizes discrete groups for fast routing, adapts to network size changes gracefully and scales to very large networks. Finally, we demonstrate via analysis and simulation that Z-Ring significantly reduces routing hops, maintains high routing success rate while keeping maintenance costs low on large networks.

The rest of the paper is organized as follows. We describe the basic Z-Ring protocol in Section 2 and its adaptability and scalability features in Section 3. Next, Section 4 describes our membership implementation. Section 5 evaluates Z-Ring analytically while Section 6 evaluates the protocol with extensive simulations. Finally, we discuss related work in Section 7 and conclude in Section 8.

2. The Z-Ring Protocol

In this section, we describe the basic operations of the Z-Ring protocol. We first describe some basic terminology and parameters, along with background information on group membership protocols and Pastry.

In our examples, we assume that peers use 160-bit integer ids, with bits indexed from left to right as 0–159. Logically, all peers are ordered in a *ring* based on their ids in the id space, as in Chord [15] and Pastry [13]. Routing groups have size $G=4K=2^{12}$, a size too large for Pastry to maintain as the base of digits, but small enough for an efficient membership protocol to maintain with low bandwidth costs. In this section, we assume the network size N is 16 million = 2^{24}. We show Z-Ring's adaptation to general systems sizes in Section 3.

2.1. Cost-efficient membership protocol

On each peer Z-Ring uses a membership protocol to maintain a large routing table, which is the set of active peers belonging to the same group. While traditional group membership protocols ([1]) provide strong consistency in membership views, their heavier overheads restrict them to cluster environments. In contrast, Z-Ring does not require strong or eventual consistency (i.e., eventually all peers agree on the same membership view). Z-Ring allows inconsistencies in membership views throughout the lifetime of the system. It only requires that the membership view is maintained with a certain level of accuracy in order to keep the number of routing hops low and the routing success rate high. This relaxed consistency allows Z-Ring to significantly lower bandwidth costs.

Cost-efficient membership protocols exist in literature. For example, the SWIM protocol [4] uses gossiping for failure detection and dissemination. We build a protocol based on Pastry by using Pastry's leafset to limit detection of peer join and leave events, and using Pastry's routing table to quickly disseminate events to all peers in the group. We add an anti-entropy [5] protocol to increase resilience to lost events in dissemination. The combination means membership changes are quickly disseminated to all nodes in the system, providing highly accurate routing state for Z-Ring. We defer detailed descriptions of our membership protocol implementation to Section 4.

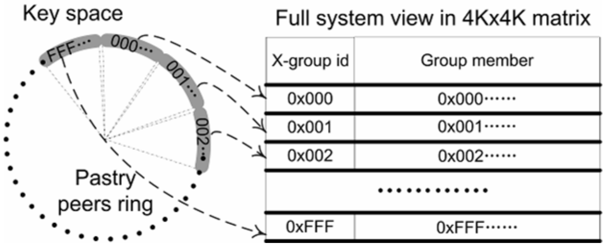
2.2. Pastry prefix routing

Z-Ring accelerates prefix routing in protocols such as Pastry by using extremely large prefix bases. We give a brief description of the Pastry protocol, and refer readers to the full Pastry paper for additional details [13].

Pastry organizes the peers in a ring based on their ids, and routes messages by key to the peer whose id is numerically closest to the key. Pastry routing state consists of a leafset and a routing table. The leafset on a peer records its closest $L/2$ peers on each side of the ring while the routing table on a peer has $\log_b N$ levels, where b is a small base (typically 2 to 16) and N is the number of peers in the system. Each level j (from 0 to $\log_b N$) of the routing table has b-1 entries, each referring to a peer whose id shares the first j digits (base b) with the local id, but whose $(j+1)^{th}$ digit differs. Each entry contains the IP address of the referred peer.

Pastry routes messages by matching prefixes in each routing step. At each hop, Pastry routes the message to a node that matches at least one additional digit of the destination id. When routing encounters an empty entry in the routing table, messages are forwarded through leafsets to the final destination node.

2.3. X-group routing in Z-Ring



X-group id	Group member
0x000	0x000······
0x001	0x001······
0x002	0x002······
············	
0xFFF	0xFFF······

Fig. 1. Full system view of the routing matrix with the relationship to the view of Pastry peer ring.

We begin our introduction of the Z-Ring protocol using a simplified routing scenario. With $G=2^{12}$ and $N=2^{24}$, Z-Ring routing can be viewed as routing on a two dimensional matrix (Fig. 1 and Fig. 2). We first describe one-level Z-Ring routing (horizontal routing within a row of the matrix) with membership, and then describe the second level Z-Ring routing (vertical routing to locate a row). Together they provide two-hop routing across 16 million peers in the system. This performance is for the ideal case where each cell in the routing matrix has exactly one peer id and its IP address. Randomized ids can result in some empty cells, and we discuss how to deal with the imperfect routing matrix in subsection 2.5.

As we described, the main idea in Z-Ring is to classify peers into different closed groups at each routing level. To demonstrate, we organize the 16 million peer ids into a 4Kx4K matrix (Fig. 1). Each row in the matrix contains peer ids with the same bits 0~11 and forms an *X-group*. The bits 0~11 are the id of the X-group. In our example, 16 million peers are organized into 4K X-groups, where each X-group represents a continuous range or arc in the Pastry id ring (Fig. 1).

2.3.1. **Routing.** For Z-Ring routing with X-groups, each peer maintains its full X-group membership list in its membership table. When routing a message with a key k, assume for the moment we use basic Pastry to resolve the first 12 bits of k , that is, it reaches a peer that is in the same row with the destination id that owns the key k. Then by looking up the X-group membership list, the destination can be located in one more hop. The X-group hop can be view as resolving 12 bits (bits 12~23) in one step.

Using only the X-group routing, we can reduce the number of hops from 6 for Pastry routing using hexadecimal base (b=16), to 4: three Pastry routing hops followed by one X-group hop.

2.3.2. **Maintenance.** We now describe join and leave processes for Z-Ring.

Join: X-groups are defined by their id prefix, and each X-group represents continuous arcs on the id space ring. When a new peer joins the system, it joins the Pastry ring and locates the group arc that it belongs to. The new joining peer will find any nodes that exist inside the arc, and will learn from them about all other peers in the same group. The group maintenance protocol will notify all other group members about the new arrival. If no peer exists in the group, the new peer will initiate the group.

Leave: Peers leaving the system are handled by the membership protocol. When a peer leaves, the membership protocol notifies other peers in the X-group, who then update local routing state accordingly.

2.4. Y-group routing to resolve the first 12 bits

To accelerate the resolution of the first 12 bits, we organize every row's peers into columns according to its id bits 12~23. Fig. 2 shows the matrix with Y-group applied. Similar to the X-group, every peer also uses its id's bits 12-23 to identify its Y-group. We call these bits the Y-group id. In the ideal case, the routing matrix is perfectly balanced, and each column (Y-group) has 4k entries sharing the same bits 12~23.

2.4.1. **Routing.** Each peer belongs to both an X-group and a Y-group. With these groups, the peer can see the entire row and column it resides in. Fig. 2 shows an intuitive view of these groups. Using the Y-group, nodes can route to a node matching the same first 12 bits as the destination in 1 hop. Using both X- and Y-groups, Z-Ring routes arbitrary messages in a network of 16 million peers using only 2 hops: Y-group routing resolves bits 0~11 in one hop and X-group routing resolves bits 12~23 in one hop.

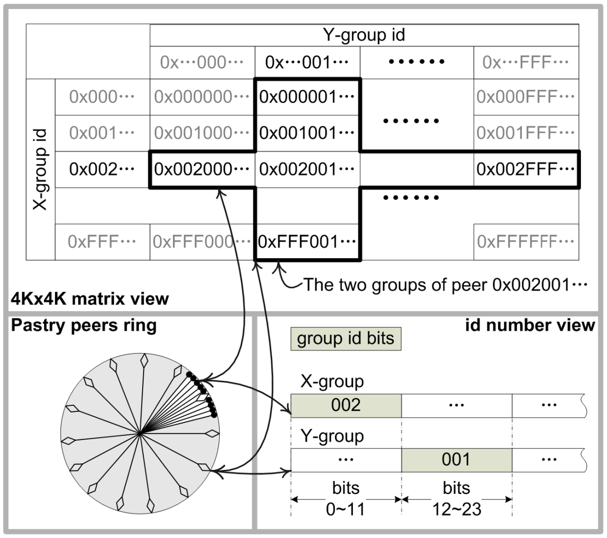


Fig. 2. Full system routing matrix, with the relationship to the view of Pastry peer ring and to the view of X- and Y-group peer ids.

2.4.2. **Maintenance.** The challenge in Y-group maintenance is how a new joining peer locates other members in its Y-group. Because the Y-group is sparsely distributed throughout the Pastry ring (Fig. 2, the ring view), locating Y-group members is not trivial. We solve this problem by applying what we do for the X-group, but with a new twist.

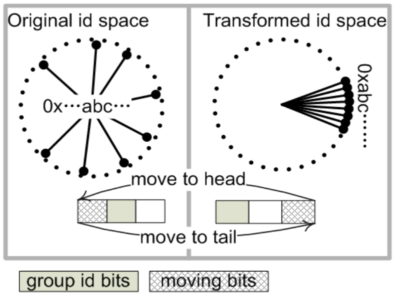
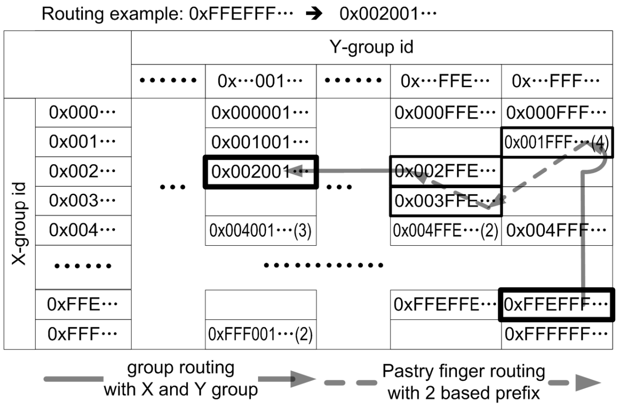


Fig. 3. Id transformation for Y-group bootstrap.

Locating X-group members utilizes the underlying Pastry ring to route the joining request from the new peer to the arc corresponding to its X-group members. To apply the same idea to the Y-group, we transform Y-group bits 12~23 to position 0~11 as shown in Fig. 3, by rotating the bits 0~11 of every peer id to the tail and building another Pastry ring. The Y-group members in the original id space become clustered together on a continuous arc in the newly-transformed id space, just like the X-groups in the original space. Therefore, in the transformed id space, the simple bootstrap protocol can be reused for a new joining peer to locate its Y-group members. To support Y-group bootstrap, Z-Ring thus requires another set of Pastry routing state for the transformed id space.

2.5. Routing with imperfect routing matrix

Our previous examples assume that the routing matrix is perfect in that each cell in the routing matrix has exactly one entry. In reality, randomized selection of peer ids means a particular group may have no peers or be crowded by many peers. In this section, we describe how routing works with an imperfect routing matrix.


Routing example: 0xFFEFFF··· ➔ 0x002001···
group routing with X and Y group
Pastry finger routing with 2 based prefix

		Y-group id				
		·······	0x···001···	·······	0x···FFE···	0x···FFF···
X-group id	0x000···		0x000001···		0x000FFE···	0x000FFF···
	0x001···		0x001001···			0x001FFF···(4)
	0x002···	···	0x002001···	···	0x002FFE···	
	0x003···				0x003FFE···	
	0x004···		0x004001···(3)		0x004FFE···(2)	0x004FFF···
	·······		············			
	0xFFE···				0xFFEFFE···	0xFFEFFF···
	0xFFF···		0xFFF001···(2)			0xFFFFFF···

Fig. 4. Full system view and Z-Ring routing with imperfect routing matrix (scale=16 million).

The issue with an imperfect matrix is that routing through cells which can potentially be empty. Fig. 4 shows such an example. Cell (0x002, 0xFFF) is empty and unreachable, while cell (0x001, 0xFFF) is filled by 4 peers. Routing a message from 0xFFEFFF to 0x002001 is broken at cell 0x002FFF because it has no reachable peers. Our solution is straightforward. Since membership routing is only to accelerate Pastry routing, when acceleration fails, we fall back to normal Pastry routing. More specifically, when the cell (0x002, 0xFFF) is empty, the routing message will be sent to the nearest available cell (0x001, 0xFFF) in the Y-group. This incomplete group routing hop resolves only 10 bits instead of 12 bits. The 2 unresolved bits depends on Pastry routing hops. Our example takes 2 hops on 2-based Pastry routing table. If we use 4 or larger value based Pastry, it can be completed in one Pastry routing hop. Results from our analysis (Section 5) and simulation (Section 6) show that the average Pastry routing hops is small (0.445 with *N*=16 million and uniform id distribution).

It is important to see that Z-Ring routing and maintenance at each level are symmetric. This modular use of membership protocols simplifies implementation and makes the protocol adaptable to larger networks, as we show in the next section.

3. Z-Ring adaptability and scalability

We have described the Z-Ring protocol for a network of 16 million peers. We now show how the basic Z-Ring protocol can adapt to arbitrarily large network sizes. This flexibility is the main advantages of Z-Ring over other static one-hop or two-hop protocols ([8][7]). We show its adaptability in Section 3.1, and its scalability in Section 3.2 through the example with *N*=64 billion ($N=G^3$).

3.1. Adaptability

When the network size is a power of two ($N=2^t$), Z-Ring can always maintain group size at the level of 4K by adjusting the length of the group id. If the network size is not a power of two ($2^t<N<2^{t+1}$), the group size needs to be adjusted but we keep the size at around 4K level to minimize maintenance cost. We now discuss these two cases separately.

3.1.1. Scale is power of two ($N=2^t$). As an example, we show how the protocol adapts to a network of size N=224 shrinking to size N = 220. For groups of size G, we can resolve log2G bits in 1 group hop. Therefore we try to keep the group size constant while reducing the number of groups in all X-group and Y-group membership tables.

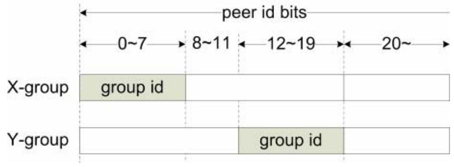


Fig. 5. Two levels of groups maintained at one peer, for network size of one Million (2^{20}). There is an overlap between the to-be-resolved bits of Y-group (bits 0~11) and the to-be-resolved bits of X-group (bits 8~19).

With $N = 2^{20}$ and $G=2^{12}$, we will have 256 X-groups, which can be identified by the first 8 bits (bits 0~7) (Fig. 5). Comparing with the case in the previous section where $N=2^{24}$ (Fig. 2), the X-group id shrinks 4 bits. This means one X-group covers a longer arc in the Pastry ring. The X-group bootstrap still uses the same protocol, but in this case, it only needs to match the prefix of 8 bits in order for a new joining peer to locate its X-group members.

The case for the Y-groups is the same. Y-group id bits shrink from bits 12~23 to bits 12~19 so that each Y-group on average still has 4K members. Bits 12~19 becomes bits 0~7 in the transformed id space (Fig. 3), so Y-group bootstrap is still the same --- it needs to match 8-bit prefix in the transformed space. In general, group ids have t-12 bits with N=2^t.

The routing procedure in this case does not change from that of Fig. 2. The routing steps are still Y-group routing followed by X-group routing, with Y-group routing resolving bits 0~11 and X-group routing resolving bits 8~19. The difference is that the lowest 4 bits resolved by Y-group routing overlap with the highest 4 bits to be resolved by X-group routing (Fig. 5). This overlap reduces the need for the complementary Pastry routing hops between Y-group routing and X-group routing. The complementary Pastry routing hops are only needed when Y-group routing fails to resolve the top 8 bits, the probability of which is negligible (Section 5.1). Therefore, the routing hop number is reduced when N is between G and G^2 because the complementary Pastry routing hops is reduced, and it is very close to two hops in many cases (*e. g.* when N is about $G^2/8$ or less).

3.1.2. Scale is between power of two ($2^t \sim 2^{t+1}$). When the network size is not a power of two, the size of each X- and Y-groups will change. We demonstrate system adaptability with the case of scale growing from 220 to 221.

When the network sizes up from 2^{20}, the number of groups is kept constant because it cannot continuously change. Thus the size of each group grows proportionally during this stage. For example, when the network size is $1.1 \cdot 2^{20}$, average group size will be 1.1G = 4505. When group size keeps increasing beyond the critical point of 4/3G = 5461, group will split into two smaller groups with size 2/3G = 2731. The choice of 4/3G as the threshold is to keep the average group size to be around G.

Group splitting is done by adding the 8th bit into the group id: The X-group is split by expanding group id from bits 0~7 to bits 0~8; the Y-group is split by expanding group id from bits 12~19 to 12~20. Here we can see that id transformation scheme (Fig. 3) does not change while Y-group id expands during group split. This means that Z-Ring consistently uses the same background Pastry for the transformed id space, rather than rebuild a Pastry every time group id changes.

Symmetrically, when the network size decreases, the groups will be merged. Two groups are neighbors if they are at the same level and their group id only differs at the last id bit. When the total number of members in the two neighboring groups is reduced to lower than 4/3G = 5461, the two groups will be merged to one, and the merged group removes the last bit in the ids of the two original groups and thus has one less bit in its id.

To avoid frequent splits and merges when the group size is around the threshold 4/3G, we can put some buffer zones around. For example, the merge occurs when the combined size of two neighboring groups is below 4/3G-1/10G, and the split occurs when the group size is above 4/3G+1/10G.

In a real system, not all peers will split/merge their group simultaneously by expanding/shrinking their group ids. Some peers may have 8 bits group id while others have 9 bits, especially when network size is close to the critical point. Fortunately Z-Ring is only an acceleration scheme and thus does not require highly consistent group id. Based on the aggressive routing rule, we guarantee that routing messages are always able to route towards destination. Therefore, as long as Z-Ring provides some mechanism to allow convergence of group ids, the transient inconsistency will not cause problems in routing.

To support group merges, Z-Ring needs some extra inter-group support beyond the normal intra-group membership protocols. These supports include: (a) periodic communication between the two neighboring groups to check if the merge threshold has been surpassed; (b) fast merging of two neighboring groups together; and (c) gossiping within the group and between the neighboring groups to resolve potential inconsistencies in group ids. In Section 4, we will show that our implementation provides these inter-group membership supports.

An important property of Z-Ring's adaptability is that it is based on local decisions. Whether to split a group or merge two groups is based on the size of the local group perceived by the peer and the size of its neighboring group. Therefore, group splitting or merging does not require any global knowledge, such as N, the scale of the system.

3.2. Scalability (N=G^3)

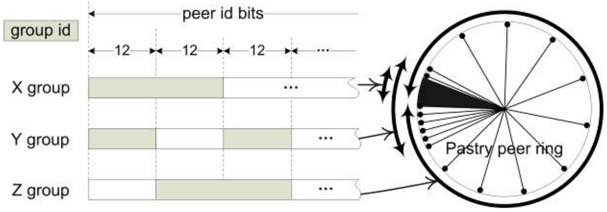


Fig. 6. Three levels of groups maintained at one peer, for network size of 64 Billions (2^{36}). X-group spans over the shortest arc with the highest density. Y-group spans over moderate arc with the moderate density. Z-group spans over the longest arc with the lowest density.

In this section we discuss how to scale up our solution for larger systems up to size G^3. The idea is to enable the 3-d matrix by Z-groups on z-axis. We can imagine the 3-d matrix as 4K layers of the 2-d matrix. The Z-group index is the layer index. Peer ids includes {layer-index, row-index, column-index}, where each index component is 12 bits long. Bits 0~11 denote the layer, bits 12~23 denote the row, and bits 24~35 denote the column. With this scheme, X-group members share bits 0~23, Y-group ids share bits 0~11 and 24~35, and the Z-group shares bits 12~35 (see Fig. 6).

Routing consists of three steps: Z-group resolving bits 0~11, Y-group resolving bits 12~23, and X-group resolving bits 24~35. When bits in group hops cannot be resolved, routing falls back on Pastry routing hops.

Maintaining three types of groups is also similar. The X- and Y- groups are maintained as before. The Z-group is maintained by the Pastry in id space transformed from the original by rotating bits 0~11 to the tail (Fig. 7). To further increase the namespace, we can continue to add more groups beyond the Z-group. Each additional group level increases the namespace size by a factor equal to the group size.

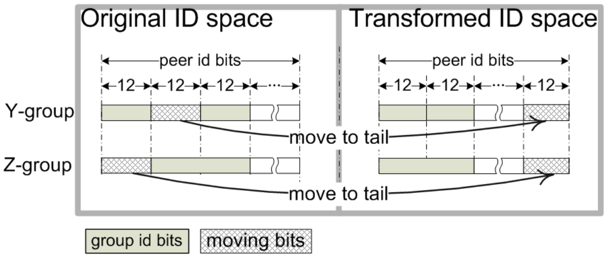


Fig. 7. Id transformation for Y- and Z-group maintenance.

4. Membership protocol implementation

4.1. Intra-group membership protocol

Our implementation of the intra-group membership protocol is conceptually similar to SWIM [7]. It includes leave/join detection and event dissemination. Detection is performed between a small number of peers, who then disseminate the event to the entire group using fast broadcasts complimented by periodic anti-entropy. The result is an efficient membership protocol that incurs much less overhead than pair-wise periodic probing between members of the group.

For each routing level (X, Y, Z), we use an independent instance of Pastry (base 2) to connect all group members using leafsets (ID transformation is needed for Y- and Z-groups). Each peer sends periodic probes to its leafset members. When a peer *x* joins or leaves the network, the event is detected by *x'*s leafset. These peers push the event out to their leafset members and their routing table entries. Other peers repeat the process tp push the event to all group members. This instance of Pastry is internal to the group level, but utilizes the same node ids as the global network. Since all peers in the group share a prefix, routing inside the group is consistent with normal prefix routing.

To ensure event notifications are ordered, events are embedded with a timestamp generated by its source, either *x* for a join event, or the first peer to detect *x*'s absence in a leave event. The timestamp uses the source's local physical clock, assuming that the clock does not go backwards even after recovering from failures. For leave events, a peer *x* is "dead" when its leafset peers fail to receive 3 consecutive heartbeats. The peers then propagate the leave event, where the timestamp is the last heartbeat timestamp received from *x* plus a small ε, which is any value smaller than the granularity of the incarnation timestamp. The small value ε guarantees that a leave event does not conflict with any join events in incarnation timestamps, and it correctly override prior join events and is overridden by latter join events.

To avoid redundant messages, we build a broadcast tree out of Pastry's routing tables. The initiator of the broadcast sends the message to all peers in its routing table (within the group range). When *x* receives a message from *y*, it forwards it to every peer *z* in its routing table where *z* and *x* shares a longer prefix in their ids then the prefix shared by *y* and *x*. Also, an event is not forwarded if the peer's local membership list is already updated by anti-entropy or leafset updates, no matter if it is updated by the same event or an older event. This may potentially cause event losses during broadcast, but it is unlikely and we choose the tradeoff of saving cost. Leafset updates guarantee that every node in the group will be notified of the event.

Event broadcasts may not reach all members in a group due to failures or node departures. To compensate, we augment broadcasts with periodic anti-entropy rounds to ensure consistency of membership lists. At every heartbeat interval, a peer randomly chooses another peer in the group to exchange routing state and resolve differences in membership lists. We use checksums to minimize bandwidth of these exchanges.

Peers exchange a checksum of membership entries by performing an XOR of all ids of all membership entries. If checksums do not match, peers try to predict the cause of the inconsistency by checking for an entry matching the XOR of the two checksums, and using its timestamp to determine validity. If the inconsistency is not caused by a single entry, the full membership lists are exchanged. The XOR checksum significantly reduce the need to send the full membership list in anti-entropy.

4.2. Supporting group splits and merges

As pointed out in Section 3.1.2, we also need additional mechanisms to support group splits and merges in a dynamic system. When a group splits, anti-entropy is used to resolve inconsistencies. Pair-wise anti-entropy between nodes will quickly propagate information about group split across the group. Group merges are triggered by border members that are in two neighboring groups and form leafsets together. These border members exchange their local group size in their leafset heartbeats and determine if their total size is below the threshold (e.g., 4/3*G*) to trigger a group merge. If so, the merge event is propagated by event broadcast and anti-entropy to all members in the two groups. The merge event message contains the full membership list of the other group to be merged with.

5. Protocol analysis

We now analyze the number of routing hops, routing success rate, and the maintenance cost of Z-Ring. The variables used in the analysis are given in Table 1.

Table 1: Variables used in the analysis

Variable	Comment	sample value
N	network size	16M
G	membership group size	4K
B	prefix routing table base	2~16
L	leafset size	4
T	average session time of a peer	30 minute
D *	probe interval on one entry in the membership table	seconds

* Assuming probes and anti-entropy messages sent at the same frequency.

5.1. Number of routing hops

Given group size *G* and system size *N*, Z-Ring needs $\log_G N$ level membership tables, and thus it requires $\log_G N$ group routing hops. After each group routing hop, some bits may not be resolved and thus complementary routing hops on the background Pastry are needed. We now calculate the average number of these routing hops between any two group hops. We assume that the peer ids are distributed uniformly at random for this calculation, and we use 2-based Pastry. We do not consider false negatives and false positives in the membership protocol for this analysis.

Let $G=2^g$, and $N=r\cdot 2^t$, with $t>g$ and $2/3<r\leq 4/3$. Thus the length of group id is t-g, and each group contains $r\cdot G$ members on average.

Each group routing hop is targeted to resolve *g* bits, but due to the imbalance in peer ids, it may resolve less than *g* bits. We now calculate the probability that one group routing hop fails to resolve exactly *j* bits, with $0\leq j\leq g$, when the group size is $r\cdot G$.

We reduce the problem to a ball-and-box probability problem. There are totally 2^g values for the *g* bits to be resolved. We assign one box to all values that share the same g-j bit prefix. So there are totally $2^{g-j} = G/2^j$ boxes. The routing target id is in one of the boxes. Each routing entry in the group of size $r\cdot G$ corresponds to a ball to be thrown randomly into the boxes. The probability that the group routing hop resolves at least g-j bits is the same as the probability that the box with the target id contains at least one ball when throwing $r\cdot G$ balls into the $G/2^j$ boxes at random. This probability is given as:

$$1-\left(1-\frac{2^j}{G}\right)^{r\cdot G} \approx 1-e^{-r\cdot 2^j} .$$

Therefore, the probability that the group routing hop resolves *exactly* g-j bits, i.e., fails to resolve *exactly j* bits is:

$$\left(1-e^{-r\cdot 2^j}\right)-\left(1-e^{-r\cdot 2^{j-1}}\right)=e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^j} .$$

When the group routing hop fails to resolve *j* bits, the complementary Pastry routing hops needs to resolve these bits. With 2-based Pastry, it resolves one bit at a time with a chance that the subsequent bits happen to be resolved simultaneously. The first bit in the last *j* bits has to be resolved by Pastry and it takes one hop. For each of the rest (*j*-1) bits, when it is the time to resolve the bit, with probability 0.5 the bit has already been resolved by the previous Pastry routing hop. So each bit needs one more hop with probability 0.5, and thus each of the (*j*-1) bits costs 0.5 Pastry routing hops. Hence, on average the total number of Pastry routing hops to resolve *j* bits is (*j*+1)/2.

Therefore, the expected number of complementary routing hops is:

$$\sum_{j=1}^{g}\left(e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^j}\right)\cdot\frac{j+1}{2} . \qquad (1)$$

When r=1 the above value converges to about 0.445 with $g\geq 4$.

We now need to consider two special cases. First, when the network size *N* is not exactly a power of *G*, the to-be-resolved bits of Y-group and X-group overlap (see Fig. 5), and this reduces the number of complementary Pastry routing hops needed. Suppose *k* bits are overlapped, then $k = g\cdot\ t/g\ -t$. In this case, the complementary Pastry routing hops are needed only when the Y-group routing fails to resolve more than *k* bits and it only needs to resolve j-k bits where *j* is the number bits failed to be resolved by Y-group routing. Thus the expected number of complementary Pastry routing hops for Y-group is adjusted to:

$$\sum_{j=k+1}^{g}\left(e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^j}\right)\cdot\frac{j-k+1}{2} . \qquad (2)$$

The second special case is when an entire group routing level can be bypassed because the source and the destination happen to belong to the same group at the next level. For example, with two-level routing, if the source and the destination are in the same X-group, then Y-group routing is bypassed. For higher level routing beyond Y-group, this is unlikely to occur since the probability is at most $1/G$, and thus we ignore this probability. But for Y-group routing, the probability cannot be ignored if there is some overlapping in the to-be-resolved bits. Using the notation above, when there are k bits overlapped, the probability that Y-group routing is bypassed is $2^k/G=2^{k-g}$.

To summarize, let $G=2^g$, $N=r\cdot 2^t$, with $t>g$, $2/3<r\leq 4/3$, and $k=g\cdot\ t/g\ -t$. Overall, (a) we need t/g group routing hops; (b) for group routing beyond Y-group, the average complementary Pastry routing hop between each group routing level is given by (1); (c) between Y-group and X-group routing, the average complementary Pastry routing hops is given by (2); and (d) with probability 2^{k-g}, Y-group routing can be bypassed. We combine all these factors into the following formulas:

When t/g =2, we have

$$2-2^{k-g}+(1-2^{k-g})\cdot\sum_{j=k+1}^{g}\left(e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^{j}}\right)\cdot\frac{j-k+1}{2},$$

And when t/g >2, we have

$$\lceil t/g\rceil-2^{k-g}+(\lceil t/g\rceil-2)\cdot\sum_{j=1}^{g}\left(e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^{j}}\right)\cdot\frac{j+1}{2}+$$

$$(1-2^{k-g})\cdot\sum_{j=k+1}^{g}\left(e^{-r\cdot 2^{j-1}}-e^{-r\cdot 2^{j}}\right)\cdot\frac{j-k+1}{2}$$

Based on the analysis, we plot the curve that shows the number of routing hops under different network sizes (Fig. 8). In the next section, we will see that this analytical result matches with the simulation result very well. In this figure, the step function represented by the dashed line takes the value at size G^n for all sizes between G^{n-1} to G^n. Its formula is:

$$1.45\lceil\log_G N\rceil\text{- }0.45$$

It can be seen that this formula gives a simple and reasonable estimate of the actual number of routing hops.

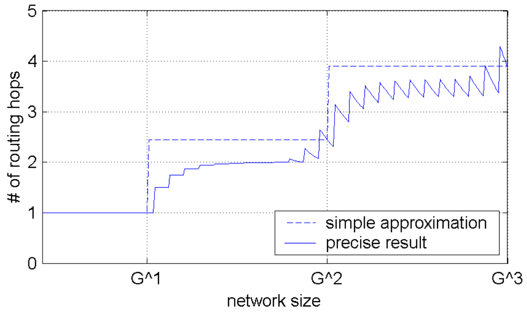


Fig. 8. Routing hops based on the analytical result.

5.2. Routing success rate under churn

We consider routing by simple message forwarding without retries. The routing success rate is thus the single hop success rate powered by the number of routing hops. A single hop fails if it routes to a failed peer that is still in the membership table, so the single hop success rate is determined by the false positive ratio of the membership service, which is in turn affected by the system churn rate.

We assume that a peer is declared failed and removed from the membership after three consecutive failed probes (as in our implementation). We also assume that the propagation delay of a failure event is much smaller than the probe interval. Thus, the false positive ratio of the membership protocol is equal to the probability that a peer being probed has already failed, which can be approximated by $3d/T$, where d is the probing interval and T is the average session time of the peer. Therefore, the success rate of Z-Ring is approximated by:

$$\left(1-\frac{3d}{T}\right)^{1.45\log_G N-0.45}$$

For example, if every peer only stays online for 30 minutes for each session, with G=4K and $N=G^2$, the routing success rate is roughly 0.92, with $d=T/100$. This is a very good success rate under high system churn of almost 10000 peers joining and leaving every second, with probes every 18 seconds.

5.3. Maintenance cost

Z-Ring's maintenance cost consists of local memory cost and network bandwidth costs. Z-Ring's local memory cost is $O(G\log_G N)$ for maintaining $\log_G N$ level of membership list and $O(((b\text{-}1)\log_b N+L)\log_G N)$ for maintaining $\log_G N$ background Pastry tables. This amounts to only a few 10s of Kbytes for $G=2^{12}$ and $N=2^{36}$. So local memory cost is not an issue, and we henceforth focus on the network cost.

We measure the network cost by the number of message per second sent by a peer in the system. There are several types of maintenance messages in Z-Ring: (a) probing messages for underlying Pastry's leafset and routing table entries; (b) repair messages to fix a failed entry in Pastry; (c) membership broadcast messages for peer join and leave events; (d) membership anti-entropy messages; and (e) bootstrap messages for a new peer to locate its group members. Type (e) messages are ignored since there are only a few messages per online session of a peer. Type (d) messages are ignored because there is only one message per probe interval.

(a) With 2-based Pastry, the size of the routing table and the leafset of each Pastry ring is $(\log_2 N+L)$, and each entry is probed every d seconds. With $\log_G N$ Pastry rings in Z-Ring, the total probing cost at one peer is ($\log_2 N$+L) $\log_G N/d$.

(b) Each of the $\log_2 N$ Pastry routing table entry needs repair every T seconds on average. In the worst case, the repair operation needs $\log_2 N$ messages due to the routing operation. So the repair message cost is $\log_2 N \cdot \log_2 N \cdot \log_G N / T$.

(c) For membership broadcast, every join or leave event is broadcast through the Pastry routing table broadcast tree to all members in the group. So every event generates G messages. Each peer generates one join event and one leave event for every one of its online session, with a period of T. Therefore, on average every peer's broadcast cost is $2G/T$ for maintaining each group, and there are $\log_G N$ groups for every peer.

Therefore, the total network cost is:

$$\left(\frac{\log_2 N + L}{d} + \frac{(\log_2 N)^2}{T}\right)\log_G N + \frac{2G}{T}\log_G N$$

6. Simulation Results

We have fully implemented Z-Ring on top of the WiDS platform [10]. This platform encapsulates the network socket layer and system threads. The Z-Ring implementation includes three types of operations: message posting, message handler, and timer. This enables us to also run the real implementation on top of WiDS-Sim (simulation version of WiDS).

To validate Z-Ring performance, we need to simulate up to G^2 peers. With the parallel simulation feature of WiDS, we are able to run simulations on 6 PCs to simulate 65536 peers with G=256. Future experiments will simulate even larger scales. The parameters for the experiment are:

- Leafset heartbeat interval: 10 seconds
- Leafset size: 4 neighbors
- Leafset failure threshold: after 3 failed heartbeats
- Pastry routing table entry probe interval: 10 seconds
- Anti-entropy interval: 10 seconds.
- Network latency: random values from 2ms to 100 ms.

We measure the numbers of routing hops, success rates and maintenance costs while varying system sizes and churn rates.

6.1. Routing hops over system size

Our first experiment measures the number of routing hops over various system sizes. We choose 31 sample points as system size grows from 2,048 to 65,536. For each data point, we first stop join and leave events to stabilize the system, and then initiate 2000 routing requests across randomly chosen source and destination pairs. Fig. 9 shows the results. As expected, messages take 2 overlay hops on average, increasing logarithmically with network size. The figure shows that our simulation result matches with our analytical result very well.

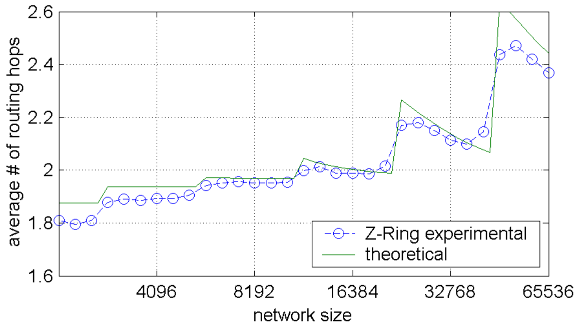


Fig. 9. Routing hops with various network sizes. Each point is based on 2000 individual random routings.

6.2. Routing hops and success rate over churn

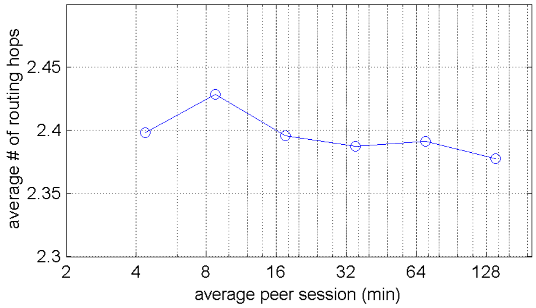


Fig. 10. Routing hops under churn.

Our second experiment measures the number of routing hops and success rates under churn. Note that by success rate, we mean end-to-end delivery rate without retransmissions. We fix the system size to 65,536, and vary the churn rate from 15 join/leaves per second (average online session length of 140 minutes), to 500 join/leave per second (average online session of 4.4 minutes). The experiment includes 6 sample points. Each point contains results from 1000 random route requests.

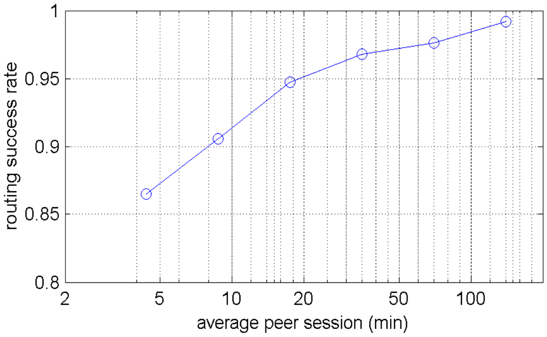


Fig. 11. Routing success rate under churn.

System churn generates two kinds of false entries in membership tables: false positives and false negatives. False negatives reduce the membership table size and thus increase the number of routing hops. False positives cause routing message loss when messages are forwarded to dead peers, therefore reducing the routing success rate. Fig. 10 shows the impact of system churn on the number of routing hops. The result is a tiny rise of average routing hops at high system churn rates, which means that false negatives are very low, and that event propagation effectively reaches most peers. However, the impact of false positives to routing success rate is significant, because it takes three probe intervals to declare a peer dead. Fig. 11 shows the routing success rate over various system churn rates.

The routing success rate drops significantly at high churn rates. According to previous measurements [14], the typical average session time for today's P2P network is about 30 minutes. In Fig. 11 the corresponding routing success rate is about 96%. Recall that this is the probability that the message reaches the destination without retransmissions. The expected delivery rate with retransmissions would be exponentially higher.

6.3. Maintenance cost of membership protocol

Our third experiment measures the maintenance cost of Z-Ring over various system churn rates on a network of size 65,536. Other parameters are identical to the second experiment. We vary the same test over the same variance in churn rates, and count the number of maintenance messages in probing and event dissemination traffic on each peer. Fig. 12 presents these results. Every message is only counted at its sender side, i.e., each peer only counts its outgoing traffic.

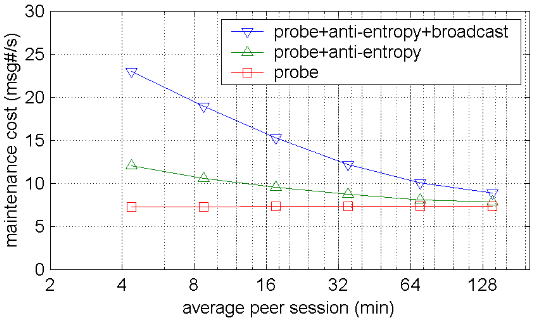


Fig. 12. Z-Ring maintenance under churn (membership and probe)

The maintenance cost of Z-Ring's group membership scales inversely to the average session length, i.e., proportional to the churn rate. Consider a typical case when the average online session time is 30 minutes [14], there are 1.7 messages per second for anti-entropy and 4.0 messages per second for broadcast. Since all these messages are smaller than 256 bytes, and broadcast messages are 48 bytes, these results mean that each peer consumes up to 0.63KB/s of outgoing bandwidth for membership maintenance. Total bandwidth used, including incoming bandwidth, is less than 1.3KB/s. Note that our experiment uses G=256. For G=4096, the cost will likely scale 8-fold to 10KB/s, still reasonable for broadband users.

6.4. Comparisons to Pastry

We implemented the Pastry protocol on WiDS platform and run Pastry simulations with base 16 and compare the results with our Z-Ring implementation. The comparison includes maintenance cost, routing hop number and success rate.

Fig. 13 compares the maintenance cost in number of messages sent per second by a peer with different average peer session times. It shows that the cost of Pastry does not vary with churn rate, because Pastry only maintains leafsets and routing tables with probes at constant rates. The cost of Z-Ring is affected by the churn rate, because when the churn rate is high, the membership protocol in Z-Ring needs to send out more broadcast messages to disseminate join and leave events.

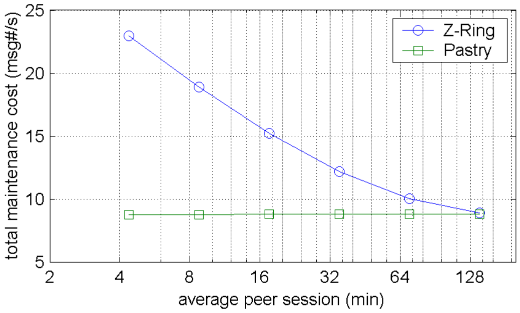


Fig. 13. Per peer maintenance cost comparison between Z-Ring and Pastry. The network size is 65536, group size is 256.

Our results show that Z-Ring's maintenance cost is similar to Pastry's when system churn rate is low. When the churn rate is high, Z-Ring's cost increases significantly, but is still reasonable for home users. The benefit is significantly lower number of routing hops per message. The reduced hop count leads to lower bandwidth cost incurred in data traffic. In addition, Z-Ring reduces the need for message retransmissions by increasing routing reliability.

Fig. 14 compares the number of routing hops between Z-Ring and Pastry for various network sizes. Z-Ring saves close to two hops when network size is 65536. While our simulations are limited to these sizes, the visible trends show that Z-Ring would show even an even bigger reduction in hops from Pastry for larger networks. Fig. 15 compares the routing success rate between Z-Ring and Pastry, under various churn rates. We see because Pastry routing requires more than double the hops of Z-Ring, its expected failure rate is also doubled. The reduction of overlay hops in Z-Ring leads to higher expected routing success rate and significant savings in network bandwidth.

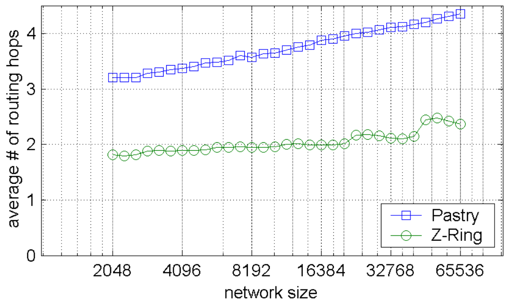


Fig. 14. Routing hops comparison between Z-Ring and Pastry with various network sizes and no system churn.

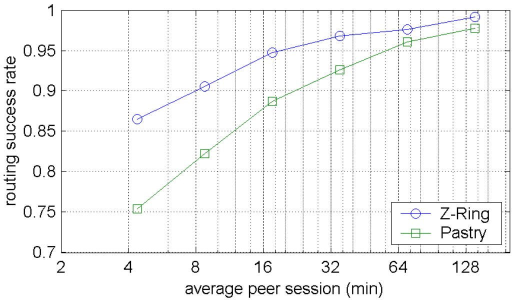


Fig. 15. Routing success rate comparison between Z-Ring and Pastry. The network size is 65536, group size is 256.

Bamboo DHT [11] proposes three methods that improves routing success rate. We already adopt periodical probe recovery in our comparison. While the other two methods (timeout calculation and proximity neighbor selection) can also be adopted in Z-Ring to further improve success rate.

6.5. Recovering from Network Partitions

Given the frequency of wide-area disconnections, recovering from network partitioning can significantly improve robustness for Internet applications. We run a partition healing experiment by connecting two independent partitions, each containing 32,768 nodes chosen from the same id space. After waiting for each partition to stabilize, one peer from partition A sends a heartbeat to a peer in partition B and begins partition healing. During the process, we measure the success rate of random route messages and the number of control messages generated. A message is successfully routed if it arrives at the destination closest to its goal id, even if the route would cross partition boundaries. Before healing, routing will be successful only if the source and destination lie in the same partition. For control messages, we only measure the amount of broadcast and anti-entropy messages exchanged, since the rate of normal heartbeats and leafset probes will not change during healing.

We found that a basic implementation of Pastry heals partitions slowly. Peers only forward information on new neighbors to their leafset members. This healing process converges in time of $C_1 \cdot N / leafset\_size$, where N is the system size. In contrast, fast dissemination of join/leave events in Z-Ring quickly notifies all members of the X-group, significantly speeding up the process. When a peer receives information on a new neighbor, it probes the neighbor to confirm its validity.

We note that the healing will slowly spread between X-groups, and can still take O (N/G) to converge. We accelerate this by forwarding new nodes discovered through X-group notification to the Y-group. Broadcast through the Y-group potentially reaches all X-groups in one broadcast step, and then each X-group notifies its members. This implementation accelerates the partition healing converge time from O (N) to O ($\log_G N$).

In this experiment, we initiate routing requests once every 10 milliseconds, log all messages, and measure the message counts and success rates of requests every second. Fig. 16 shows how the success rate of route requests significantly increases as the partitions heal. As expected, the success rate starts fluctuating around 50%. Healing is complete after 50 seconds.

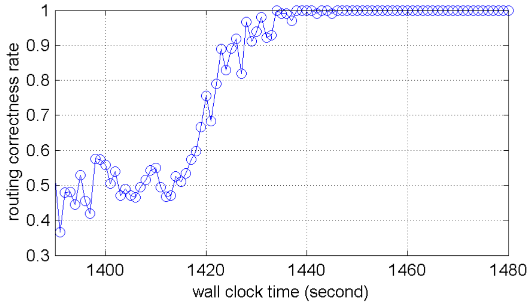


Fig. 16. Routing correctness during partition healing of 65536 nodes

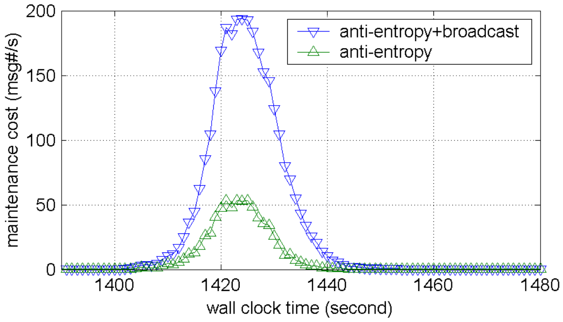


Fig. 17. Number of broadcast and anti-entropy messages per peer during partition healing of 65536 nodes

In Fig. 17, we plot the number of broadcast and anti-entropy messages during partition healing. The first message connecting the partitions is initiated at time

1400. The figure shows that the partition healing is done after only 5 heartbeat intervals (50s). The cost of broadcast and anti-entropy messages grows up to 200 messages per second for each peer. The actual bandwidth cost per peer is reasonable (20KB/s), since messages are smaller than 256 bytes and broadcasts are 48 bytes. Over the healing process, each peer sends 2900 messages on average.

7. Related Work

There are numerous structured overlays in addition to Chord [15], Pastry [13] and Tapestry [17]. In particular, Pastry and Tapestry utilize prefix routing to incrementally route towards a destination key. Z-Ring is similar in routing by incrementally matching digits to the destination key. Each digit resolution chooses a member of a group of nodes sharing all other digits. Recent work has shown that Pastry maintenance traffic can be reduced to a small number of messages per second per node [1].

SWIM [4] and XRing [16] are efficient membership protocols that can be used by Z-Ring to maintain its routing groups. The work in [7] estimates the group membership maintenance cost and shows its feasibility for real systems. [10] discusses the full membership service, but focuses on finding a network size that allows a full membership service. Its calculation can be used by Z-Ring to determine the group size *G* in a particular system.

[7] and [8] study two-hop routing for large networks. However, their data structures are fixed, and cannot easily extend to larger networks. Z-Ring provides similar results when implemented with two-level membership table, but its design allows it to easily extend to larger networks adaptively.

8. Conclusion

Z-Ring uses efficient membership maintenance to support one or two-hop key-based routing in large dynamic networks. Our analysis and simulations show that Z-Ring provides efficient routing with very low maintenance overhead. We believe these membership maintenance techniques will allow us to deploy structured P2P protocols across previously unsupportable environments, including large scale networks, low-bandwidth hosts and networks with high churn.

References

[1] Castro, M., Costa, M. and Rowstron, A., “Performance and dependability of structured peer-to-peer overlays”, In *Proc. of DSN*, Florence Italy, June 2004.

[2] Chockler, G. V., Keidar, I., and Vitenberg, R., “Group communication specifications: A comprehensive study”, *ACM computing Surveys*, 33:4, pp.427 – 469, 2001.

[3] Dabek, F., Zhao, B. Y., Druschel, P., Kubiatowicz, J. and Stoica, I. “Towards a common API for structured P2P overlays”, In *Proc. of IPTPS*, Berkeley, CA, USA, April 2003.

[4] Das, A., Gupta, I., Motivala, A., "SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol", In *Proc. of DSN*, Washington DC, June, 2002

[5] Demers, A., et al, “Epidemic algorithms for replicated database maintenance”. Operating Systems Review, 22(1):8–32 (Jan. 1988).

[6] Golding, R., and Taylor, K., “Group membership in the epidemic style”, *Technical Report UCSC-CRL-92-13*, UCSC, May 1992.

[7] Gupta, A., Liskov, B., and Rodrigues, R., “Efficient routing for peer-to-peer overlays”, In *Proc. of NSDI*, San Francisco, CA, April, 2004.

[8] Gupta, I., Birman, K., Linga, P., Demers, A., van Renesse, R., “Kelips: building an efficient and stable P2P DHT through increased memory and background overhead”, In *Proc. of IPTPS*, Berkeley, CA, Feb. 2003.

[9] Labovitz, C., Ahuja, A., and Jahanian, F., “Experimental study of Internet stability and wide-area backbone failures”, University of Michigan Technical Report, CSE-TR-382-98, 1998.

[10] Lin, S., Pan, A., Zhang, Z., Guo, R. and Guo, Z., “WiDS: an Integrated Toolkit for Distributed System Development”, In *Proc of HotOS*, Santa Fe, NM, June 2005.

[11] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J., "Handling Churn in a DHT", In Proc. of the USENIX Annual Technical Conference, June 2004.

[12] Rodrigues, R. and Blake, C., "When Multi-Hop Peer-to-Peer Lookup Matters", In *Proc. of IPTPS,* San Diego, CA, USA, February, 2004.

[13] Rowstron, A. and Druschel, P., “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”. In *Proc. of ACM Middleware*, Heidelberg, Germany, November, 2001.

[14] Saroiu, S., Gummadi, P. K. and Gribble, S., “A measurement study of Peer-to-Peer File Sharing Systems,” In *Proc. of Multimedia Computing and Networking*, 2002.

[15] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H., “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, In *Proc. of SIGCOMM*, San Diego, CA, August 2001.

[16] Zhang, Z., Lian, Q., and Chen, Y., “XRing: a Robust and High-Performance P2P DHT”, *Microsoft Research Technical Report MSR-TR-2004-93*, 2004.

[17] Zhao, B. Y. et al, “Tapestry: A Resilient Global-scale Overlay for Service Deployment”, *IEEE Journal on Selected Areas in Communications*, January 2004, Vol. 22, No. 1.