

Gemini: Probabilistic Routing Algorithm in Structured P2P Overlay*

Ming Li, Jinfeng Hu, Haitao Dong, Dongsheng Wang, and Weimin Zheng

Computer Science and Technology Department, Tsinghua University, Beijing, China
{lim01, hujinfeng00, dht02}@mails.tsinghua.edu.cn
{wds, zwm-dcs}@tsinghua.edu.cn

Abstract. In this paper, we propose a new structured overlay protocol, which is more efficient and scalable than previous ones. We call it *Gemini*, because its routing table consists of two parts, one containing nodes with common prefix and the other containing nodes with common suffix. Gemini routes messages to their destinations in just 2 hops, with a very high probability of 0.99. When running in a p2p network of 5,000,000 peers, each Gemini node consumes only 6 messages per second for maintenance. This cost, as well as routing table size, varies as a $O(\sqrt{N})$ function to the overlay scale, so Gemini can also run well in an even larger environment.

Introduction

Currently, the number of concurrent users of Kazaa [3] has exceeded 3,000,000. When facing so large a scale, existing structured overlay networks all expose some weakness.

Pastry-like overlays (Tapestry [9], Chord [8], CAN [5], Pastry [6], Kademlia [4]) require too many hops, being a barrier to high efficient routing. A common 16-based Pastry network needs about $\log_{16} 3,000,000 \approx 5.38$ hops for message forwarding.

One-hop overlay [1] desires too much bandwidth for routing table maintenance. Nowadays, the average lifetime of peers is about one hour [7], that is to say, every node must receive information of $3,000,000 * 2 = 6,000,000$ member-changing events per hour. Each piece of information is about 200 bits, at least containing corresponding node's nodeId, ip and port. Thus the bandwidth cost is beyond 33.3 kbps, which is a heavy burden to most modem-linked peers.

To cope with the increasingly large scale of p2p applications, we design a new structured-overlay protocol, called *Gemini*, which simultaneously obtains high efficiency (2-hop routing in 99% probability) and low bandwidth overhead (6 messages per second in a 5,000,000-node network).

Similar with Pastry-like overlays', Gemini routing table only keeps a small fraction of all nodes. This causes that routing cannot be accomplished in just one hop. On the other hand, as one-hop overlay, Gemini uses multicast to update routing tables, which substantially enlarges routing table size in comparison to the manner of heartbeat. Obviously, larger routing table induces fewer routing hops.

Moreover, Gemini introduces probabilistic routing into structured overlay. How many hops a message passes before reaching its destination is not strictly determined.

* This work is granted by Chinese National Basic Research Priority Program (G1999032702), and National Science Foundation of China (60273006).

Although its expectation can be calculated, no fixed upper limit lies. This makes room for scalability design: we can raise the expectation of hops to keep a low overhead by adjusting system arguments.

Kelips [2] is another p2p protocol with $O(1)$ -hop lookup efficiency. We will compare Gemini and Kelips after a 5,000,000-node case study in section 2. Then in section 3 we give the formal description of Gemini and explain how to determine system arguments. Section 4 proposes some novel methods to gain even high scalability. Final conclusion is in section 5.

Case Study: 5,000,000-node Gemini

To make our presentation more tangible, we first describe how Gemini works in a network of 5,000,000 nodes. Formal description and analysis are left to the next section.

Like Pastry and Chord, Gemini assigns every node a *nodeId*. Commonly a *nodeId* is 64-bit long, being hash result of the node's IP address.

To a node, say node M^1 , we call the first 10 bits of M the node's *hat*, and the last 10 bits the node's *boot*.

Routing table. A Gemini node has a routing table consists of two parts, the first one containing all the nodes having the same cap with M , called *hatcase*, while the second one containing all the nodes having the same boot with M , called *bootcase*.

Obviously if two nodes have a same hat, their hatcases must also be the same. In this way, all the nodes are split into $2^{10}=1024$ groups according to their different haps. We call these groups *hat clubs*. Nodes within the same hat club are fully interconnected via their hatcases. Because *nodeIds* scatter evenly, every hat club contains about $5,000,000/1024 \approx 4883$ nodes. The *boot clubs* are defined symmetrically.

Routing. Every message has a destination address that is also 64-bit long. The first 10 bits of the address is also called its *hap*, and the unique hat club with this cap is called its *hat club*.

Slightly different with Pastry, root node of a message satisfies the following two conditions:

- 1) It is in the hat club of the destination address.
- 2) Its *nodeId* is numerically closest to the destination address among the whole club.

Considering that nodes in a hat club are fully interconnected, the basic task of message routing turns to forward the message to the correct hat club. After that the message will directly reach its root node in one hop.

In a probability of 99%, Gemini accomplishes this task using only one hop. This is because that a node's bootcase contains 4,883 items that is almost evenly distributed in the *nodeId* space whilst only 1,024 hat clubs exists.

When routing a message with destination address D , node M first checks whether M and D have the same cap. If so, M directly forwards it to the root node, which must be in M 's cap chest; otherwise M inspects its bootcase, tries to seek out a node who has D 's cap and forwards the message to it. The probability that M successfully finds the proper node from its bootcase is called *hit ratio*, which is determined by the size of M 's bootcase (noted Bc) and the number of hat clubs (noted H). Figure 1 shows

¹ Node M means this node has a *nodeId* M .

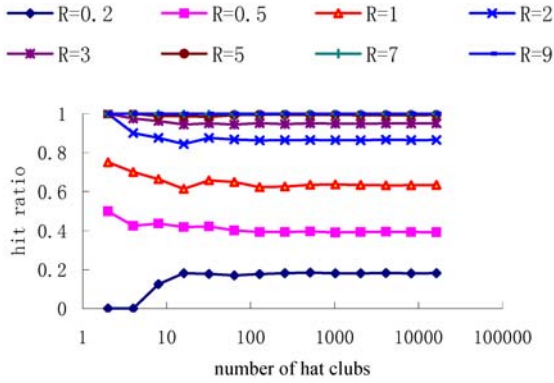


Fig. 1. Hit ratio vs. number of hat clubs. R is the ratio of bootcase size (B_c) to number of hat clubs (H). It shows that when H exceeds 50, hit ratio is almost a constant.

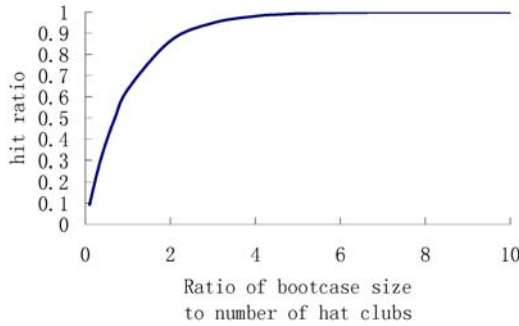


Fig. 2. Hit ratio vs. R , ratio of bootcase size to number of hat clubs. Here number of hat clubs fixes at 1024.

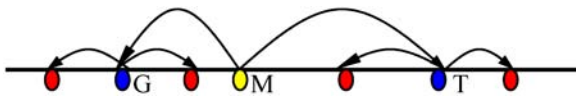


Fig. 3. An instance of tree-based multicast.

how hit ratio varies with respect to H under different ratios of $R=B_c/H$. We can see that when H exceeds 50, hit ratio is almost a constant value. For clearly illustrating, Figure 2 fixes H at 1024. In our case, $R=4883/1024 \approx 4.77$, so hit ratio is beyond 0.99. The equation supporting Figure 1 and 2 will be introduced in section 3.

If there is no node with D 's cap existing in M 's hatcase, M forwards the message randomly to a node in its hatcase whose boot is different with M .

Maintenance. Like one-hop overlay, Gemini uses broadcast (or say multicast) to maintain nodes' routing table. Note that hat clubs are independent to one another, and so do boot clubs, which allows us only to consider how to maintain nodes' hatcase/bootcase within a hat/boot club.

Nodes in a club are thought of as a sorted list in `nodeId` order. Every node sends heartbeats to the next node in the list every 10 seconds, while the last node sends heartbeats to the first one. If a node does not respond to heartbeats continuously for 3 times, the sender then multicasts its death to all the nodes in this club. The message used to inform other peers a member-changing event is called an event message.

Noting that nodes within a club are fully interconnected, we have many ways to realize such multicast of events. Here we only employ a simple tree-based multicast, illustrated in Figure 3. When a node *M* initiates a multicast, it sends the event to a node before it (say *G*) and another behind it (say *T*). Then alike, *G* and *T* each sends the event to two nodes, one ahead and the other behind. This procedure continues. At each step every node should ensure that once it sends the event to another node, there is no other node between them that has already received this event.

In this manner, except for the changing member's `nodeId`, ip address and port, an event message should additionally include `nodeId` of the first node before the receiver who has already received the message, as well as `nodeId` of the first such node behind it. Plus UDP header (64 bits) and ip header (160 bits), an event message will not exceed 500 bits.

If more efficient multicast is desired, the multicast tree can be modified to be 2^b -based. And if more reliable multicast is desired, response-redirect mechanism can be deployed, which will double the maintenance cost.

Maintenance cost. Assuming that the average lifetime of nodes is 1 hour, which is the statistic result from [6], all the items in the routing table have to be refreshed in a period of 1 hour. It means that on average every node receives $(4883+4883)*2=19532$ even messages per hour, namely 5.43 messages per second. Plus heartbeats and their responses, the total message count per second will not exceed 6, i.e., the bandwidth cost is lower than 3kpbs.

Joining of new node. When a new node *X* joins Gemini, it first contacts an existing Gemini node *E*. *E* picks a node *H* with *X*'s hat from its bootcase, as well as a node *B* with *X*'s boot from its hatcase. *X* collects its hatcase from *H* and bootcase from *B*. If *E* does not find appropriate *H* or *B*, it asks another node in its hat or bootcase for help.

Remarks. From this case we can see that 1) the reason why Gemini can route messages efficiently is that nodes in bootcase are sufficient to cover all the hat clubs in a very high probability; 2) the low maintenance cost comes from the maintaining manner of event multicast.

Comparison with Kelips. Another p2p protocol Kelips [2] also ensures that data lookups be resolved in $O(1)$ hops. There are four major differences between Gemini and Kelips:

First, Kelips divides nodes into groups only in one manner, while Gemini divides nodes in two symmetric manners.

Second, Kelips nodes explicitly keep pointers to other groups, while Gemini nodes employ items in the other dimension of groups as such pointers.

Third, in Gemini whether a message can successfully reach its hat club in one hop is in some probability. Reducing this probability can trade hop count for bandwidth overhead, making Gemini more scalable. (Detail discussion is in section 4) Kelips has no such character.

Fourth, Kelips is designed for file lookups, so it replicates file pointers all over a group. Gemini is just a substrate, leaving data operations to its application designers.

Formalization and Analysis

In this section, we describe how to determine the length of hat and boot in a given system environment, and then estimate routing table size and maintenance cost.

Assuming that in a Gemini overlay network of N nodes, every node has a hat with h bits and a boot with b bits. Then there are $H=2^h$ hat clubs and $B=2^b$ boot clubs totally. On average, each hat club contains $Hc=N/2^k$ nodes and each boot club contains $Bc=N/2^b$ nodes.

We hope that message routing can be accomplished in 2 hops, which is directly determined by hit ratio. To a certain node M , it has a bootcase with about Bc items. M wishes that these items could cover all the hat clubs. But unfortunately it is not the reality: there must be some hat clubs that are not pointed to by any items in M 's bootcase. We note number of such hat clubs as S . Expectation of S can be calculated as follows:

$$S = \sum_{k=0}^{H-1} k \times \frac{\binom{H}{k} \binom{Bc-1}{H-k-1}}{\binom{H+Bc-1}{H-1}}$$

They occupy a fraction of $s=S/H$ in the bootcase, so hit ratio is $hr = 1 - s = 1 - \frac{S}{H}$.

Figure 1 and 2 are based on this inequation. We can see from Figure 2 that when H is larger than 50, choosing R as 4.7 will ensure a hit ratio more than 0.99. So in common cases we demand that $R > R_0 = 4.7$. That is to say, $N/(2^{b \cdot 2^h}) > R_0$, namely

$$b + h \leq \log_2(N/R) \quad (1)$$

Next we estimate maintenance cost. Heartbeat cost is a fixed small value, so we simply ignore it. The substantial cost is for maintaining hatcase and bootcase, with a size of $Hc+Bc$. Assuming that nodes' average lifetime is L seconds, then each node triggers two events in a period of L seconds on average. So every node receives $2 \cdot (Hc+Bc)$ events every L seconds. If redundancy of employed multicast algorithm is f , then the number of messages a node receives per second is

$$m = (Hc + Bc) \times 2 \times f / L = \left(\frac{N}{2^h} + \frac{N}{2^b} \right) \times 2f / L$$

When $h = b$, m reaches its minimum

$$m_0 = \frac{2N}{2^{\frac{1}{2}(h+b)}} \times 2f = \frac{4N \cdot f}{2^b \cdot L} \quad (2)$$

² This problem is equivalent to the scenario randomly throwing r identical balls into n different boxes. The total number of possible results is $\binom{r+n-1}{n-1}$.

³ Choosing h and b that holds $|h-b|=1$ is also feasible, but analysis will be more complicated. Considering that its result is of only slight difference with the scenario where $h=b$, we omitted it in this paper.

Simultaneously considering (1) and (2), we should set $b = h = \left\lfloor \frac{\log_2(N/R)}{2} \right\rfloor$ to ensure a 2-hop-routing probability larger than 0.99 using a minimal maintenance cost. Then we can get the following results:

a) Routing table size R_{size} satisfies

$$2\sqrt{N \cdot R} \leq R_{size} < 4\sqrt{N \cdot R} \tag{3}$$

b) Maintenance cost m_0 satisfies

$$\frac{\sqrt{N \cdot R} \times 4f}{L} \leq m_0 < \frac{\sqrt{N \cdot R} \times 8f}{L} \tag{4}$$

Figure 4 show the variation of b , R_{size} and m_0 as functions of N using $f=1$ and $L=3600$.

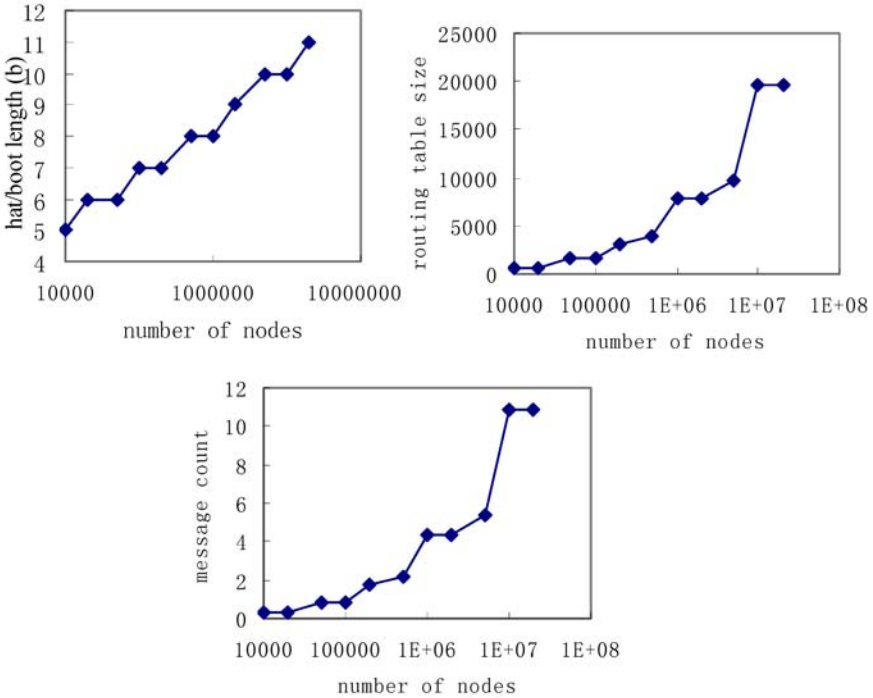


Fig. 4. Hat length vs. N , Routing table size vs. N and Routing table size vs. N .

Scalability

Inequation (3) and (4) show that a Gemini node has a routing table with $O(\sqrt{N})$ items and consumes $O(\sqrt{N})$ bandwidth to maintain it. This allows for a good scalability of Gemini.

In addition, when the maintenance cost is not acceptable by peers, Gemini also can trade hops for bandwidth consumption like other overlay protocols. To illustrate it, we put Gemini into a stricter environment where $N=10,000,000$, $f=2$ and $L=2,400$. Using inequation (4) we know that if keeping 2-hop routing, a node should receive at least 22 messages per second.

Gemini can reduce this cost by decreasing R , which will raise b and h , increase H , shrink Bc , and reduce hr . Expectation of hop count can be calculated as:

$$\text{hops} = 2 \cdot hr + 3 \cdot (1 - hr) \cdot hr + \dots = 1 + \frac{1}{hr}$$

Even when hr drops to 0.25, average hop count only rises to 5. How message cost and hop count vary in relation to R is shown in Figure 5.

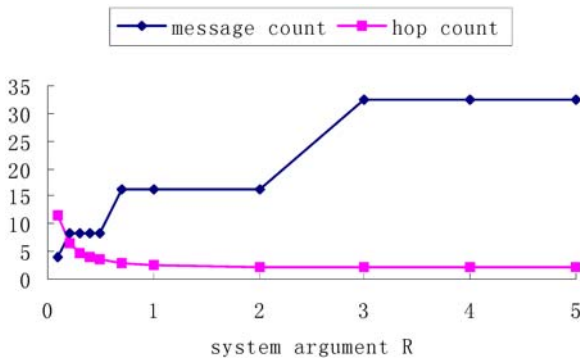


Fig. 5. Message cost and hop count in relation to the argument R , where $N=10,000,000$, $f=2$ and $L=2,400$.

Conclusion

Apparently, in structured overlay design, larger nodes' routing table is, fewer routing hops are needed. Traditional Pastry-like protocols can only afford a small number of entries for the reason that they use heartbeats to maintain routing tables. Recent one-hop overlay employs broadcast for maintaining, significantly decreasing bandwidth cost per entry. But letting every node keep a panoramic sight does burden most weak nodes.

References

1. Anjali Gupta, Barbara Liskov, Rodrigo Rodrigues. One Hop Lookups for Peer-to-Peer Overlays. HOTOS IX. May 2003.
2. Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse. Kelips: building an efficient and stable P2P DHT through increased memory and background overhead. IPTPS '03. February 2003.
3. Kazaa. <http://www.kazaa.com>. November 2003.
4. Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. IPTPS '02. March 2002.

5. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. SIGCOMM 2001. August 2001.
6. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Middleware 2001. November 2001.
7. Saroiu, S., Gummadi, P. K., and Gribble, S. D. A Measurement Study of Peer-to-Peer File Sharing Systems. MMCN '02. January 2002.
8. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM 2001. August 2001.
9. Ben Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley. April 2001.