CBRBrain: Provide Content Based Routing Service Over Internet Backbone

Wen-Zhan Song * Xiang-Yang Li *

***Abstract*— Peer-to-peer(P2P) networking has come to a family of technologies and techniques for organizing distributed applications, which takes advantage of resources available at the Internet edges. In this paper, we propose an innovative P2P system architecture, called *CBRBrain*, to implement the content based routing (CBR) service over the backbone routers instead of at the terminal hosts. Hence CBRBrain avoids some drawbacks in previous P2P systems and significantly improves the efficiency and security. Data locating process is easily implemented on CBRBrain by associating a hashed key with each data item and storing the (key, address) pair in routers. The cost for topology update is neglectful since the routers is almost static in Internet and the topology is not affected by the frequent joining and leaving of hosts at all. The traffic generated by CBRBrain system over Internet is also expected to be significantly smaller compared with other P2P systems. The CBRBrain backbone adopts the self-routing structure *de Bruijn graph* as the topology, which has a number of preferred properties such as bounded degrees, low diameters and fault tolerance. As an illustration, we describe the mechanism of P2P file sharing application under *CBRBrain* architecture.**

Our work is a first step to provide an intelligent backbone as the core of the next Internet. An implicit contribution of CBRBrain is to facilitate the emergence of various intelligent applications over Internet, besides P2P file sharing.

***Index Terms*— system design, P2P networking, content based routing, de Bruijn graph, bounded degree, low diameter.**

I. Introduction

The term P2P comes to the force with the rise and fall of Napster[1]. Although there are prior systems in this evolutionary phase of distributed computing, P2P system first emerges as a significant social and technical phenomenon right after the deployment of Napster. Currently most popular P2P systems are aiming at the file sharing applications, where files are stored at user hosts rather than at a central server in the traditional client/server model.

The first widely used and well-known P2P system is Napster, although it is not a pure decentralized P2P system. Napster[1] uses a central server to store the index of all the files available within its community. Assume that a user wants to download a music file online, he needs first to browse the directory on the central server to get the IP-address of the host that stores the requested music. The user then contacts the target host and downloads the desired music files directly. The idea behind Napster is simple and straightforward but very successful, however, the central server is vulnerable to attacks, which intrigues massive research on the decentralized P2P system. Napster has a revolutionary impact on Internet applications due to its simple design approach: after the initial search for material, clients connect with each other and exchange data directly.

Gnutella[2] goes a step further than Napster and decentralizes the file location process as well. Users in a Gnutella network self-organize into an application-level mesh on which requests for a file are flooded within a certain number of hops. Flooding on every request is clearly not efficient and scalable, and thus it has to be curtailed at some point. Consequently, flooding may fail to find a content that is actually in the system.

Several P2P schemes were proposed recently to replace the flooding based query mechanism with smarter routing and/or group communication mechanism, e.g., Chord [3], CAN [4], Viceroy [5], Tapestry [6], Pastry [7], D2B [8]. These P2P schemes build a *distributed hash table* (DHT) on top of the overlay to provide efficient querying. In DHTs, keys are mapped into a keyspace and assigned to all participated hosts. Each host needs to take care of the related information of its assigned keyspace. Then, a lookup request for a key simply means finding the host which is responsible for the key's hash value. These systems can be categorized into the second generation P2P system, *DHT-based P2P system*. So far, much attention has been given to constructing large networks with some nice properties such as bounded degrees and low diameters.

However, prior art on decentralized P2P system falls into one extreme: the system is totally decentralized to the hosts at the Internet edge, and each host has to act as a router to select proper next-hop neighbors and relay the messages. Although these DHT-based P2P systems enjoy several technical advantage over the first generation of P2P systems, they still suffer some drawbacks:

1) **Bottleneck caused by the low-resource hosts.** Each host in these P2P systems acts the same role. However, the network connections and computing resources of different hosts may have big difference. Hosts with low resources become the bottleneck in the routing chains, hence slow down the whole system.
2) **High maintenance cost on the dynamic topology.** The number of end hosts in Internet is enormous and they could join and leave a P2P system frequently, which in turn causes the P2P topology changing frequently. Consequently, the cost for dynamic maintenance is high and this will cause a big performance oscillation of the system.
3) **Overload on hosts.** In a pure decentralized P2P system, each edge host needs to forward the data for others. In practice, most users are consumers instead of providers in a file sharing P2P system. Counting those consumer hosts (without sharing files) into routing chain is not helpful at

* Department of Computer Science, Illinois Institute of Technology, 10 W. 31st Street, Chicago, IL 60616, USA. Email *songwen@iit.edu, xli@cs.iit.edu.*
The work of the second author is partially supported by NSF CCR-0311174.

all, since it will increase unnecessary path hops. On the other hand, the consumers are irritated by such resource overload, because they even cannot control its own resources after they join those P2P networks.

4) **Security problems on routing chains.** In those DHT-based P2P systems, each user host keeps a lookup table for a subset of keys. An attacker can break the routing chain easily by joining the system then falsifying its lookup table. In addition, since the transactions are performed among individual hosts, it becomes difficult to implement sophisticated security mechanisms.
5) **Non-cooperative behavior of selfish hosts.** Traditionally, most current P2P systems assume that the end user are either *correct/obedient* or *faulty* (also called adversarial). However, in P2P systems, the end user may not follow the designed protocol, e.g., it may deny relaying messages for other nodes since the user wants to save its computing and network resources. We could assume that the individual users are *rational*. The rational users respond to well-defined incentives and will deviate from the protocol if it does not improve its gain.

In this paper, we propose an innovative P2P system architecture, called *CBRBrain*, which implements the content based routing (CBR) service for P2P applications over the Internet backbone instead of at the edges. Data locating process is easily implemented on CBRBrain by associating a hashed key with each data item and storing the (key, address) pair in routers, so edge hosts are not involved in the routing chains. Consequently, the cost for topology update is neglectful since the routers is almost static in Internet and the topology is not affected by the frequent joining and leaving of hosts at all. The traffic generated by CBRBrain system over Internet is also expected to be significantly smaller compared with other P2P systems. The CBRBrain backbone is built upon the self-routing structure *de Bruijn graph*, which has a variety of properties such as bounded degrees, low diameters and fault tolerance. As a running example, we describe the implementation of a P2P file sharing application under *CBRBrain* architecture. By constructing the CBRBrain over the routers instead of hosts, we avoid the limitations of previous P2P systems.

The rest of this paper is organized as follows. In Section II, we describe our innovative P2P architecture CBRBrain. In Section III, we give an introduction on de Bruijn graph that is used to build the backbone of CBRBrain system and show how to maintain the backbone topology. In Section IV, we discuss in detail the implementation of P2P file sharing over CBRBrain. In SectionV, we show how different CBRBrain networks can work independently and cooperatively in the Internet. Finally, we conclude our paper in Section VI.

II. The Architecture of CBRBrain System

The design of P2P system can gain useful lessons from previous decentralized system in Internet evolution history. Usenet is an instructive example of the evolution of a decentralized system. Usenet propagation is symmetric: hosts share traffic. But because of the high cost of keeping a full news feed, in practice there is a backbone of hosts that carry all of the traffic and serve it to a large number of "leaf nodes" whose role is mostly to receive news. Within Usenet, there was a natural trend toward making traffic propagation hierarchical, even though the underlying protocols do not demand. This form of "soft centralization" may prove to be economic for many peer-to-peer systems with high-cost data transmission.

CBRBrain system builds a distributed logical network overlapping Internet backbone, allowing the discovery of data and/or resources identified by keys in Internet. In the architecture, we still kept the content located at the edge of Internet, while promoting the duty of content based routing(CBR) to the Internet backbone.

The backbone of CBRBrain system is a content addressable network, which can be described by a pair (K,G) where K is a set of keys and $G = (V, E)$ is a logical graph or topology. The set K is generated by hosts who hash each shared content into a value, hereafter called *key*, and publish it to the backbone. Each node u in G is assigned a subset of keys K_u such that $\cup_{u \in V} K_u = K$. In practice, node u needs to store a lookup table which contains necessary information related to each key $k \in K_u$, such as the address of the host who published the key and owns the content. The assignment of keys to nodes is performed by mapping both keys and nodes' labels to a real domain, then the key/value pairs are assigned to the *closest* server. Here the term *closest* has different meaning in different system, for instance, Chord[3] assigns a key k to the first node whose identifier is equal to or follows k in the identifier space. CAN[4] assign a key k to the node who owns the zone. In D2B[8], a key k is assigned to the node whose label is the prefix of k. As opposed to other networks, routing in a content addressable network is not performed according to the destination address, but according to the content key. More precisely, no one can know the address of the *closest* server in advance. It is eventually found out by content routing and key matching.

In CBRBrain system, IP routers act as content ROUTERS in finding the best route from one point to another, and the user host will not participate in any intermediate routing and forwarding. Figure 1 illustrates such an architecture. The region inside the cloud represents the CBRBrain backbone which overlays the backbone of Internet. The set of routers construct a self-routing topology, whose detail will be addressed in Section III. The end hosts connect to the network through those gateway routers inside the backbone. Notice that, we do not force all routers to participate into CBRBrain network, as will see later, the uninvolved routers are transparent to the system like network cables; and any dedicated host can also act as a *router* in the system if it is authorized by the system coordinator. For simplicity of presentation, here before and after, the *router* always represent the backbone router or dedicated host who has joined the routing chain by authorization.

By constructing the CBRBrain over the routers instead of hosts, we can avoid the limitations of prior art described in Section I. Firstly, the host with low bandwidth and low computing resource will not affect the routing performance of the P2P system since it is not in the routing chain. Secondly, the routers rarely leave the networking, so the topology is almost static once it is configured. Thirdly, the joining and leaving of individual host is handled by its corresponding router and is

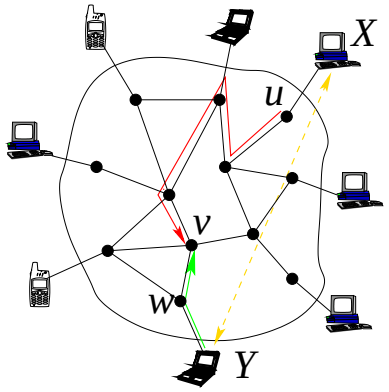

Fig. 1. The architecture of CBRBrain system.

transparent to the system if it did not publish contents. In addition, the routing is performed on routers, so each host member bares no maintenance and routing cost. Fourthly, we also get rid of the non-cooperative problem of user hosts by letting only the routers perform the content based routing, where the routers are assumed to be obedient. In addition, we implement the look up service on the Internet backbone instead of edges, which is expected to improve the search efficiency and to reduce the traffic and workload of Internet.

From the viewpoint of users, CBRBrain network works like a central server, where user could query to and get response from, though the backbone is formed by many routers and the content location is actually decentralized to individual user hosts. From the viewpoint of Internet, CBRBrain backbone is an overlayed logical network over Internet backbone, which provides additional service, *Content Based Routing*, to sustain various peer-to-peer applications and other intelligent services in the future.

For illustration, we briefly discuss how to retrieve a file in P2P file sharing application under the CBRBrain system. Notice that the CBRBrain architecture itself is not restricted to the file sharing. Figure 1 illustrates an example that follows:

1) A host X inquires the CBRBrain system about a file Ω stored in Internet. Host X first uses the globally predefined DHT function to map Ω to a key k, then sends it to the gateway router u.
2) The CBRBrain backbone performs the content based routing service, which will be described in Section III, and finds the target router v who has the key k in its lookup table. The router v then finds the corresponding IP address(es) of the target host Y if it exists. There are two options here: 1) the router could then retrieve the content and feed it to the requesting host, or 2) the router gives the IP address of Y to the requesting host and let it retrieve the content. The first approach makes the targeting host anonymous to the requesting host, while the second approach alleviates the burden of the router.

III. The Construction of CBRBrain Backbone

In building DHTs, the system needs to design the overlay topology carefully to ensure the efficiency of publish/retrieve. Several objectives should be met in designing the backbone for a DHT-based P2P system [8]. Firstly, all nodes in the system are mutually reachable, i.e., the topology is connected. Secondly, keys are evenly distributed among nodes with high probability. Thirdly, lookups are performed on a key-basis, i.e., the route from the query host to a supplier host is set up according to the knowledge provided by the key of the resource only. Fourthly, the lookup latency should be small. From any given query node to reach a node responsible for any given key, the lookup path must be short, i.e., the constructed topology has low diameter. Fifthly, the traffic load incurred by lookups routing through the system should be evenly distributed among nodes, i.e., the congestion is evenly distributed. Last, but not least important, the redistribution of keys due to node's leaving or joining must be fast, i.e., the update can be performed efficiently.

CAN [4] uses a d-dimensional Cartesian coordinate space to implement a distributed hash table that maps keys onto values. Thus, each node maintains $O(d)$ states and the lookup cost is $O(dN^{1/d})$. Chord [3] is based on the a ring topology with long-distance hop pointers (hypercube). Both the expected node degree and diameter are $O(\log n)$. Viceroy [5] uses the same key set as Chord, but adopts the butterfly graph as the underlying topology. The simplified version of Viceroy has expected degree $O(1)$ and diameter $O(\log n)$. Recent work in D2B [8] proposed a novel topology based on de Bruijn graph. It has expected degree $O(1)$ and diameter $O(\log n)$ in high probability. All these systems build the topology on the set of individual hosts, i.e., the Internet edge. Table I summarizes the comparison of expected performance measures.

TABLE I

Performance measurements of different topologies.

	Update	Lookup	Congestion
CAN	$O(d)$	$O(dn^{1/d})$	$O(dn^{1/d-1})$
Chord	$O(\log n)$	$O(\log n)$	$O(\frac{\log n}{n})$
Tapestry	$O(\frac{d \log n}{\log d})$	$O(\frac{\log n}{\log d})$	$O(\frac{\log n}{n})$
Viceroy	$O(1)$	$O(\log n)$	$O(\frac{\log n}{n})$
D2B	$O(1)$	$O(\log n)$	$O(\frac{\log n}{n})$

The CBRBrain system does **not** put constraint on the underlying topology over its backbone. However, it is preferred to be a self-routing structure with low degrees, low congestions, and low diameters. Under the careful analysis on different graphs and topologies and the inspiration from the work in [8], in this paper, we recommend the underlying topology to be built upon *de Bruijn graph* since it has all nice properties mentioned above and can be easily constructed. For easy presentation, we first review the property of de Bruijn graph here.

A. de Bruijn Graph

The de Bruijn graph [10] $B(d,k)$ is the directed graph whose nodes are all strings of length k on the alphabet $\{0,\cdots,d-1\}$, and there is an edge from any node $x_1x_2\cdots x_k$ to node $x_2\cdots x_ky$ for any $y \in \{0,\cdots,d-1\}$, which enables self-routing. Figure 2 illustrates $B(2,3)$. Routing from $x_1x_2\cdots x_k$ to $y_1y_2\cdots y_k$ is achieved by the following route $x_1x_2\cdots x_k \rightarrow x_2\cdots x_ky_1 \rightarrow x_3\cdots x_ky_1y_2 \rightarrow \cdots \rightarrow x_ky_1\cdots y_{k-1} \rightarrow y_1\cdots y_k$. A shorter route is obtained by looking for the

longest sequence that is suffix of $x_1x_2\cdots x_k$ and prefix of $y_1y_2\cdots y_k$. Suppose that $x_i\cdots x_k = y_1\cdots y_{k-i+1}$, then the shortest path from node $x_1x_2\cdots x_k$ to node $y_1y_2\cdots y_k$ is $x_1\cdots x_k \rightarrow x_2\cdots x_ky_{k-i+2} \rightarrow x_3\cdots x_ky_{k-i+2}y_{k-i+3} \rightarrow \cdots \rightarrow x_{i-1}\cdots x_ky_{k-i+2}\cdots y_{k-1} \rightarrow y_1\cdots y_k$. The route from any node to any other node is at most k hops. That is to say, the graph $B(d,k)$ with $n = d^k$ nodes has a diameter $k = \log_d n$.

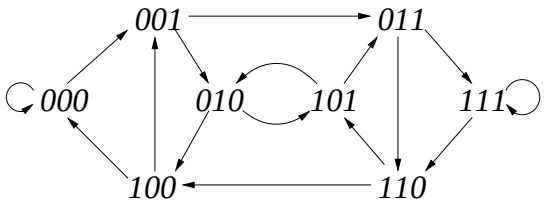

Fig. 2. The de Bruijn graph $B(2,3)$.

The classical de Bruijn graph is *balanced* in the sense that all node labels have the same length. The de Bruijn graph can be generalized to any set of vertices whose labels form a universal prefix set [8]. A universal prefix set is a set S of labels on an alphabet Σ such that, for any infinite word $w \in \Sigma^\star$, there is a *unique* word in S, which is a prefix of w. A generalized de Bruijn graph is *pseudo-balanced* if the lengths of the labels are different by at most one. In geometry viewpoint, the node labels in a pseudo-balanced de Bruijn graph correspond to the leaf node labels in a *full* binary tree, in which the depth difference of any two leaf nodes is at most one and any non-leaf node has 2 children. Figure 3 illustrates the correspondence between a pseudo-balanced de Bruijn graph and a full binary tree. In the figure, the pseudo-balanced de Bruijn graph is defined on the leaf nodes with directed edges.

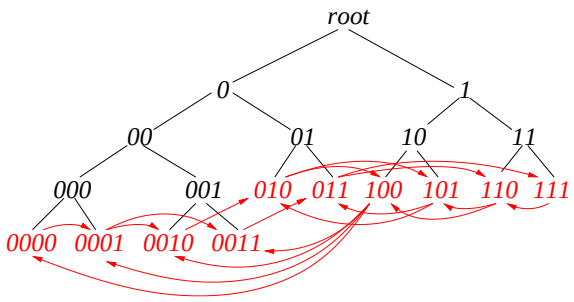

Fig. 3. The correspondence between full binary tree and pseudo-balanced de Bruijn graph.

For simplicity, we still denote a pseudo-balanced de Bruijn graph on alphabet $\{0,1\}$ by $B(2,k)$ if the node labels have length at least k bits and at most $k+1$ bits. We will only consider pseudo-balanced de Bruijn graph. In $B(2,k)$, each node has 2 in-neighbors and at most 4 out-neighbors. To route a packet from a node u with label $x_1x_2\cdots x_{s-1}x_s$ to another node v with label $y_1y_2\cdots y_{t-1}y_t$, where $s,t \in [k,k+1]$. Node u will forward the packet to its neighbor node with label $x_2\cdots x_{s-1}x_s$, or $x_2\cdots x_{s-1}x_sy_1$, or $x_2\cdots x_{s-1}x_sy_1y_2$. Notice that since the labels of the nodes are a universal prefix set, we know that *exactly* one of these three labels does exist. The following nodes keep forwarding the packet similarly until it reaches node v. Consequently, the diameter of a pseudo-balanced de Bruijn graph is still $O(\log n)$.

B. CBRBrain Backbone Construction

The routers in CBRBrain backbone form a pseudo-balanced de Bruijn graph as its logical topology. Each router is assigned a label $x_1x_2\cdots x_t$ with $t = k$ or $k+1$ from the de Bruijn graph. To enable the data locating and content based routing service in CBRBrain network, each serving router keeps two tables: *label-routing table* and *lookup table*. The *label-routing table* records the label and IP-address of all logical out-neighbors in de Bruijn graph, and the *lookup table* records those (key, address) pairs whose key has its label as prefix and other additional information such as the network bandwidth or the computing resource at that host.

For instance, in Figure 3, node 100 has 4 out-neighbors 0000,0001, 0010 and 0011, its *label-routing table* is illustrated in Table II. A query with a key 0011...101 will simply be forwarded to the router 192.47.152.27, whose label matches the key's prefix. Its *lookup table* is illustrated in Table III. Hosts 172.56.185.29, 175.45.182.11 and 173.12.257.32 all own same content-key but have different network bandwidth. To reply a query with key 100...0011, node 100 could choose the host 172.56.185.29 with a higher probability if our strategy is to balance the traffic flow. In practice, we may use some sophisticated technique to store (key, address) pairs to improve the search efficiency on the table, which is out of the scope of this paper.

TABLE II

THE LABEL-ROUTING TABLE IN THE ROUTER 100.

Neighbor-Label	IP-Address
0000	147.29.215.92
0001	216.47.152.81
0010	225.98.151.26
0011	192.47.152.27

TABLE III

THE LOOKUP TABLE IN THE ROUTER 100.

Content-Key	IP-Address	Bandwidth
100...0001	216.47.152.88	64K
100...0011	172.56.185.29	100M
	175.45.182.11	10M
	173.12.257.32	64K
100...0100	185.45.181.27	10M
⋮	⋮	⋮
100...1110	175.45.182.11	10M
	173.12.257.35	64K

The adoption of balanced or pseudo-balanced binary de Bruijn graph enjoys at least the following nice properties:

1) Keys can be uniformly distributed in all routers in CBRBrain backbone with high probability, since the length difference of node labels is at most one.
2) The size of the *label routing table* in each router is at most 4 due to the number of out-neighbors is at most 4. That is to say, each router only keeps at most 4 live links, hence the workload for maintaining links is small and the

router's joining or leaving only affects 2 in-neighbors and at most 4 out-neighbors.
3) Reduce the congestion in whole network since the workload is uniformly assigned in high probability.
4) The diameter of the topology is $O(\log n)$ where n is the number of serving routers in CBRBrain system.

In CBRBrain system, the backbone topology is formed by the routers, so the topology is relative stable and static. In addition, the number of routers is limited and they usually are configured before using. We suggest to use a *certificate server*[1] to build and maintain the logical topology, i.e., assign/change/delete labels due to routers joining/leaving, though it is possible to maintain a dynamic de Bruijn graph[8], [11] in a distributed manner. The reasons are follows:

1) Security control. Attacker can join in CBRBrain network as a falsified router and hence break the P2P system by malicious routing or denying to relay messages. Using a certificate server to verify each joining/leaving router could diminish this kind of attack.
2) Easy to maintain a pseudo-balanced de Bruijn graph. As mentioned in Section III-A, a pseudo-balanced de Bruijn graph enjoys many nice properties, however, it is expensive to be maintained in a distributed way.
3) To achieve high efficiency and reduce communication cost. According to the experiment in [12], 55% of all traffic generated by early Gnutella network is for the topology maintenance, though it is decreased to 8% in the improved version. Most DHT-based P2P systems have strict constraint on topology, so the cost for topology maintenance is probably even higher. A certificate server can update the topology faster with less communication cost.

The logical topology and router labels is updated if and only if routers joining or leaving, which is not affected by any user hosts at all. In the certificate server, the full binary tree[2] corresponding to the current topology is recorded, so are the (label, address) pairs of those routers. Let n be the number of routers currently in the CBRBrain backbone, then the label length should be in the range $[m, m+1]$ where $m = \lfloor \log_2 n \rfloor$. The backbone construction and maintenance are controlled by the certification server as follows:

1) Router joining. The joining router u first sends a request to the certificate server for authorization. If it passes verification, the server first splits the smallest m-bits label $x_1...x_m$ into two new ones $x_1...x_m0$ and $x_1...x_m1$; then returns the new label $x_1...x_m1$ to the joining router u and records its IP-address. At the same time, the server notifies the router v, having the old label $x_1...x_m$, its new label $x_1...x_m0$. After that, routers u and v and their logical neighbors update their *label-routing table*s according to the connection rules in de Bruijn graph. Router v then moves those (key, address) pairs with key prefix $x_1...x_m1$ in its *lookup table* to router u.
2) Router leaving. The leaving router u sends a request to the certificate server. After verification, the system performs the update as follows:
   a) If u has a $(m+1)$-bits label $x_1...x_mx_{m+1}$, then the server simply asks the router v with label $x_1...x_m\overline{x_{m+1}}$ to shrink to $x_1x_2...x_m$. After that, routers u and v and their logical neighbors update their *label-routing table*s according to the connection rules in de Bruijn graph. Router u moves all (key, address) pairs in its *lookup table* to router v then leaves safely.
   b) If u has a m-bits label. Assume there exists a $(m+1)$-bits label in the tree. Otherwise, it is similar to the case a). The server then asks the router v with the largest $(m+1)$-bits label $y_1...y_m1$ to replace the position left by u, and requests the router w to replace its label $y_1...y_m0$ by $y_1...y_m$. After that, routers u, v, w and their logical neighbors update their *label-routing table*s according to the connection rules in de Bruijn graph. Router v moves all (key, address) pairs in its *lookup table* to router w. Router u moves all (key, address) pairs to router v and leaves safely.

   In case of router leaving due to power off, the detection mechanism can be implemented by the periodical PING from the certificate server, or from its in-neighbors who then reports the connection loss to the server.

IV. P2P FILE SHARING APPLICATION IN CBRBRAIN SYSTEM

In this section, we describe an implementation of the P2P file sharing application in our system. The other P2P applications or intelligent Internet applications could be developed similarly under this architecture. In CBRBrain system, the joining or leaving of hosts is transparent to the system, unless they did one of the following: (1) publish content (2) retrieve content (3) leave the system after publishing something.

A. User Host Publishes and Retrieves Content

In CBRBrain system, the publishing and retrieving operation by a user host works as follows:

1) Publish Content
   a) A host X hashes its sharing file into a m-bits key $k = c_1c_2...c_m$ and sets its sharable bandwidth B, then sends a message PUBLISH(k,$IP(X)$,B) to the nearest gateway router u that participates in the CBRBrain backbone.
   b) Content based routing service over backbone eventually forwards the message to the router v whose label matches the prefix of k, as shown in Figure 1.
   c) Router v inserts the (key, address) pair (k,$IP(X)$,B) to its lookup table; see Figure III.
2) Retrieve Content
   a) A host Y hashes its query into a m-bits key $k = c_1c_2...c_m$, then sends a message QUERY(k,$IP(Y)$) to the nearest gateway router w that participates in the CBRBrain backbone.

[1] In practice, we may use a group of servers instead of a single server to increase fault tolerance and security, which is beyond the scope of the paper.
[2] Actually, the server only need store the leaf node labels in a list.

b) Content based routing service over backbone eventually forwards the message to the router v whose label matches the prefix of k.
c) Router v searches k thoroughly in its lookup table. If found, then return [3] one or all of the corresponding IP-address(es) to the host Y depending on the strategy. Otherwise it returns a FALSE message.

B. User Host Joins and Leaves

The user host's joining and leaving will not affect the CBRBrain network topology at all, unless the leaving host has published content in the network. Hence our architecture can significantly reduce the computation and communication cost. The leaving host X which has content published should first notify the system before it leaves. However, if it leaves impolitely, the system will detect it once a report on failed retrieve or failed periodical PING is received by the *hosting* router. When a host X leaves and it already published some contents, the follows will be performed:

1) For each published content, host X sends a message LEAVE(k, $IP(X)$) to the nearest router u.
2) Content based routing service over backbone eventually forwards the message to the router v whose label matches the prefix of k.
3) Router v simply deletes the matching (key, address) pair from its lookup table.

V. THE SCALABILITY OF CBRBRAIN NETWORK

The CBRBrain system also enjoys good scalability. Several CBRBrain systems may connect with each other without conflict because one CBRBrain system could act like a user host to retrieve content from another CBRBrain system. They would even be overlapped in the Internet physically. Figure 4 illustrates such an idea. If the CBRBrain system A can not find the content for a special query, then it forwards the query to some CBRBrain system B through the connection between their gateway routers u and v. The router u works as a host to retrieve content from the system B.

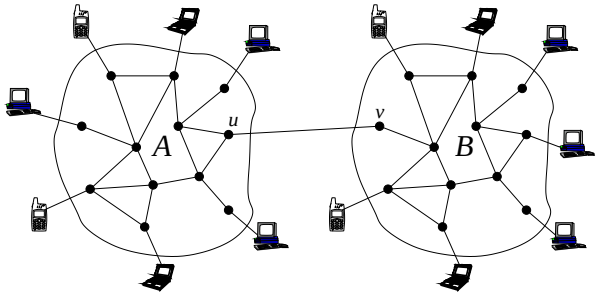

Fig. 4. The scalability of CBRBrain model.

The clustering technique is expected to significantly improve the search efficiency and reduce the cost in practice. Users are usually interested with different contents. Hence it is better to build different subject-oriented CBRBrain systems and then connect them with each other. A user in one CBRBrain system can query any content in other CBRBrain systems without actually joining them as a client since all CBRBrain systems can work together cooperatively.

[3] Again, the router v may simply return the IP address or retrieve content then feed back, as discussed in Section II.

VI. CONCLUSION

In this paper, we propose an hierarchical architecture, called *CBRBrain*, which implements the content based routing (CBR) service for P2P applications over the Internet backbone instead of at the edges. The concept P2P itself is far more significant than the P2P file sharing applications. P2P networking approaches the dream that "Internet is a big intelligent computer". The brain of the "Computer" needs to be integrated with efficient resource locating and sharing functions besides IP routing. The goal of our CBRBrain architecture is to facilitate the emergence of various intelligent applications over Internet, besides P2P file sharing.

The architecture of the Internet has caused the largest transfer of power from organizations to individuals the world has ever seen, and it is only getting started. Millions of passive consumers are replaced by millions of one-person media channels. This is not to say that all contents are going to the edges of the Internet, or that every user is going to be an enthusiastic media outlet. But enough consumers will become providers as well to blur present distinctions between producer and consumer. This social shift will make the next generation of the Internet, currently being assembled, a place with greater space for individual contributions that people accustomed to the current split between client and server, and therefore provider and consumer, had ever imagined.

REFERENCES

[1] NAPSTER, "http://www.napster.com," .
[2] GNUTELLA, "http://www.gnutella.com," .
[3] I.Stoica, R.Morris, D.Karger, M.Kaashoek, and H.Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, 2001.
[4] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, 2001, pp. 167–172.
[5] D.Malkhi, M.Naor, and D.Ratajczak, "Viceroy: a scalable and dynamic lookup network," in *21st ACM Symposium on Principles of Distributed Computing(PODC)*, 2002.
[6] B.Zhao, J.Kubiatowicz, and A.Joseph, "Tapstry: an infrastructure for fault-tolerant widearea location and routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, 2001.
[7] A.Rowstron and P.Druschel, "Pastry: a scalable and dynamic lookup network," in *21st ACM Symp. on Principles of Distributed Computing (PODC)*, 2002.
[8] Pierre Fraigniaud and Philippe Gauron, "The content-addressable network d2b," in *Technical Report TR-LRI-1349 (also appeared in 22nd ACM Symp. on Principles of Distributed Computing (PODC))*, 2003.
[9] P.V.Mockapetris and K.J.Dunlap, "Development of the domain name system," in *ACM SIGCOMM*, 1988, pp. 123–133.
[10] N. de Bruijn, "A combinatorial problem," in *Koninklijke Nederlandse Academie van Wetenschappen*, 1946, 49, pp. 758–764.
[11] Wen-Zhan Song, Xiang-Yang Li, Yu Wang, and Weizhao Wang, "db-blue: Low diameter and self-routing bluetooth scatternet," in *ACM Dialm-POMC Joint Workshop on Foundations of Mobile Computing*, 2003.
[12] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.